Proceedings of ICTCS2013
14th Italian Conference on Theoretical Computer Science

9-11 September 2013
Palermo (Italy)
Facoltà di Scienze MM FF NN
Preface

The Italian Conference on Theoretical Computer Science (ICTCS) is the conference of the Italian Chapter of the European Association for Theoretical Computer Science (EATCS). The Conference, besides being a forum of exchange of ideas, provides the ideal environment where junior researchers and Ph.D. students meet senior researchers. ICTCS is also open to researchers from outside Italy, who are welcome to submit papers and attend the Conference.


In this edition, sponsored by the University of Palermo and by the Italian Chapter of the EATCS, 31 papers have been accepted for presentation, covering all areas of Theoretical Computer Science. The participants came from several Italian Universities as well as foreign institutions, such as King’s College London (UK), Université de Caen Basse Normandie (France), Ural State University (Russia), Northeastern University Boston (USA), University of California Los Angeles (USA) and University of Porto (Portugal).

We would like to express our thanks to the three invited speakers: Pierpaolo Degano (University of Pisa), Giuseppe F. Italiano (University of Rome Tor Vergata) and Luca Moscardelli (University of Chieti–Pescara), the recipient of the Young Researcher in Theoretical Computer Science Award 2013, conferred by the Italian Chapter of the EATCS.

Furthermore, at ICTCS 2013 the two recipients of the Best Ph.D. Thesis in Theoretical Computer Science Award 2013, assigned by the Italian Chapter of the EATCS, Jacopo Mauro (University of Bologna) and Alessandra Scafuro (University of California Los Angeles, USA), gave a talk illustrating the content of their work.

We also wish to thank all the members of the Program Committee for their collaboration in the reviewing process of the papers, and the colleagues of the Organizing Committee for their invaluable cooperation. Finally, we thank the people of the secretariat of the Dipartimento di Matematica e Informatica and of the Science Faculty of the University of Palermo, who assisted and supported us in the organization of the meeting.

September 2013

Sabrina Mantaci and Marinella Sciortino
Organization

Program Committee

Tiziana Calamoneri (Sapienza University of Rome)
Giusi Castiglione (University of Palermo)
Pierluigi Crescenzi (University of Florence)
Mariangiola Dezani (University of Turin)
Roberto Grossi (University of Pisa)
Sabrina Mantaci (University of Palermo, co-chair)
Simone Martini (University of Bologna)
Paolo Massazza (University of Insubria)
Eugenio Moggi (University of Genoa)
Antonio Restivo (University of Palermo)
Marinella Sciortino (University of Palermo, co-chair)
Ugo Vaccaro (University of Salerno)

Organizing Committee

Giusi Castiglione (University of Palermo)
Chiara Epifanio (University of Palermo)
Gabriele Fici (University of Palermo)
Sabrina Mantaci (University of Palermo)
Giovanna Rosone (University of Palermo)
Marinella Sciortino (University of Palermo)
Marco Elio Tabacchi (University of Palermo)
Contents

Invited contributions

L. Galletta, P. Degano and G.-L. Ferrari  
\textit{A formal model of context-oriented programming} .......................... 1

G. F. Italiano  
\textit{Strong bridges and strong articulation points of directed graphs} ............. 2

L. Moscardelli  
\textit{Convergence issues in congestion games} ........................................ 3

Regular papers

U. de’Liguoro  
\textit{The approximation theorem for the }\Lambda\mu\textit{-calculus} .................................. 4

V. Lonati, D. Mandrioli, F. Panella and M. Pradella  
\textit{Free grammars and languages} .................................................. 10

M. B. Abu Ayyash and E. Rodaro  
\textit{A language theoretical approach to some problems on inverse semigroups} 17

D. Battaglino, M. Bouvel, A. Frosini, S. Rinaldi, S. Socci and L. Vuillon  
\textit{Pattern-avoiding polyominoes} .................................................. 23

R. Reis and E. Rodaro  
\textit{Reset regular decomposition complexity of regular ideal languages} ........... 30

M. H. ter Beek, F. Gadducci and F. Santini  
\textit{Validating reconfigurations of Reo circuits} ..................................... 36

L. Aceto, D. Della Monica, A. Ingólfsdóttir, A. Montanari and G. Sciavicco  
\textit{A complete classification of the expressiveness of interval logics of Allens relations over dense linear orders} ........................................ 45

A. Bernini, S. Bilotta, R. Pinzani and V. Vajnovszki  
\textit{Two Gray codes for }q\text{-ary }k\text{-generalized Fibonacci strings} ..................... 54

S. Crespi Reghizzi and P. San Pietro  
\textit{Commutative consensual counter languages} ..................................... 60

M. H. ter Beek, A. Lluch Lafuente and M. Petrocchi  
\textit{Combining declarative and procedural views in the feature-oriented specifi- cation and analysis of product families} ........................................... 69

\textit{Proceedings of ICTCS 2013}
F. Honsell, M. Lenisa and R. Redamalla

Polarized multigames ........................................ 78

I. Bonacina and N. Galesi

Space complexity in algebraic proof systems ..................... 86

N. Beerenwinkel, S. Beretta, P. Bonizzoni, R. Dondi and Y. Pirola

Covering pairs in directed acyclic graphs .......................... 93

P. Cenciarrelli, D. and I. Salvo

On two different forms of inefficiency in network design ............. 101

J. Clément and L. Giambruno

On the number of prefix and border tables ........................ 109

E. G. Omodeo, A. Policriti and A. Tomescu

Bridging syllogistics with combinatorics .......................... 117

L. Bernardinello, C. Ferigato, S. Haar and L. Pomello

Dynamically closed sets in occurrence nets ........................ 123

S. Bistarelli, S. N. Foley, F. Santini and F. Vicino

An introduction to consistently merging trust-networks with bipolar preferences ........................................ 129

V. V. Gusev, M. I. Maslennikova, E. V. Pribavkina and E. Rodaro

Language theoretic approach to synchronizing automata .............. 137

M. Anselmo, D. Giammarresi and M. Madonia

Strong prefix codes of pictures .................................. 142

M. Bartoletti, P. Di Giamberardino and R. Zunino

Towards a linear contract logic .................................. 148

S. Brunetti, G. Cordasco, L. Gargano, E. Lodi and W. Quattrociocchi

Minimum weight multicolor dynamos ................................ 157

L. Bernardinello and C. Ferigato

On the composition of regional structures and their logics ........... 165

R. Gentilini

A SAT encoding for solving games with energy objectives .......... 172

R. Reis and E. Rodaro

The language of initially connected deterministic finite automata .... 180

M. Crochemore, C. S. Iliopoulos, A. Langiu and F. Mignosi

On the multiple common substring problem ........................ 186

Proceedings of ICTCS 2013
D. Ancona, P. Giannini and E. Zueca
Incremental rebinding ................................................................. 193

C. Barton, C. S. Iliopoulos and S. P. Pissis
Circular string matching revisited ................................................. 201

M. M. Bersani, M. Rossi and P. San Pietro
Continuous-time metric temporal logic ......................................... 207

A. Alatabbi, C. S. Iliopoulos, A. Langiu and M. Sohel Rahman
Computing the longest common abelian factor ................................ 215
A formal model of context-oriented programming

Letterio Galletta, Pierpaolo Degano, and Gian-Luigi Ferrari

Dipartimento di Informatica – Università di Pisa, Italy
{galletta,degano,giangi}@di.unipi.it

Abstract. Context-Oriented programming languages provide us with primitive constructs to adapt program behaviour depending on the evolution of their operational context. We introduce a core Context-Oriented functional language, that includes facilities to declaratively program contexts, and supports the definition of modular adaptation patterns via high-order mechanisms. We equip the language with a type and effect system that type-checks programs and computes, as an effect, a sound approximation of their behaviour. The effect is exploited to statically verify that programs correctly operate in all possible run-time contexts.
Strong Bridges and Strong Articulation Points of Directed Graphs

Giuseppe F. Italiano

Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università di Roma “Tor Vergata”, Italy
italiano@disp.uniroma2.it

Abstract. Given a directed graph \( G \), an edge is a strong bridge if its removal increases the number of strongly connected components of \( G \). Similarly, a vertex is a strong articulation point if its removal increases the number of strongly connected components of \( G \). Strong articulation points and strong bridges are related to the notion of 2-vertex and 2-edge connectivity of directed graphs, which surprisingly seems to have been overlooked in the past. In this talk, we survey some very recent work in this area, both from the theoretical and the practical viewpoint.
Convergence issues in congestion games

Luca Moscardelli
University of Chieti-Pescara, Italy
moscardelli@di.univaq.it

Abstract. Congestion games are a widely studied class of non-cooperative games. In fact, besides being able to model many practical settings, they constitute a framework with nice theoretical properties: Congestion games always converge to pure Nash Equilibria by means of improvement moves performed by the players, and many classes of congestion games guarantee a low price of anarchy, that is the ratio between the worst Nash Equilibrium and the social optimum. Unfortunately, the time of convergence to Nash Equilibria, even under best response moves of the players, can be very high, i.e., exponential in the number of players, and in many settings also computing a Nash equilibrium can require a high computational complexity. Motivated by the above facts, in order to guarantee a fast convergence to Nash Equilibria, in the last decade many computer science researchers focused on special classes of congestion games (e.g., with linear or polynomial delay functions), on simplified structures of the strategy space (e.g., on symmetric games in which all players share the same set of strategies or on matroid congestion games in which the set of strategies constitutes a matroid) and on the relaxation of the notion of Nash Equilibria (e.g., exploiting the notion of $\epsilon$-Nash Equilibria). We survey such attempts that, however, only in some very specific cases have led to satisfactory results on the speed of convergence to Nash Equilibria. If we relax the constraint of reaching a Nash Equilibrium, and our goal becomes that of reaching states approximating the social optimum by a “low” factor, i.e., a factor being order of the price of anarchy, significantly better results on the speed of convergence under best response dynamics can be achieved. Interestingly, in the more general asymmetric setting, fairness among players influences the speed of convergence. For instance, considering the fundamental class of linear congestion games, if each player is allowed to play at least once and at most $\beta$ times every $T$ best responses, states with approximation ratio $O(\beta)$ times the price of anarchy are reached after $T[\log\log n]$ best responses, and such a bound is essentially tight also after exponentially many ones. It is worth noticing that the structure of the game implicitly affects its performances in terms of convergence speed: In particular, in the symmetric setting the game always converges to an efficient state after a polynomial number of best responses, regardless of the frequency each player moves with. Most of these results extend to polynomial and weighted congestion games.

Proceedings of ICTCS 2013
The Approximation Theorem for the $\Lambda\mu$-Calculus
(extended abstract*)

Ugo de'Liguoro
deliguoro@di.unito.it
Università di Torino

Introduction

The $\Lambda\mu$-calculus is an extension of Parigot’s $\lambda\mu$-calculus [5], proposed by de Groote e.g. in [2] and developed by Saurin [7–9]. The interest of $\Lambda\mu$ lies in the fact that it preserves the separability property, namely the Böhm Theorem of the $\lambda$-calculus with $\beta\eta$-conversion [7], which is not the case of $\lambda\mu$ [1].

The Approximation Theorem is a central result in the study of sensible $\lambda$-theories, relating them to the structure of Scott’s $D_\infty$ $\lambda$-models. The theorem states that the denotation $[M]$ of any term $M$ in a domain theoretic model is the directed sup of the denotations $[A]$ of its approximations $A$, where $A$ is a partially defined term (including a constant $\Omega$ for the undefined parts) recording the stable part of $M$ reducts.

With intersection types the approximation theorem rephrases into the claim that $M$ has a type $\sigma$ if and only if there is an approximation $A$ of $M$ that can be typed by $\sigma$ [3]. In [12] we obtained an intersection type assignment system by presenting Streicher and Reus’s model of $\lambda\mu$ [11] as a filter model. But we also remarked that Streicher and Reus’s is not a model of $\Lambda\mu$.

Recently Nakazawa and Katsumata [4] have proposed a variant of Streicher and Reus’s construction, which is a model of $\Lambda\mu$. Hence it make sense to study the structure of such models via intersection type systems.

Here we study Saurin’s notion of approximate normal form for $\Lambda\mu$, together with a pre-congruence relation such that the set $\mathcal{A}(M)$ of approximants of a (closed) $\Lambda\mu$-term $M$ is an ideal. We then introduce our intersection type preorder and type assignment system and establish that it generates a filter model, so that the typings are invariant under subject conversion. Finally we establish the main result of the paper, namely that a term-type $\delta$ can be assigned to a $\Lambda\mu$-term $M$ in some bases $\Gamma, \Delta$ (for term and stream variables respectively) if and only if $\delta$ can be assigned to some approximation $A$ of $M$ in the same $\Gamma, \Delta$.

1 Approximants for the $\Lambda\mu$-calculus

The distinctive feature of the $\Lambda\mu$-calculus w.r.t. Parigot’s $\lambda\mu$ consists into abolishing the distinction between named terms $[\alpha]M$ and unnamed or ordinary terms. In particular the restrictions that in $\mu\alpha.M$ the subexpression $M$ is named and that in $\lambda x.N$ term $N$ is unnamed are dropped. In the grammar

* The full paper has been accepted for publication in MSCS.

Proceedings of ICTCS 2013
below we have adopted Saurin’s notation \((M)\alpha\) for Parigot’s and de Groote’s \([\alpha]M\). This makes explicit the intuition that \(\alpha\) represents a potentially infinite stream of terms to which \(M\) is applied.

**Definition 1 (Term Syntax).**

\[
M, N ::= x \mid \lambda x.M \mid (M)N \mid \mu\alpha.M \mid (M)\alpha
\]

\(\text{Var}_T\) and \(\text{Var}_S\) are denumerable sets of *term variables* ranged over by \(x, y, \ldots\) and *stream variables* ranged over by \(\alpha, \beta, \ldots\) respectively. \(\Sigma_{\Lambda\mu}\) is the set of terms generated by the grammar in Definition 1 and \(\Sigma_{\Lambda\mu}\) is the subset of closed terms. Bound and free variables, written \(\text{fv}(M)\), are defined as usual, with both \(\lambda\) and \(\mu\) as binders.

For \(M, N \in \Sigma_{\Lambda\mu}\) and \(\alpha \in \text{Var}_S\) we define the *structural substitution* \(M[\alpha \leftarrow N]\) (in de Groote’s notation) as the replacement of any subterm \((P)\alpha\) of \(M\) with \(\alpha \in \text{fv}(M)\), by the subterm \((P)N\alpha\). In particular:

\[
((M)\alpha)[\alpha \leftarrow N] = (M[\alpha \leftarrow N])N\alpha.
\]

By \(M[x := N]\) and \(M[\alpha := \beta]\) we mean ordinary capture avoiding substitutions.

**Definition 2 (Axioms).**

\[
\begin{align*}
(\beta_T) \quad (\lambda x.M)N &= M[x := N] \\
(\beta_S) \quad (\mu\alpha.M)\beta &= M[\alpha := \beta] \\
(\eta_T) \quad \lambda x.(M)x &= M & \text{if } x \not\in \text{fv}(M) \\
(\eta_S) \quad \mu\alpha.(M)\alpha &= M & \text{if } \alpha \not\in \text{fv}(M) \\
(\mu) \quad (\mu\alpha.M)N &= \mu\alpha.M[\alpha \leftarrow N]
\end{align*}
\]

We write \(\vdash M = N\) if this equality is derivable from the axioms above by equational reasoning. By orienting the axioms from left to right one obtains a non-confluent notion of reduction [8]. This is fixed by replacing the left-to-right version of the \((\mu)\)-axiom by rule \((\text{fst})\) below:

\[
(\text{fst}) \quad \mu\alpha.M \rightarrow \lambda x.\mu\alpha.M[\alpha \leftarrow x] & \quad \text{if } x \not\in \text{fv}(M).
\]

The following is Theorem 2.16 in [6] and Theorem 3.1 in [10].

**Theorem 1 (Confluence of \(\rightarrow\) w.r.t. \(\Sigma_{\Lambda\mu}^n\)).** For \(M, M_1, M_2 \in \Sigma_{\Lambda\mu}^n\):

\[
M \rightarrow^* M_1, M_2 \Rightarrow \exists M_3 \in \Sigma_{\Lambda\mu}^n. M_1, M_2 \rightarrow^* M_3.
\]

In “sensible” theories of the ordinary \(\lambda\)-calculus a term \(M\) has computational meaning if and only if it reduces to a head normal form. By replacing \(\beta\)-redexes in \(M\) by a constant \(\Omega\) and equating \(\Omega N = \Omega = \lambda x.\Omega\) one obtains a context which remains unchanged in any reduction out of \(M\), because the only parts that can be affected by reduction are those subterms replaced by \(\Omega\). If one orders \(\lambda\Omega\)-terms (which are obtained by adding the constant \(\Omega\) to the grammar of \(\lambda\)-terms) by the compatible closure of the inequality \(\Omega \preceq M\), one gets the notion of approximate normal form in the \(\lambda\)-calculus.

Building over Saurin’s definition of head normal forms in \(\Lambda\mu\) [9] we define:
Definition 3 (Approximate Normal Forms).

\[ A := \Omega \mid \lambda x_0 \mu a_1 \lambda x_1 \ldots \mu a_n \lambda x_n. (y) A_0 \beta_1 A_1 \ldots \beta_m A_m \]

Let \( A \) be the set of approximate normal forms.

Definition 4 (Pre-Congruence over \( \mathcal{A} \)). Let \( \preceq \) be the least pre-order over \( \mathcal{A} \) and \( \succeq = \preceq \cap \succeq \), which are (pre-)congruences and such that:

1. \( \Omega \succeq A \),
2. \( \lambda x.(A)x \simeq A \) if \( x \not\in \text{fv}(A) \) and \( A \) is \( \lambda \mu \)-free,
3. \( \mu a.(A)\alpha \simeq A \) if \( \alpha \not\in \text{fv}(A) \) and \( A \) is \( \lambda \mu \)-free,
4. \( \mu a.A \succeq \lambda x.\mu a.A[\alpha \leftarrow x] \), if \( A \not\in \Omega \) and \( x \not\in \text{fv}(A) \).

A term \( R \) is a pre-redex if it has one of the shapes \((\lambda x.M)N\), \((\lambda x.M)\alpha\), \((\mu a.M)N\) or \((\mu a.M)\beta\). The pre-redex \((\mu a.M)N\) includes the \((\text{fst})\)-redex \(\mu a.M\), which after contraction gives rise to the \((\beta_\text{T})\)-redex \((\lambda x.\mu a.M[\alpha \leftarrow x])N\). The pre-redex \((\lambda x.M)\alpha\) is only a potential redex, which becomes an actual redex after a \((\text{fst})\)-reduction only if it occurs as a subterm of some term in which \( \alpha \) is bound.

Definition 5 (Approximants). Let \( \phi : \Sigma_{\Lambda \mu} \rightarrow \mathcal{A} \) be the map:

1. \( \phi(\lambda x_0 \mu a_1 \lambda x_1 \ldots \mu a_n \lambda x_n.(R) M_0 \beta_1 M_1 \ldots \beta_m M_m) = \Omega \), if \( R \) is a pre-redex,
2. \( \phi(\lambda x_0 \mu a_1 \lambda x_1 \ldots \mu a_n \lambda x_n.(y) M_0 \beta_1 M_1 \ldots \beta_m M_m) = \lambda x_0 \mu a_1 \lambda x_1 \ldots \mu a_n \lambda x_n.(y) \phi(M_0) \beta_1 \phi(M_1) \ldots \beta_m \phi(M_m) \),

where \( \phi(M_i) \) is the componentwise application of \( \phi \) to the terms of the vector \( M_i \). For \( M \in \Sigma_{\Lambda \mu} \) we define the set

\[ \mathcal{A}(M) = \{ A \in \mathcal{A} \mid \exists N \rightarrow^* N \& A \preceq \phi(N) \} \]

Theorem 2. For all \( M \in \Sigma_{\Lambda \mu}^\circ \) the set \( \mathcal{A}(M) \) is an ideal over \( (\mathcal{A}, \preceq) \).

2 Intersection Types for \( \Lambda \mu \)

Even if there is just one kind of terms in the \( \Lambda \mu \) syntax, to type them we have to make assumptions also about stream variables. Consequently we follow \[12\] by having two kinds of types for terms and term variables, and for stream variables.

Type syntax is motivated by the semantics (see section 3), where terms denote functions from streams to term denotations, and streams are infinite tuples of term denotations. Therefore term types are (intersections of) arrows of the shape \( \sigma \rightarrow \delta \) where \( \sigma \) is a stream type and \( \delta \) a term type. Stream types are (intersections of) product types of the shape \( \sigma = \delta_1 \times \cdots \times \delta_k \times \omega \), where the \( \delta_i \) are term types. The ending \( \omega \) expresses the fact that \( \sigma \) only encodes a finite information about any infinite stream \( s = \langle d_1, \ldots, d_k, \ldots \rangle \) such that each \( d_i \) satisfies the respective \( \delta_i \), while \( \omega \) is the type of the infinite tail.
Definition 6 (Intersection Types).

\[ T_T : \quad \delta ::= \varphi \mid \sigma \rightarrow \delta \mid \delta \land \delta \mid \omega_T \]

\[ T_S : \quad \sigma ::= \delta \times \sigma \mid \sigma \land \sigma \mid \omega_S \]

where \( \varphi \) varies over a denumerable set of atomic types.

\[ T_T \] is the set of term types and \( T_S \) the set of stream types. When clear from the context we shall write just \( \omega \) in place of \( \omega_T \) or \( \omega_S \).

Definition 7 (Subtyping). The relations \( \leq_T \) and \( \leq_S \) over \( T_T \) and \( T_S \) respectively are the least preorders such that \( \land \) is the meet, \( \omega_T \) and \( \omega_S \) are the top w.r.t. \( \leq_T \) and \( \leq_S \) respectively, and:

1. \( \omega \leq_T \omega \rightarrow \omega \)
2. \( \varphi \leq_T \omega \rightarrow \varphi \leq_T \varphi \)
3. \( (\sigma \rightarrow \delta_1) \land (\sigma \rightarrow \delta_2) \leq_T \sigma \rightarrow (\delta_1 \land \delta_2) \)
4. \( \sigma_2 \leq_S \sigma_1, \delta_1 \leq_T \delta_2 \Rightarrow \sigma_1 \rightarrow \delta_1 \leq_T \sigma_2 \rightarrow \delta_2 \)
5. \( \omega \leq_S \omega \times \omega \)
6. \( (\delta_1 \times \sigma_1) \land (\delta_2 \times \sigma_2) \leq_S (\delta_1 \land \delta_2) \times (\sigma_1 \land \sigma_2) \)
7. \( \delta_1 \leq_T \delta_2, \sigma_1 \leq_S \sigma_2 \Rightarrow \delta_1 \times \sigma_1 \leq_S \delta_2 \times \sigma_2 \)

We abbreviate by \( \delta_1 \sim_T \delta_2 \) the inequalities \( \delta_1 \leq_T \delta_2 \) and \( \delta_2 \leq_T \delta_1 \). Similarly for \( \sim_S \). Usual (in) equalities holding in EATS systems are derivable for the \( \leq_T \) pre-order. Concerning the \( \leq_S \) preorder we have:

\[ \forall \sigma \in T_S \exists k \in \mathbb{N}, \delta_1, \ldots, \delta_k \in T_T. \sigma \sim_S \delta_1 \times \cdots \times \delta_k \times \omega, \]

\[ \delta_1 \times \cdots \times \delta_k \times \omega \leq_S \delta_1' \times \cdots \times \delta_k' \times \omega \iff h \leq k \land \forall i \leq h, \delta_i \leq_T \delta_i'. \]

A basis for term variables is a set \( \Gamma = \{ x_1 : \delta_1, \ldots, x_n : \delta_n \} \) with the \( x_i \) pairwise distinct and \( \delta_i \in T_T \); a basis for stream variables \( \Delta = \{ \alpha_1 : \sigma_1, \ldots, \alpha_m : \sigma_m \} \) is defined similarly with \( \sigma_j \in T_S \). A judgement is an expression \( \Gamma \vdash M : \delta \mid \Delta \), where \( \Gamma, \Delta \) are bases, \( M \in \Sigma_{A\mu} \) and \( \delta \in T_T \). The writing \( \Gamma, x : \delta \vdash \delta \) abbreviates \( \Gamma \cup \{ x : \delta \} \) with \( x \notin \Gamma \). Similarly for \( \alpha : \sigma, \Delta \).

Definition 8 (Type Assignment System for \( A\mu \)).

\[ \Gamma, x : \delta \vdash x : \overline{\delta} \mid \Delta \quad (Ax) \]

\[ \Gamma \vdash \lambda x. M : \delta_1 \times \sigma \rightarrow \delta_2 \mid \Delta \quad (\lambda) \]

\[ \Gamma \vdash M : \delta_1 \times \sigma \rightarrow \delta_2 \mid \Delta \quad \Gamma \vdash N : \delta_1 \mid \Delta \quad (App) \]

\[ \Gamma \vdash M : \delta \mid \Delta \quad (\mu) \]

\[ \Gamma \vdash \mu \alpha. M : \sigma \rightarrow \delta \mid \Delta \quad (S) \]

\[ \Gamma \vdash M : \omega \mid \Delta \quad (\omega) \]

\[ \Gamma \vdash M : \delta_1 \mid \Delta \quad \Gamma \vdash M : \delta_2 \mid \Delta \quad (\wedge) \]

\[ \Gamma \vdash M : \delta_1 \rightarrow \delta_2 \mid \Delta \quad \Gamma \vdash M : \delta_1 \mid \Delta \quad \delta_1 \leq_T \delta_2 \quad (\leq) \]
3 Extensional $\Lambda\mu$ Filter Model

The type system introduced in the previous section is motivated by the semantics of $\Lambda\mu$-terms as defined in [4].

**Definition 9 (Extensional $\Lambda\mu$-model).** An extensional $\Lambda\mu$-model is a quadruple $(D, S, ::, [\cdot])$ where:

1. $D$ and $S$ are non-empty sets satisfying, up to isomorphism, the equations:

   $$D = [S \to D] \subseteq D^S, \quad S = D \times S;$$

2. $\cdot : D \times S \to S$ is the inverse of the isomorphism $S = D \times S$;

3. $\cdot : \Sigma_{\Lambda\mu} \times \text{Env} \times S \to D$, where $\text{Env} = (\text{Var}_T \to D) + (\text{Var}_S \to S)$, is such that:

   $$\begin{align*}
   [x] e s &= e(x) \\
   [\lambda x. M] e (d :: s) &= [M] e[x \mapsto d] s \\
   [(M) N] e s &= [M] e ([N] e s) :: s \\
   [\mu \alpha. M] e s &= [M] e[\alpha \mapsto s] \\
   [(M) \alpha] e s &= ([M] e s) s(\alpha)
   \end{align*}$$

   where $e[x \mapsto d](y) = d$ if $x = y$, $e(y)$ else. $e[\alpha \mapsto s]$ has a similar meaning.

An extensional $\Lambda\mu$-model can be constructed as a $D_\infty$ model, endowed with more structure to provide stream denotations. Extensional $\Lambda\mu$-models are a variant of Streicher and Reus’s models of Parigot’s $\lambda\mu$ [11]. Nakazawa and Katsumata’s models validate all $\Lambda\mu$-axioms, while Streicher and Reus’s do not validate axiom $(\beta_S)$. The following is proved in [4].

**Theorem 3 (Soundness).** If $(D, S, ::, [\cdot])$ is an extensional $\Lambda\mu$-model then for all $M, N \in \Sigma_{\Lambda\mu}$:

$$\vdash M = N \implies [M] = [N].$$

Let $\mathcal{F}_T$ and $\mathcal{F}_S$ be the sets of filters over $(T_T, \leq_T)$ and $(T_S, \leq_S)$ respectively. For $d \in \mathcal{F}_T$ and $s \in \mathcal{F}_S$ define:

$$\begin{align*}
   d :: s &= \{ \delta \times \sigma \in \mathcal{T}_S \mid \delta \in d \& \sigma \in s \} \\
   d \cdot s &= \{ \delta \in \mathcal{T}_T \mid \exists \sigma \in s. \sigma \rightarrow \delta \in d \}
\end{align*}$$

**Definition 10.** For $e \in \text{Env}_F = (\text{Var}_T \to \mathcal{F}_T) + (\text{Var}_S \to \mathcal{F}_S)$ we set:

$$\Gamma, \Delta \models e \iff \forall x \in \text{Var}_T. \Gamma(x) \in e(x) \& \forall \alpha \in \text{Var}_S. \Delta(\alpha) \in e(\alpha).$$

Then for all $e \in \text{Env}_F$ and $s \in \mathcal{F}_S$ define the map $\cdot : \Sigma_{\Lambda\mu} \times \text{Env}_F \times \mathcal{F}_S \to \mathcal{F}_T$ by

$$\begin{align*}
   [M] e s &= \{ \delta \in \mathcal{T}_T \mid \exists \Gamma, \Delta. \Gamma, \Delta \models e \& \Gamma \vdash M : \delta \mid \Delta \} \cdot s.
\end{align*}$$

We take $(\mathcal{F}_T, \subseteq)$ and $(\mathcal{F}_S, \subseteq)$ with the Scott topology. The product $\mathcal{F}_T \times \mathcal{F}_S$ and the function space $[\mathcal{F}_S \to \mathcal{F}_T]$ are in the category of algebraic lattices.

**Theorem 4 (Filter Model).** The structure $(\mathcal{F}_T, \mathcal{F}_S, ::, [\cdot])$ is an extensional $\Lambda\mu$-model.
4 The Approximation Theorem

The approximation property of the interpretation mapping $\llbracket \cdot \rrbracket$ w.r.t. a model $D$ states that: $\llbracket M \rrbracket e = \bigcup \{ \llbracket A \rrbracket e \mid A \in \mathcal{A}(M) \}$ for all environment $e$, where $\llbracket \cdot \rrbracket$ extends to approximate normal forms by setting $\llbracket \Omega \rrbracket e = \bot$, namely the bottom of $D$.

In case of the filter model generated by the intersection type assignment system in section 3, it is equivalent to the statement that $\Gamma \vdash M : \delta \mid \Delta$ if and only if $\Gamma \vdash A : \delta \mid \Delta$ for some $A \in \mathcal{A}(M)$:

**Theorem 5 (Approximation Theorem).** For all $M \in \Sigma_{\lambda \mu}$ and $\delta \in \mathcal{T}_T$:

$$\Gamma \vdash M : \delta \mid \Delta \iff \exists A \in \mathcal{A}(M). \Gamma \vdash A : \delta \mid \Delta.$$  

The if part is a rather easy consequence of the fact that any type that can be assigned to $\Omega$ is equivalent to $\omega$. The only if part can be established by a clever adaptation of Tait’s computability argument.

References

Free Grammars and Languages

Violetta Lonati\textsuperscript{2}, Dino Mandrioli\textsuperscript{1}, Federica Panella\textsuperscript{1}, Matteo Pradella\textsuperscript{1}

\textsuperscript{1} DEIB - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
\{dino.mandrioli, federica.panella, matteo.pradella\}@polimi.it

\textsuperscript{2} DI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@di.unimi.it

Floyd’s Operator Precedence languages (OPLs) were originally introduced to support deterministic parsing of programming languages [11]; then, interest in them decayed for several decades, probably due to the advent of more expressive grammars, such as LR ones which also allow for efficient deterministic parsing.

Recently, however, we renewed our interest in this family of languages on the basis of two major properties thereof: 1) their “locality property”, i.e., the fact that partial strings can be parsed independently of the context in which they occur within a whole string; this enables more effective parallel and incremental parsing techniques than for other deterministic languages [3, 4]; and 2) the fact that, to the best of our knowledge, OPLs are the largest family closed w.r.t. Boolean operations, concatenation, Kleene * and other classical operations [7]; furthermore they are recognized by a peculiar automata family [12] and are characterized in terms of classical monadic second order (MSO) logic [13]. This latter property entitles OPLs as a best candidate for extending the application of powerful verification techniques such as model-checking far beyond the original class of regular languages and even other recent families such as Visibly Pushdown Languages (VPLs) [2] which are strictly contained within OPLs too.

In this paper we introduce (more precisely, resume) a subclass of OPLs, namely free languages (FrLs) which were defined in [9, 8] with the main motivation of grammar inference. FrLs constitute a kind of algebra within the structure defined by any given operator precedence matrix (OPM) [8]. Besides briefly outlining a “specialized” version of automata explicitly tailored for FrLs, we offer a new logic characterization in terms of first-order logic, as opposed to the traditional but more complex one in terms of MSO logic. FrLs however, lose some closure properties and have some “distinguishing” limits in terms of generative power, which nevertheless covers various relevant and heterogeneous cases: thus, they better lend themselves to act as “basic skeleton” for describing the structural part of a language to be complemented either with a suitable intersection with “regular control” or with additional restrictions formalized in terms of first-order properties conjuncted with the original ones associated with a grammar (FrG), somewhat in the spirit of the classical Chomsky-Schützenberger characterization of CF languages.

Basic definitions

The reader may find more details on OPGs in [7, 8, 11]. Let \( \Sigma \) be an alphabet and \( \varepsilon \) the empty string. Let \( G = (N, \Sigma, P, S) \) be a context-free (CF) grammar,
where $N$ is the nonterminal alphabet, $P$ the rule (or production) set, and $S$ the axiom. A rule is in \textit{operator form} if its right hand side (r.h.s.) has no adjacent nonterminals; an \textit{operator grammar} (OG) contains only such rules. The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $r, s, t, u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \beta, \gamma$ denote strings over $\Sigma \cup N$. The strings may be empty, unless stated otherwise.

For an OG $G$ and a nonterminal $A$, the \textit{left and right terminal sets} are

$$
L_G(A) = \{a \in \Sigma \mid A \Rightarrow Baa\} \quad \text{and} \quad R_G(A) = \{a \in \Sigma \mid A \Rightarrow a\alpha B\}
$$

where $B \in N \cup \{\varepsilon\}$ and $\Rightarrow$ denotes the derivation relation.

The following binary operator precedence (OP) relations are defined:

- equal in precedence: $a = b \iff \exists A \rightarrow aab\beta, B \in N \cup \{\varepsilon\}$
- takes precedence: $a < b \iff \exists A \rightarrow \alpha\beta, D \in N$ and $a \in R_G(D)$
- yields precedence: $a > b \iff \exists A \rightarrow \alpha\beta, D \in N$ and $b \in L_G(D)$

The \textit{operator precedence matrix} (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that associates with any ordered pair $(a, b)$ the set $M_{ab}$ of OP relations holding between $a$ and $b$. An OG $G$ is an \textit{operator precedence} or \textit{Floyd grammar} (OPG) if, and only if, $M = OPM(G)$ is a \textit{conflict-free} matrix, i.e., $\forall a, b, |M_{ab}| \leq 1$.

A conflict-free OPM $M$ on an alphabet $\Sigma$ assigns a structure to strings in $\Sigma^*$, i.e., a string can be uniquely associated with a tree.

If $M_{ab} = \varnothing$, where $\varnothing \in \{<, =, >\}$, we write $a \circ b$. For $u, v \in \Sigma^*$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$. $M$ is \textit{complete} if $M_{ab}$ is defined for every $a$ and $b$ in $\Sigma$. Moreover in the following we assume that $M$ is $\leq$-\textit{acyclic}, which means that $c_1 \preceq c_2 \preceq \ldots \preceq c_k \preceq c_1$ does not hold for any $c_1, c_2, \ldots, c_k \in \Sigma, k \geq 1$. See [8, 7, 15] for a discussion on this hypothesis.

A \textit{simple chain} is a string $c_0c_1c_2\ldots c_{\ell}c_{\ell+1}$, written as $c_0[c_1c_2\ldots c_{\ell}]^{c_{\ell+1}}$, such that: $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}, c_i \in \Sigma$ for every $i = 1, 2, \ldots, \ell, M_{c_0c_{\ell+1}} \neq \varnothing$, and $c_0 < c_1 \preceq c_2 \preceq \ldots \preceq c_{\ell-1} \preceq c_{\ell} > c_{\ell+1}$ (by convention the special symbol $\#$ yields precedence to all elements of $\Sigma$ and all elements of $\Sigma$ take precedence over it). A \textit{composed chain} is a string $c_0s_0c_1s_1c_2\ldots c_{\ell}s_{\ell}c_{\ell+1}$, where $c_0[c_1c_2\ldots c_{\ell}]^{c_{\ell+1}}$ is a simple chain, and $s_i \in \Sigma^*$ is the empty string or is such that $c_0[s_i]^{c_{\ell+1}}$ is a chain (simple or composed), for every $i = 0, 1, \ldots, \ell$. Such a composed chain will be written as $c_0[s_0c_1s_1c_2\ldots c_{\ell}s_{\ell}]^{c_{\ell+1}}$. A string $s \in \Sigma^*$ is \textit{compatible} with the OPM $M$ if $\#[s]#$ is a chain.

Let $s$ be any word in $\Sigma^*$. For $0 \leq x < y \leq |s| + 1$, we say that $(x, y)$ is a \textit{chain boundary} iff there exists a sub-string of $#s#$ which is a chain $a^{|r|^b}$, such that $a$ is in position $x$ and $b$ is in position $y$. In general if $(x, y)$ is a chain boundary, then $y > x + 1$, and a position $x$ may be in such a relation with more than one position and vice versa. Moreover, if $s$ is compatible with $M$, then $(0, |s| + 1)$ is a chain boundary.

**Definition 1.** Free Grammar and Language

First, the definition of left and right terminal sets of a grammar $G$ is extended from $N$ to $(\Sigma \cup N)^*$ in the following natural way:

---

**Proceedings of ICTCS 2013**
Let $G$ be an OPG in the usual form such that: the axiom $S$ does not occur in the r.h.s. of any rule, no empty rule exists except possibly $S \rightarrow \varepsilon$, the other rules having $S$ as l.h.s are renaming, and no other renaming rules exist. $G$ is a free grammar (FrG) iff a) for every production $A \rightarrow \alpha$, with $A \neq S$, $L_G(A) = L_G(\alpha)$ and $R_G(A) = R_G(\alpha)$ and b) for every nonterminals $A, B$, $L_G(A) = L_G(B)$ and $R_G(A) = R_G(B)$ implies $A = B$. A language generated by a FrG is a free language (FrL).

Notice that, by definition, a free grammar is invertible (i.e., no two rules have identical r.h.s). Also, being $N \setminus \{S\}$ isomorphic to $P(\Sigma) \times P(\Sigma)$, it is customary to use $P(\Sigma) \times P(\Sigma) \cup \{S\}$ as the nonterminal alphabet of a free grammar. Given an OPM $M$, the maxgrammar associated with $M$ is the free grammar that contains all productions that are compatible with $M$. Thus, the set of free grammars with a given OPM is a lattice whose top element is the maxgrammar associated with the matrix $[8]$.  

**First properties and examples**

In this section we investigate the generative power of free grammars by means of some examples and comparisons with other classes of languages.

**Example 1.** The FrG depicted in Figure 1 generates unparenthesized arithmetic expressions with the usual precedences of $\ast$ w.r.t. $+$, which are not VPLs. This grammar is obtained from the maxgrammar by taking only those nonterminals that have $a$ in both left and right sets. By this way we guarantee that all strings generated by the grammar begin and end with an $a$. We did not include the copy rules $S \rightarrow A$, for each nonterminal $A \in P(\Sigma) \times P(\Sigma)$, for brevity.

$$L_G(\alpha) = \begin{cases} \text{if } \alpha = a\beta, a \in \Sigma, \beta \in (\Sigma \cup N)^+ \text{ then } \{a\}, \\ \text{else if } \alpha = Aa\beta, A \in N \text{ then } L_G(A) \cup \{a\} \end{cases} \quad \forall a \in \Sigma$$

$R_G(\alpha)$ is defined symmetrically.

![Fig. 1. A FrG for unparenthesized arithmetic expressions (right), and its OPM (left).](image-url)

---

In [8] it is also shown that each free grammar is the top element of a Boolean algebra and that the whole family of OPLs compatible with a given OPM is itself a Boolean algebra whose top element is the language generated by the maxgrammar.
Extending the above grammar to generate also parenthesized arithmetic strings is an easy exercise by observing that all that we need are rules of the type \( \{\{\}, \{\}\} \rightarrow \{X\} \), where \( \{ \) and \( \}\) denote left and right parentheses, \( X \) represents anyone of previous nonterminals, and the new nonterminal can occur in turn wherever \( \{a\}, \{a\} \) occurs in the above grammar. Free grammars are also adequate to generate various other types of languages, which model sequences of operations handled in a classical LIFO policy which can be interrupted by high(er) priority interrupts, sequences of operations that manage documents and their updating, etc. On the other hand FrLs are noncounting [6]; thus, since regular languages can be counting, FrLs are incomparable with regular languages and VPLs.

The above example also illustrates a typical feature of FrGs: they are not intended to be built by hand; being driven by the powerset of \( \Sigma \), both \( N \) and \( P \) suffer from combinatorial explosion. However, according to their original motivation to support grammar inference, they are well suited to be built by some automatic device.\(^4\) Furthermore, their typical canonical form makes also easy the application of the classical minimization procedure that extends to structure grammars the minimization of finite state machines [14].

We envisage two major approaches to build FrGs generating a desired language: a bottom up one abstracts away from a given sample of language sentences as in traditional grammar inference (in this case it exploits the distinguishing property of FrGs that they can be inferred in the limit on the basis of a positive sample only [9]); a top-down one, instead, starts from the maxgrammar and “prunes” nonterminals and productions that would generate undesired sentences: this technique has been applied to build the grammar of the above examples. We will see that FrGs can be useful even when the language to define exceeds the limits of their generative power.

The next result concerns FrLs closure properties.

**Theorem 1.** FrLs (with a fixed OPM) are closed w.r.t. intersection but, unlike general OPLs and VPLs, not w.r.t. complement, union, concatenation, and Kleene’s *.

FrLs can be associated with a natural and simple class of automata accepting them: intuitively, a free automaton (FrA) shifts all input characters onto the stack, then, as soon as a full r.h.s (of the FrG) is on top of the stack, and is recognized with the help of the OPM, it replaces the r.h.s. with the unique corresponding l.h.s., if any; otherwise the string is rejected. Without going into formal details we consider FrAs as “stateless” since they simply must push symbols onto the stack and verify whether and when a r.h.s. on its top is ready to be reduced.

**First-order characterization of free languages**

Unlike general OPLs, VPLs and many other language families that require a typical MSO logic characterization [2, 13], FrLs can be characterized by means

---

\(^4\) In fact the grammar of the example and a few others have been produced by the prototype tool available at [http://home.deib.polimi.it/pradella](http://home.deib.polimi.it/pradella).
of a first-order logic (FOL). The key idea to achieve such a simplified formalization derives from the “stateless nature” of FrLs. In fact, MSO formulas characterizing “normal” languages are quantified w.r.t. sets of positions corresponding to the states entered by the accepting automaton, such states being a qualifying feature of each automaton. In the case of FrLs, instead there are no states and the stack alphabet is fixed a priori (it is essentially $\Sigma \cup \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$); thus, each position of the string can be associated with elements that belong to a finite set of possible choices: this allows to avoid quantification w.r.t. to sets of positions. Next, we illustrate the key points of the construction.

**Theorem 2.** Let $G$ be a FrG: then a FO formula $\psi_G(m)$ can be effectively built such that $w \in L(G)$ iff $w \models \psi_G(|w|)$.

As customary, first-order variables are interpreted over positions of the string: $a(x)$ is true iff the character in position $x$ is $a$; the other logical symbols have the usual meaning. Moreover, we introduce the predicate $\bowtie$: $x \bowtie y$ is true iff $(x, y)$ is a chain boundary. Figure 2a illustrates the intuitive meaning of the relation: $x \bowtie y$ means that $x + 1$ and $y - 1$, respectively, are the positions of the leftmost and rightmost leaves of the subtree with root labeled as $H$ in the figure.

![Fig. 2.](image)

W.r.t. MSO syntax on the one side we drop second-order variables and, on the other side, for every subset $S \subseteq \Sigma$ we add monadic predicates $L_S(x)$ and $R_S(x)$: intuitively, $L_S(x)$ will hold iff, in the leftmost path of a syntax tree going from the leaf at position $x$ towards the root there exists a (nonterminal) node whose $L$-set contains exactly the elements in $S$; for instance, in the case of Figure 2b $L_{\{a\}}(z), L_{\{a, d\}}(z), L_{\{a, c, d\}}(z)$ hold. Then, the construction provides axioms such that, if $x \bowtie y$, then $L_L(H)(x + 1)$ and $R_R(H)(y - 1)$ hold. For instance, for all terminal rules of the type $\rho = \{\{c_1\}, \{c_k\}\} \rightarrow c_1c_2\ldots c_k$ we build the axioms

$$\varphi_1^\rho := \forall x \left( x \bowtie x + k + 1 \land c_1(x + 1) \land c_2(x + 2) \land \ldots \land c_k(x + k) \Rightarrow L_{\{c_1\}}(x + 1) \land R_{\{c_k\}}(x + k) \right)$$
and

$$\varphi_2^n := \forall x \left( x \cap x + k + 1 \wedge L \{c_1\}(x + 1) \wedge R \{c_k\}(x + k) \Rightarrow c_1(x + 1) \wedge c_2(x + 2) \wedge \ldots \wedge c_k(x + k) \right)$$

We build similar axioms $\varphi_3^n$ and $\varphi_4^n$ for all rules of the type $\rho = (L, R) \rightarrow (L_0, R_0)c_1(L_1, R_1)c_2(L_2, R_2)c_3 \ldots (L_k, R_k)c_k(L_{k+1}, R_{k+1})$, with $(L_i, R_i) \in \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \cup \{\emptyset\}$, $0 \leq i \leq k + 1$, and $L = L_0 \cup \{c_1\}$, $R = R_{k+1} \cup \{c_k\}$ (when $(L_i, R_i) = \{\emptyset\}$, $L_i$ and $R_i$ are empty) and we require also that $\bigwedge_{(L, R)} \varphi(L, R)$ holds, where $\varphi(L, R) := \bigvee_{\rho=(L, R) \rightarrow \alpha} \varphi_1^n \wedge \varphi_2^n \wedge \varphi_3^n \wedge \varphi_4^n$.

The converse of Theorem 2 does not hold: in fact, by means of FO formulas we can define also counting languages.

Conclusions

Recently, the old-fashioned OPGs somewhat surprisingly generated renewed interest and potential application in the context of novel technologies such as parallel compilation and model-checking. In our long term path aiming at exploiting their properties we resumed free grammars, which were originally introduced to support automatic grammar inference. The main result of this paper, i.e., the characterization of FrLs in terms of FO logic, as opposed to the more general and traditional MSO one, could represent a first step towards extending to OPLs and their subfamilies other classic results on various types of logic characterization (FO, tree logic [1], LTL, . . . ) of various language families (star-free regular languages [10, 5], VPLs, . . . ).

References

A language theoretical approach to some problems on inverse semigroups

Mohammed B. Abu Ayyash and Emanuele Rodaro

1 Dipartimento di Matematica, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
mohd.abu@mail.polimi.it

2 Departamento de Matemática, Faculdade de Ciências
Universidade do Porto, 4169-007 Porto, Portugal
emanuele.rodaro@fc.up.pt

1 Introduction

An inverse semigroup is a semigroup $S$ with the property that for each element $a \in S$ there is a unique element $a^{-1} \in S$ such that $a = aa^{-1}a$ and $a^{-1} = a^{-1}a a^{-1}$. Inverse semigroups may be regarded as semigroups of partial one-to-one transformations, so they arise very naturally in several areas of mathematics. More recently also computer scientists paid attention to inverse semigroups for different reasons. First the inverse of an element in an inverse semigroup can be seen as the “undo” of the action represented by the element. Then decidability and algorithmic problems for inverse semigroups have received considerable attention in the literature during the past 30-35 years and they are essentially problems on inverse word automata. Namely in the study of these problems the concept of Schützenberger automaton introduced by Stephen [6] is crucial. Let $S = Inv(Y; T)$ be an inverse semigroup presented by finite sets $Y$ and $T$ of generators and relations (equivalently, $S$ is the quotient of the free semigroup $(Y \cup Y^{-1})^+$ by the least congruence $\tau$ that contains the Vagner relation $\nu$ and $T$ where $\nu$ is the congruence relation generated by the relations $\{(uu^{-1}u, u), (uu^{-1}v, v), (v v^{-1}u, u) : u \in (Y \cup Y^{-1})^+\}$. The Schützenberger automaton $A(Y; T; w)$ is a deterministic inverse word automaton with initial state $ww^{-1}\tau$ and a unique final state $w\tau$. These automata extend the notion of (brooted) Munn tree for free inverse semigroups [4]. Namely as the Munn trees for free inverse semigroups, Schützenberger automata have the remarkable feature that any two words $w, w' \in (Y \cup Y^{-1})^+$ represent the same element of $S$ if and only if $A(Y; T; w) = A(Y; T; w')$, or equivalently if these two automata accept the same language [6]. Then, for instance, the word problem for finitely presented inverse semigroups is reduced to the equivalence (inclusion) problem for languages. In [6] it is also proved that Schützenberger automata can be seen as the direct limit of a directed system of inverse word automata obtained, starting from the linear automaton of a word, by iteratively applying elementary expansions and determinations. Elementary determination folds two edges starting from the same vertex and labeled by the same letter of the alphabet $Y \cup Y^{-1}$. The elementary expansion applied to an inverse word graph $\Gamma$ adds to $\Gamma$ a path $(v_1, r, v_2)$ wherever there is in $\Gamma$ a path $(v_1, s, v_2)$ and $(r, s) \in T$. The limit process ensures that the final
automaton is closed with respect to the presentation \( \langle Y \mid T \rangle \), i.e. no more elementary expansion or determination can be applied. It is clear that the above procedure is not constructive and in general produces infinite automata. So it is very helpful in dealing with decidability and algorithmic problems in inverse semigroup to classify the languages recognized by Schützenberger automata of some families of inverse semigroups in the framework of language theory, so that language theoretical properties can be used to solve and analyze the complexity of some algorithmic questions. For instance languages recognized by Munn trees are Dyck languages and in [2] it was proved that the languages of Schützenberger automata of amalgams of finite inverse semigroups are deterministic context-free languages.

In this paper we focus on the languages recognized by Schützenberger automata of HNN-extensions of finite inverse semigroups (from now on just HNN-extensions). Following similar ideas of [2] we prove that the language recognized by the Schützenberger automaton of a word \( w \) with respect to the standard presentation of an HNN-extension is a real time context-free language. The paper is organized as follows: in Section 2 we recall some basic notion and facts regarding the Schützenberger automata of HNN-extensions. In Section 3 we present the real time deterministic pushdown automata associated to such Schützenberger automata and we sketch the proof that they accept the same language.

2 Schützenberger automata of HNN-extensions

Let \( S = \text{Inv}(X|R) \simeq (X \cup X^{-1})^+/\omega \) be an inverse semigroup and let \( \varphi : A \to B \) be an isomorphism of an inverse subsemigroup \( A \) of \( S \) onto an inverse subsemigroup \( B \) of \( S \) where \( e \in A \subseteq eSe \) and \( f \in B \subseteq fSf \) (or \( e \notin A \subseteq eSe \) and \( f \notin B \subseteq fSf \)) for some idempotents \( e, f \in S \). Then following [8] the inverse semigroup \( S^* = \text{Inv}(S, t \mid t^{-1}at = \varphi(a), t^{-1}t = f, \, tt^{-1} = e, \forall a \in A) \) is called the HNN-extension of \( S \) associated with \( \varphi : A \to B \). A presentation of the inverse semigroup \( S^* \) is \( \langle X \mid R_{HNN} \rangle \) where \( X = X \cup \{ t \} \) and \( R_{HNN} = R \cup \{ t^{-1}at = \varphi(a), t^{-1}t = f, tt^{-1} = e \mid a \in A \} \), we call such presentation the standard presentation of \( S^* \). The aim of this section is to describe \( A(X, R_{HNN}, w) \) for some \( w \in (X \cup X^{-1})^+ \). First we need to introduce some terminology from [5]. Let \( \Gamma \) be a deterministic inverse word graph on \( X \). The triple \( (v_1, u, v_2) \) with \( v_1, v_2 \in V(\Gamma) \) and \( u \in (X \cup X^{-1})^+ \) denotes the (unique) path from the vertex \( v_1 \) to \( v_2 \) labeled by \( u \). A subgraph \( \Delta \) of \( \Gamma \), with at least an edge is called \( S \)-lobe (or lobe for short) if it is a maximal connected inverse subgraph on \( X \). With a slight abuse of notation we say that in some \( S \)-lobe \( \Delta \) there is a path \( (v_1, s, v_2) \) for some \( s \in S \) meaning that there is some \( u \in (X \cup X^{-1})^+ \) representing \( s \) such that \( (v_1, u, v_2) \) is a path in \( \Delta \). A t-edge is an edge of \( \Gamma \) labeled by \( t \). Two vertices \( v_1, v_2 \) are called t-adjacent (for short adjacent) if they are connected by a t-edge, i.e. if either \( (v_1, t, v_2) \) or \( (v_2, t, v_1) \) are edges of \( \Gamma \). Two \( S \)-lobes \( \Delta_1, \Delta_2 \) are called adjacent if there are two t-adjacent vertices \( v_1 \in V(\Delta_1), v_2 \in V(\Delta_2) \), each of these vertices is called an intersection vertex of \( \Gamma \), if the t-edge connecting \( v_1 \) to \( v_2 \) is \((v_i, t, v_{3-i})\) for some \( i \in \{1, 2\} \) then the ordered pair \((v_i, v_{3-i})\) is called an intersection pair. The lobe graph of the
inverse word graph $\Gamma$ is the directed graph $G(\Gamma)$ whose vertices are the $S$-lobes of $\Gamma$ and there is a directed edge $e = (\Delta_1, \Delta_2)$ from a lobe $\Delta_1$ to a lobe $\Delta_2$ if there is an intersection pair $(v_1, v_2)$ with $v_1 \in V(\Delta_1)$ and $v_2 \in V(\Delta_2)$. [5] described five constructions whose iterative application, starting from the linear automaton of $w$, generates a directed system of automata whose direct limit is the Schützenberger automaton $A(\overline{X}, R_{HNw}; w)$. These operations are formed by grouping in a suitable order expansions and determinations. Iteratively applying the first four constructions one gets after finitely many steps a finite inverse word automaton on $\overline{X}$, called $t - \text{Core}(w) = (\alpha, \Omega_0, \beta)$ whose underlying inverse word graph $\Omega_0$ has a particular shape called t-opuntoid which is characterized by the following properties: each $S$-lobes is closed (with respect to $\langle X|R \rangle$) and it is a quotient of some Schützenberger automaton for some word of $X$; the lobe graph is a directed tree; it has the $t$-assimilation property, i.e. for any intersection pair $(v_1, v_2)$ belonging to the adjacent lobes $\Delta_1, \Delta_2$ there is a path $(v_1, a, v)$ for some $a \in A$ in $\Delta_1$ if and only if $(v_2, \varphi(a), v')$ is a path in $\Delta_2$ and $(v, v')$ is also an intersection pair. In general $t - \text{Core}(w)$ is not closed with respect to $\langle X|\overline{R}\rangle$. Indeed, in any vertex $v$ of $t - \text{Core}(w)$ which is not an intersection vertex for which there is a loop labeled by some element of $A$ or $B$ an expansion must be applied (because of the relations of the form $t^{-1}at = \varphi(a)$). For this reason, in a general t-opuntoid $\Gamma$ we call a $bud$ any vertex $v$ of $\Gamma$ such that the set of the loops centered in $v$ and labelled by elements of $A$ or $B$ is non-empty. In [5] it is proved that the last construction, called Construction 5, applied to a bud $v$ of a t-opuntoid $\Gamma$ generates (after finitely many steps) a new t-opuntoid $\Gamma'$ obtained from $\Gamma$ by adding a new $S$-lobe $\Delta$ connected to $v$ by a $t$-edge. This construction can be applied iteratively to obtain a directed system of t-opuntoid automata, and the direct limit of such system is $A(\overline{X}, R_{HNw}; w)$. For our purpose it is enough to know that Construction 5 leaves $\Gamma$ unaffected and it depends locally on the $S$-lobe $\Delta'$ containing the bud $v$ and not on the whole $\Gamma$. For this reason, for two generic $S$-lobes $\Delta, \Delta'$ which are quotient of Schützenberger automata of some elements of $S$, we write $\Delta' \xrightarrow{p,q,t} \Delta \xrightarrow{p^{-1},t^{-1},q} \Delta$ whenever applying Construction 5 to $\Delta'$ at the vertex $p$ which is a bud, we obtain a $t$-opuntoid consisting of two adjacent $S$-lobes $\Delta, \Delta'$ where $q$ is a vertex of $\Delta$ and $(p, q)$ ($(q, p)$) is an intersection pair. Note that, by the $t$-assimilation property, all the intersection vertices of $\Delta$ are connected to each other by paths labeled by elements of $A$ or $B$ depending on the fact that $(p, q)$ or $(q, p)$ is the intersection pair. Therefore an external lobe type is a pair $(\Omega, \Xi)$ consisting of an $S$-lobe $\Omega$ which is a quotient of a Schützenberger automaton of some word on $X$, and a set $\Xi$ of vertices of $\Omega$ (called the root of $(\Omega, \Xi)$) connected to each other by paths labeled by $A$ ($B$). Since $S$ is finite, for the fix presentation $\langle \overline{X}|R_{HN} \rangle$ there are finitely many (up to isomorphism) external lobe types $(\Omega_1, \Xi_1), \ldots, (\Omega_K, \Xi_K)$ for some positive integer $K$, and we also denote by $B_i$ for $i \in \{1, \ldots, K\}$ the corresponding set of buds of $\Omega_i$. We add to the previous set of external lobe types the element $(\Omega_0, \Xi_0)$ where $\Omega_0$ is the underlying inverse word graph of $t - \text{Core}(w)$, $\Xi_0 = \{\emptyset\}$ and we denote by $B_0$ the set of buds of $\Omega_0$ (notice that $\Omega_0$ is not an $S$-lobe). We refer to the pair $(\Omega_j, \Xi_j)$ as
the lobe of type \( j \). Therefore, for \( 0 < h \leq K \) and \( 0 \leq k \leq K \) we write \( k \xrightarrow{p,t,q} h \) \((k \xrightarrow{p,t^{-1},q} h)\) whenever applying Construction 5 to the \( t \)-opuntoid \( \Omega_h \) at the vertex \( p \in B_k \) we obtain an \( S \)-lobe \( \Delta \) such that \((p,q')((q',p))\) is an intersection pair for some vertex \( q' \) of \( \Delta \), and there is an isomorphism \( \varphi_\Delta \) from \( \Delta \) onto \( \Omega_h \) sending \( q' \) into \( q \in \Xi_h \) and consequently, by the assimilation property, all the intersection vertices of \( \Delta \) are sent onto \( \Xi_h \). Intuitively \( A(X, R_{HNN}; w) \) is built starting from \( \Omega_0 \) by patching iteratively lobes from \( \{ \Omega_1, \ldots, \Omega_K \} \). More precisely, if \( \Gamma \) is the \( t \)-opuntoid obtained by this patching procedure at a certain step, then if \( p \) is a bud of some \( S \)-lobe \( \Delta \) of \( \Gamma \) isomorphic to some \( \Omega_k \), then we glue an \( S \)-lobe \( \Delta' \) isomorphic to \( \Omega_k \) with the constrain that either \( k \xrightarrow{p,t,q} h \) or \( k \xrightarrow{p,t^{-1},q} h \) depending on whether \((p,q)\) or \((q,p)\) is an intersection pair. Moreover if \( R_{\Delta'} \) is the set of intersection vertices of \( \Delta' \) with \( \Delta \), then there is a unique local isomorphism \( \varphi_{\Delta'} : \Delta' \rightarrow \Omega_h \) sending \( R_{\Delta'} \) onto \( \Xi_h \). In this case we say that the lobe \( \Delta' \) is of \textit{type} \( h \).

3 The language of Schützenberger automaton of HNN-extension

In this section we prove that the language of \( A(X, R_{HNN}; w) \) is a real time deterministic context free language (RDCFL) which is well know to satisfy many nice features, like decidable equivalence problem, a linear time membership algorithm and many other closure properties (see for instance \cite{3}). By these facts we derive a polynomial time algorithm to solve the world problem for \( S^* \). This extends the decidability result in \cite{5}.

\textbf{Definition 1.} We associate to \( A(X, R_{HNN}; w) \) the real-time deterministic pushdown automaton \( \mathcal{P}_w = (Q, \overline{X} \cup \overline{X}^{-1}, \Sigma_w, \delta, q_0, \bot, \{ \beta \}) \) where: \( Q = \bigcup_{h \in [0,K]} V(\Omega_h) \) is the (finite) set of states, \( \overline{X} \cup \overline{X}^{-1} \) is the input alphabet, \( \Sigma_w = \{ (i,j,p,y,q) | i \in [0,K], p \in B_i, q \in \Xi_j, i \xrightarrow{p,y,q} j, y \in \{ t, t^{-1} \} \cup \{ 0, 0, \alpha, -\alpha \} \} \) is the (finite) stack alphabet whose elements will be denoted in the sequel by capital letters, \( q_0 = \alpha \in V(\Omega_0) \) is the initial state, \( \bot = (0,0,\alpha,-\alpha) \) is the initial stack symbol, where \( \alpha, \beta \) are the initial and final state of \( t - \text{Core}(w) \).

The transition function \( \delta : Q \times (\overline{X} \cup \overline{X}^{-1}) \times \Sigma_w \rightarrow Q \times \Sigma_w \) is defined as following. Let \( M = (i,j,p,y,q) \in \Sigma_w \), \( x \in X \cup X^{-1} \) and \( q_1, q_2 \in \Delta \) then

(i) Inside a lobe. If \((q_1, x, q_2)\) is an edge of the lobe \( \Omega_j \), then \( \delta(q_1, x, M) = (q_2, M) \).

(ii) Pass into a new lobe. If \((q_1, x, q_2)\) is not an edge of the lobe \( \Omega_j \) \((\text{then } x \in \{ t, t^{-1} \})\), and we have that either \( x \neq y^{-1} \), or \( q_1 \notin \Xi_j \). Then \( \delta(q_1, x, M) = (q_2, NM) \) with \( N = (j, h, q_1, x, q_2) \) if and only if \( j \xrightarrow{q_1, x, q_2} h \).

(iii) Go back into a yet visited lobe. If \((q_1, x, q_2)\) is not an edge of the lobe \( \Omega_j \) \((\text{then } x \in \{ t, t^{-1} \})\), \( x = y^{-1} \) and \( q_1 \in \Xi_j \). If \( y = t (y = t^{-1}) \) let \((q, u, q_1)\) be a path of \( \Omega_j \) for some \( u \in A \) \((u \in B) \). Then \( \delta(q_1, x, M) = (q_2, \epsilon) \) where \( q_2 \) is the vertex of \( \Omega_i \) such that \((p, \varphi(u), q_2), u \in A \text{ or } (p, \varphi^{-1}(u), q_2), u \in B \) is a path in \( \Omega_i \).

\textbf{Theorem 1.} The languages recognized by \( A(X, R_{HNN}; w) \) and \( \mathcal{P}_w \) coincide.
Proof. We have to prove that \((\alpha, z, \nu)\) is a path in \(\mathcal{A}(X, R_{HN}; w)\) if and only if
\((q_0, \perp, z')^n(q, \prod_{j=1}^{m_j} M_j, \perp, z')\)
is a computation of \(P_w\) where \(z, z' \in \langle X \cup X^{-1}\rangle^+\), \(\prod_{j=1}^{m_j} M_j \subseteq \Sigma^+_w\) and the last element on the stack is \(M_k = (k, h_s, v, y, v')\). The “only if” part is proved by induction on \(n\) by showing that there is a lobe \(\Delta\) in \(\mathcal{A}(X, R_{HN}; w)\) of type \(h_s\) such that \(\nu \in V(\Delta), q = \varphi(\nu)\). It is clear that the base case \(n = 1\) occurs only if the computation is either of type (i) or (ii) and it is left to the reader. Suppose that the statement holds for each computation of length \(n-1\), and let \(z''a = z\) for some \(z'' \in \langle X \cup X^{-1}\rangle^+\), \(a \in \overline{X} \cup \overline{X}^{-1}\). Since there are no \(\varepsilon\)-moves we have
\((q_0, \perp, z''a z')^n(q', \gamma', a z') \vdash (q, M \nu'', z')\)
for some \(\gamma', \nu'' \in \Sigma^+_w\). Now the \(n^{th}\) step could be any type of transitions in \(P_w\). For instance assume that the \(n^{th}\) step is obtained applying rule (iii) with \(a = t^{-1}\) and so \(y = t\) (we leave the other cases to the reader). Thus, in this case \(\gamma' = N \nu''\) with \(N = (h_s, h_{s-1}, p, t, p')\) and by the induction hypothesis there are a lobe \(\Delta'\) of
\(\mathcal{A}(X, R_{HN}; w)\) and a vertex \(\nu' \in V(\Delta')\) such that \(\varphi(\Delta')(\nu') = q'\) and \((\alpha, z'', \nu')\)
is a path in \(\mathcal{A}(X, R_{HN}; w)\). By the definition of rule (iii) \(q' \in \Xi_{h_{s-1}}\) and there is a path \((p', u, q')\) of \(\Omega_{h_{s-1}}\) for some \(u \in A\) and \((p, \varphi(u), q)\) is the corresponding path in \(\Omega_{h_s}\). Since \(h_s \xrightarrow{p, \nu'} h_{s-1}\) and \(\Delta'\) is a lobe of type \(h_{s-1}\), then there is a lobe \(\Delta\) of \(\mathcal{A}(X, R_{HN}; w)\) of type \(h_s\) such that \(\Delta \xrightarrow{p, \nu'} \Delta'\), and there exist local isomorphisms \(\varphi_\Delta : \Delta \to \Omega_{h_s}, \varphi_{\Delta'} : \Delta' \to \Omega_{h_{s-1}}\) such that \(\varphi_\Delta^{-1}(p) = r, \varphi_{\Delta'}^{-1}(p') = r'\) and \((r, r')\) is an intersection pair. Let \(\varphi^{-1}_\Delta(q') = \nu'\) and \(\varphi^{-1}_\Delta(q) = \nu, \) thus by (iii) and the assimilation property it is not difficult to see that \((\nu, \nu')\) is also an intersection pair. Hence \((\nu', t^{-1}, \nu)\) is an edge in \(\mathcal{A}(X, R_{HN}; w)\) and \((\alpha, z, \nu)\) is a path of \(\mathcal{A}(X, R_{HN}; w)\) with \(\nu \in V(\Delta), q = \varphi(\nu)\) and \(\Delta\) is a lobe of type \(h_s\).

Let us prove the “if” part. Thus, let \((\alpha, z, \nu)\) be a path in \(\mathcal{A}(X, R_{HN}; w)\) with \(\nu \in V(\Delta)\) for some lobe \(\Delta\) of the Schützenberger automaton. We prove by induction on \(n = |z|\) that there is a computation \((q_0, \perp, z z')^n(\varphi(\nu), \gamma', z')\).
The base case \(n = 1\) is left to the reader. So assume that the statement holds for each word of length less than \(n\). Let \(z = z''a\) for some \(a \in \overline{X} \cup \overline{X}^{-1}\). By the induction hypothesis the path \((\alpha, z'', \nu')\) corresponds to the computation
\((q_0, \perp, z''a z')^n(\varphi_\Delta(\nu'), \gamma', a z')\)
where \(\nu' \in V(\Delta')\). Here there are several cases to consider that depend on the edge \((\nu', a, \nu)\). If \((\nu', a, \nu)\) belongs to \(\Delta'\), which can be either a lobe of the underlying graph of the \(t - Core(w)\), then rule (i) is applied. Thus we can assume that \((\nu', a, \nu)\) belongs to neither \(t - Core(w)\) nor an external lobe. In this case we necessarily have \(a \in \{t, t^{-1}\}\), but we still have to discriminate between the case whether we are entering in a new lobe or we are coming back to a lobe which has been already visited. This depends on whether \(\nu'\) is in the root \(R_\Delta\) of \(\Delta'\) and \(\nu\) belongs to a lobe already visited. This fact is encoded in the last element \(M = (i, j, p, y, q)\) of \(\gamma'\) in the stack. Indeed, \(j\) is the type of the lobe \(\Delta'\), \(i\) is the type of a lobe \(\Delta\) and the last point that the path \((\alpha, z', \nu')\) crosses the two lobes \(\Delta, \Delta'\) is when it passes from the edge \((\varphi^{-1}_\Delta(p), y, \varphi^{-1}_\Delta(q))\). Thus if for instance \(a = y\) and \(\nu \in R_\Delta\), then \(\varphi(\nu) \in \Xi_j\) and so rule (iii) must be applied, otherwise
we are entering in a new lobe and so rule (ii) applies. The details are left to
the reader.

The decidability of the word problem for HNN-extensions is therefore a conse-
quence of the decidability of the equivalence problem for RDCFL. A primitive
recursive complexity upper bound for such problem was given by C. Stirling
[7] and at the best of our knowledge there is no better bound up to now. How-
ever in our context the equivalence between the languages recognized by the
Schützenberger automata of two words for the HNN-extension can be checked
using the membership problem for RDCFL. Since this problem is linear in
the length of a word [3], the algorithmic cost for the word problem is given
by the construction of the grammar. Assuming that the presentation for $S^*$
does not belong to the input, the construction of all the external lobe types
can be assumed as a constant cost. Thus the effective cost is given by the con-
struction of the $t − Core(w)$, and using arguments similar to [2] it is possible
to prove that this construction requires a number of steps polynomial on $|w|$, from which we get

**Corollary 1.** There is a polynomial time algorithm for the word problem for
HNN-extension of finite inverse semigroup.

We hope that, as in [1], Theorem 1 can be used to solve other problems on
HNN-extensions, like for instance the solvability of some families of equations.

**Acknowledgement**

The work is partially supported by the European Regional Development
Fund through the programme COMPETE and by the Portuguese Gov-
ernment through the FCT – Fundação para a Ciência e a Tecnolo-
gia under the projects PEst-C/MAT/UI0144/2011, CANTE-PTDC/EIA-
CCO/101904/2008. The second author also acknowledges the support of the
FCT project SFRH/BPD/65428/2009.

**References**

2. A. Cherubini, C. Nuccio, and E. Rodaro. Amalgams of finite inverse semigroups and
5. E. Rodaro and A. Cherubini. Decidability of the word problem in Yamamura's HNN-
7. C. Stirling. Deciding dpda equivalence is primitive recursive. In Peter Widmayer, Stephan
   Eilenbenz, Francisco Triguero, Rafael Morales, Ricardo Conejo, and Matthew Hennessy,
editors, *Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Com-
8. A. Yamamura. HNN extensions of inverse semigroups and applications. *Internat. J.
Pattern-avoiding polyominoes

Daniela Battaglino\textsuperscript{1}, Mathilde Bouvel\textsuperscript{2}, Andrea Frosini\textsuperscript{3}, Simone Rinaldi\textsuperscript{1}, Samanta Socci\textsuperscript{1}, and Laurent Vuillon\textsuperscript{4}

\textsuperscript{1} Dipartimento di Scienze Matematiche ed Informatiche, Pian dei Mantellini, 44, 53100, Siena, Italy
\textsuperscript{2} LaBRI UMR 5800, Univ. Bordeaux and CNRS, 33405 Talence, France
\textsuperscript{3} Dipartimento di Sistemi e Informatica, viale Morgagni 65, 50134, Firenze, Italy
\textsuperscript{4} Laboratoire de Mathématiques, UMR 5127 CNRS, Université de Savoie, 73376 Le Bourget du Lac, France

1 Patterns in matrices and in polyominoes

1.1 Definitions

The concept of a pattern within a combinatorial structure is undoubtedly one of the most investigated notions in combinatorics. It has been deeply studied for permutations, starting first with [8]. Analogous definitions were provided in the context of many other structures, such as set partitions [11], words [2], trees [9], and paths [1]. The aim of the present work is to define an analogous notion of pattern in the context of polyominoes and to present some guidelines to investigate this notion. We refer to [3] for the definitions on permutations/pattern-avoiding permutations, and to [4] for the definitions on polyominoes.

As usual, we represent a polyomino $P$ as a binary matrix whose dimensions are those of the minimal bounding rectangle of $P$, and where an entry $(i, j)$ is equal to 1 if the position $(i, j)$ contains a cell of $P$, 0 otherwise (see Fig. 1 (b, c)).

Letting $A, B$ be two binary matrices, we say that $B$ contains $A$ as a pattern ($A \preceq B$) if $A$ is a sub-matrix of $B$. Otherwise we say that $B$ avoids $A$.

Similarly, we say that a polyomino $P$ contains a matrix $A$ as a pattern ($A \preceq P$) if and only if the matrix representation of $P$ contains $A$. Otherwise we say that $P$ avoids $A$ (see Fig. 1).

We would like to mention that the partial order $\preceq_P$ and some combinatorial properties of families of polyominoes according to $\preceq_P$ have been considered in [6]. On the other side, our definition is slightly different from the one given in [7]. We denote $AV_p(A)$ the class of polyominoes avoiding $A$ and $AV_p(A) = \cap_{A \in \mathcal{A}} AV_p(A)$ for any set $\mathcal{A}$ of binary matrices.

1.2 A link between polyomino patterns and matrix patterns

When considering sets $AV_p(S)$, it is natural to distinguish the case where $S$ contains only patterns that are polyominoes (which will be denoted $p$-patterns) from the case where $S$ contains some matrices that are not polyominoes (which will be denoted $m$-patterns). However, defining that two sets of patterns $S$ and $S'$ are said to be equivalent when $AV_p(S) = AV_p(S')$, we may prove that:
**Proposition 1.** For every set $\mathcal{M}$ of $m$-patterns there is a set $\mathcal{P}(\mathcal{M})$ of $p$-patterns equivalent to $\mathcal{M}$.

In Figure 2, notice that the minimal $p$-patterns containing $H$ (resp. $V$) in the poset of $m$-patterns are $H_1, H_2$ (resp. $V_1, V_2$), and the reader can easily check that $\{H_1, H_2, V_1, V_2\}$ is the set of $p$-patterns equivalent to $\{H, V\}$. In addition:

**Proposition 2.** The class $AV_p(H, V) = AV_p(H_1, H_2, V_1, V_2)$ defines the class $C$ of convex polyominoes.

The proof of Proposition 1 is omitted for brevity. It is however non-constructive. In practice, given a finite set of $m$-patterns $\mathcal{M}$ we know that it is possible to find an equivalent set of $p$-patterns $\mathcal{P}(\mathcal{M})$, but we cannot state if $\mathcal{P}(\mathcal{M})$ is finite. In all the examples we have considered – for small dimensions of the matrices of $\mathcal{M}$ – the set $\mathcal{P}$ is finite. Moreover, in all these examples the dimensions of the patterns in $\mathcal{P}(\mathcal{M})$ depends on the shape of the matrices in $\mathcal{M}$, and this dependency seems to be tractable. Consequently, we conjecture that: For every finite set $\mathcal{M}$ of $m$-patterns there is a finite set $\mathcal{P}(\mathcal{M})$ of $p$-patterns equivalent to $\mathcal{M}$.

### 1.3 Stability of classes of polyominoes

The pattern relation $\preceq$ is a partial order on the class of binary matrices (or $m$-patterns). Hence the restriction of $\preceq$ to the class of polyominoes (or $p$-patterns), denoted $\preceq_p$, is a partial order on the class of polyominoes. We define $p$-stability as follows: A class $\mathcal{S}$ of polyominoes is $p$-stable if and only if $\mathcal{S}$ is downward closed w.r.t. $\preceq_p$, i.e. if and only if for every polyomino $P \in \mathcal{S}$, every $p$-pattern contained in $P$ belongs to $\mathcal{S}$. Because $\preceq_p$ is a partial order, $p$-stable sets share an important property of pattern-avoiding permutations [3]:

**Proposition 3.** A set $\mathcal{S}$ of polyominoes is $p$-stable if and only if there is an antichain $\mathcal{P}$ of $p$-patterns (i.e. a chain of pairwise incomparable $p$-patterns) such that $\mathcal{S} = AV_p(\mathcal{P})$. Moreover, $\mathcal{P}$ is unique and is called the basis of $\mathcal{S}$.

Notice that there may exist other sets $\mathcal{M}$ of $m$-patterns such that $\mathcal{S} = AV_p(\mathcal{M})$, which we denote $m$-bases of $\mathcal{S}$. They are interesting to provide efficient characterizations of $\mathcal{S}$, but they are not unique in general. An interesting problem would be to add constraints on $m$-bases that would determine uniquely the $m$-basis of a class.

![Fig. 1](image_url)

(a) $M_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

(b) $\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

(c) $\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Fig. 1. (a) a matrix pattern $M_1$; (b) a polyomino that contains $M_1$; (c) a polyomino that does not contain $M_1$.
2 Characterizing known classes of polyominoes by pattern avoidance

In this section, we are interested in some known classes of polyominoes, and we aim at describing them by pattern avoidance, as $AV_p(P)$.

2.1 Some $p$-stable classes

Let us consider first *L-convex polyominoes* – introduced by Castiglione and Restivo [6] – which can be defined as the convex polyominoes where each pair of cells can be connected by means of a path, internal to the polyomino, and using at most one change of direction. An example is shown on the right of Figure 4.

**Fig. 4.** A *L*-convex polyomino, and the matrices $M_1, M_2$ in the $m$-basis of this class.
The reader can easily check that \( L \)-convex polyominoes are a \( p \)-stable class, and an \( m \)-basis is given by the patterns \( H,V,M_1,M_2 \).

Figure 5 shows other \( p \)-stable classes of polyominoes (whose definitions are omitted for brevity) with a corresponding \( m \)-basis.

![Diagram of polyominoes and \( m \)-bases](image)

**Fig. 5.** Some \( p \)-stable classes of polyominoes with their \( m \)-bases.

### 2.2 Generalized polyomino patterns

We recall that a polyomino is \( k \)-convex if every pair of its cells can be connected by means of a monotone path, internal to the polyomino, having at most \( k \) changes of direction. Notice that 1-convex polyominoes correspond exactly to \( L \)-convex polyominoes, hence are a \( p \)-stable class of polyominoes. On the contrary, 2-convex polyominoes are not stable, as shown in Figure 6.

![Diagram of polyominoes and \( m \)-bases](image)

**Fig. 6.** The 2-convex polyomino (a) contains the 3-convex polyomino (b) as a \( p \)-pattern.

Proposition 3 then ensures that 2-convex polyominoes cannot be represented as a class of polyominoes avoiding a set of \( p \)-patterns. However, we can describe 2-convex polyominoes using a generalized notion of pattern avoidance. As in the context of pattern-avoiding permutations \([3,5]\), we impose adjacency constraints between two columns or rows by introducing special symbols. Here, we consider polyominoes possibly decorated with vertical (resp. horizontal) lines between consecutive columns (resp. rows). A polyomino \( P \) contains such...
a pattern $A$ if there is an occurrence of the underlying matrix of $A$ in $P$ such that the columns (resp. rows) of $P$ mapped by $c_i$ and $c_{i+1}$ (resp. $r_i$ and $r_{i+1}$) are consecutive every time there is a vertical (resp. horizontal) line between $c_i$ and $c_{i+1}$ (resp. $r_i$ and $r_{i+1}$). For example, the pattern $M$ of Figure 6 is not contained in (a) but is contained in (b) and (c). Generalized patterns may be used to characterize 2-convex polyominoes in terms of pattern avoidance:

**Proposition 4.** The class of 2-convex polyominoes is the class of polyominoes avoiding the following patterns:

\[
\begin{array}{cccc}
& 0 & \{} & 0 \\
0 & \{} & 0 & 0
\end{array}, \quad
\begin{array}{cccc}
0 & \{} & 1 & 1 \\
1 & \{} & 1 & 0
\end{array}, \quad
\begin{array}{cccc}
0 & \{} & 0 & 1 \\
0 & \{} & 1 & 1
\end{array}, \quad
\begin{array}{cccc}
0 & \{} & 1 & 0 \\
0 & \{} & 0 & 1
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}, \quad
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1
\end{array}, \quad
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}, \quad
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1
\end{array}
\]

The proof of Proposition 4 (which is omitted for brevity) may be generalized to provide a characterization of the class of $k$-convex polyominoes, with $k > 2$, by avoidance of generalized patterns.

### 3 Defining classes of polyominoes by pattern avoidance

New classes of polyominoes may also be defined by means of pattern avoidance. An interesting example is the class $AV_p(M_1, M_2)$, where $M_1, M_2$ are defined in Figure 4. We call $AV_p(M_1, M_2)$ the class of $L$-polyominoes, because this class is obtained by removing the convexity constraints from $L$-convex polyominoes. $L$-polyominoes enjoy a nice tomographical property [10]:

**Proposition 5.** Every $L$-polyomino is uniquely determined by its horizontal and vertical projections.

$L$-polyominoes may also be characterized by a geometric property, expressed in Proposition 6 and illustrated in Figure 7. Given a polyomino $P$, and two of its rows $r_1, r_2$ (resp. columns $c_1, c_2$), we write $r_1 \leq r_2$ (resp. $c_1 \leq c_2$) to mean that $r_1$ is geometrically included in $r_2$: for every cell of $r_1$ (resp. $c_1$) there is a cell of $r_2$ (resp. $c_2$) in the same column (resp. row) of $P$.

**Proposition 6.** The class of $L$-polyominoes coincides with the class of the polyominoes where every pair of rows (resp. columns) are comparable for $\leq$.

### 4 Further work

Because they provide a new point of view on well-studied objects, patterns in polyominoes open the way to many challenging problems on these objects, and we list only two of them here.
Fig. 7. (a) a L-polyomino, where every pair of rows and columns are comparable; for instance, \( c_1 < c_2 \); (b) a polyomino which is not a L-polyomino: the row \( r_1 \) is comparable neither with \( r_2 \) nor with \( r_3 \).

In one direction, we may study the poset \( (\mathcal{P}, \preceq_p) \) of polyominoes from an order-theoretic perspective. We can prove that the poset \( (\mathcal{P}, \preceq_p) \) of polyominoes strictly contains the poset \( (\mathcal{S}, \preceq_s) \) of permutations ordered by the usual pattern-containment order. We can also prove that \( \mathcal{P} \) is a ranked posed, and that the rank of a polyomino \( P \) is given by the semi-perimeter of the minimal bounding rectangle of \( P \). Moreover, \( \mathcal{P} \) contains an infinite anti-chain. An interesting problem would be to find a natural class of polyominoes - possibly defined in terms of connectivity/convexity constraints - with an infinite basis.

In another direction, we may use \( m \)-patterns to characterize families of polyominoes, or even the whole class \( \mathcal{P} \) of polyominoes, among all binary matrices. We have obtained a first result in the direction by characterizing the class \( \mathcal{C} \) of convex polyominoes within the class of binary matrices:

**Proposition 7.** The class of convex polyominoes can be expressed as the class of matrices avoiding the following set of generalized patterns:

\[
\begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\]

It is however an open question to know whether there is a set \( \mathcal{M} \) of \( m \)-patterns such that the whole class of polyominoes \( \mathcal{P} \) is equal to \( AV_m(\mathcal{M}) \).

**References**

Reset Regular Decomposition Complexity of Regular Ideal Languages*

(Extended Abstract)

Rogério Reis, Emanuele Rodaro**

Centro de Matemática, Universidade do Porto
R. Campo Alegre 687, 4169-007 Porto, Portugal
e-mail: rvr@dcc.fc.up.pt, emanuele.rodaro@fc.up.pt

Introduction

In this work, we study aspects related with the synchronization properties of words for finite automata, and we are not interested in automata as language recognizers but just on actions of its transition function $\delta$ on the set of states $Q$, thus, let us consider a deterministic finite automaton (DFA) as a tuple $\mathcal{A} = (Q, \Sigma, \delta)$, where the initial and final states are deliberately omitted from the definition. These automata are also refereed in the literature as semiautomata. But, because in some point of this work we refer to an automaton as a language recognizer, we also call a DFA to a tuple $\mathcal{B} = (Q', \Sigma', \delta', q_0, F)$ and the language recognized by $\mathcal{B}$ as the set $L(\mathcal{B}) = \{ u \in \Sigma^* : \delta'(q_0, u) \in F \}$. A DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is called synchronizing if there exists a word $w \in \Sigma^*$ “sending” all the states into a single one, i.e. $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any such word is said to be synchronizing (or reset) for the DFA $\mathcal{A}$. This notion has been widely studied since the work of Černý in 1964 [6] and his well known conjecture regarding the length of the shortest reset word. This states that any synchronizing automaton $\mathcal{A}$ with $n$ states admits at least a reset word $w$ with $|w| \leq (n - 1)^2$. For more information on synchronizing automata and the Černý’s conjecture we refer the reader to the survey by Volkov [7]. In this extended abstract we look to the recent results concerning a new approach to this conjecture from a language theoretic perspective. It is well known that the set ofreset words Syn($\mathcal{A}$) of a synchronizing automaton $\mathcal{A}$ is a two-sided regular ideal. A language $I \subseteq \Sigma^*$ is called a two-sided ideal (or simply an ideal) if $\Sigma^* I \Sigma^* \subseteq I$. In this work we will consider only ideal languages which are regular. In [3] the author notes that for a given ideal language $I$, the minimal DFA accepting $I$ is a synchronizing automaton with a sink state. This simple observation led the author to introduce a new notion of descriptonal complexity for the class of ideal languages. The reset complexity $rc(I)$ of an ideal $I$ is the number of states of the smallest synchronizing automaton $\mathcal{B}$ for which $I$ serves as the set of reset words of $\mathcal{B}$. The interesting fact of this notion is that the reset complexity can be exponentially smaller than its correspondent state complexity.

* Work partially supported by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT – Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011 and CANTEPTDC/EIA-CCO/101904/2008.
** Partially supported by FCT project SFRH/BPD/65428/2009.

Proceedings of ICTCS 2013
Reset Regular Decomposition Complexity of Regular Ideal Languages

There is also another interesting connection with Černý’s conjecture. It is straightforward to see that Černý’s conjecture holds if and only if \( \text{rdc}(I) \geq \sqrt{|I|} + 1 \) with \( |I| = \min\{|w| : w \in I\} \) holds for any ideal language \( I \). In [1] it is considered the special case of finitely generated synchronizing automata. This class, introduced in [4, 5], is formed by synchronizing automata \( \mathcal{A} \) such that \( \text{Syn}(\mathcal{A}) \) is a finitely generated ideal, i.e. \( \text{Syn}(\mathcal{A}) = \Sigma^*U\Sigma^* \) for some finite language \( U \). Motivated by the fact that Černý’s conjecture holds if and only if it holds for the class of strongly connected synchronizing automata, in [1] it is raised the question whether or not, given an ideal language \( I \) there is always a strongly connected synchronizing automaton having \( I \) as its set of reset words. In the same paper the authors consider the special case where the ideal \( P = \Sigma^*w\Sigma^* \) for some \( w \in \Sigma^* \), and it is presented an algorithm to generate a strongly connected synchronizing automaton \( \mathcal{B}_w \) with the same number \( |w| + 1 \) of states for the minimal DFA recognizing \( P \). Here we address the problem of exploring the general connection between ideal languages and strongly connected synchronizing automata. The main result, here presented, is that any non-unary ideal language can serve as the set of reset words of some strongly connected synchronizing automaton. This allows us to introduce another descriptional complexity measure of an ideal, called reset regular decomposition complexity. Given an ideal \( I \), the \( \text{rdc}(I) \) is the number of states of the smallest strongly connected synchronizing automaton \( \mathcal{A} \) with \( \text{Syn}(\mathcal{A}) = I \). It is clear that \( \text{rc}(I) \leq \text{rdc}(I) \) and also in this case we have that Černý’s conjecture holds if and only if \( \text{rdc}(I) \geq \sqrt{|I|} + 1 \) for any ideal language \( I \).

**Reset Regular Decomposition of an Ideal**

Since our aim is to prove that any ideal \( I \) is the set of reset words for some strongly connected synchronizing automaton, we introduce the following provisional class of strongly connected ideal language:

**Definition 1** An ideal language \( I \) is called strongly connected whenever \( I = \text{Syn}(\mathcal{A}) \) for some strongly connected synchronizing automaton \( \mathcal{A} \).

Let \( \text{SCSA}_\Sigma \) be the category of synchronizing automata with alphabet \( \Sigma \), where \( f : \mathcal{A} \to \mathcal{B} \) is an arrow between \( \mathcal{A} = (Q, \Sigma, \delta), \mathcal{B} = (T, \Sigma, \xi) \), if \( f \) is a homomorphism of DFA’s, i.e. \( f(\delta(q,a)) = \xi(f(q),a) \) for all \( q \in Q \) and \( a \in \Sigma \). Note that the strongly connectedness condition implies that \( f \) is surjective. To characterize the class of strongly connected ideals we need the following crucial notion of reset left regular decomposition:

**Definition 2** A reset left regular decomposition is a collection \( \{I_i\}_{i \in F} \) of disjoint left ideals \( I_i \) of \( \Sigma^* \) for some finite set \( F \) satisfying

i) For any \( a \in \Sigma \) and \( i \in F \), there is a \( j \in F \) such that \( I_i a \subseteq I_j \).

ii) Let \( I = \psi_{i \in F}I_i \). For any \( u \in \Sigma^* \) if there is an \( i \in F \) such that \( I_i u \subseteq I_i \), then \( u \in I \).
We say that an ideal $I$ has a (reset) left regular decomposition if there is a (reset) left regular decomposition $\{I_i\}_{i \in F}$ such that $I = \psi I F I_i$. Note that we have a dual notion of reset left regular decomposition by exchanging left ideals with right ideals and $I a, J w$ with $a I_i, u I$, respectively. Denote by $RLD_\Sigma$ (RRD$_\Sigma$) the category of the reset left regular decompositions, where an arrow $f : \{I_i\}_{i \in F} \to \{J_i\}_{i \in H}$ is any function $f : F \to H$ such that for any $i \in F$, there is an index $f(i) \in H$ with $I_i \subseteq J_{f(i)}$. Note that the reversal operator $u = u_1 \ldots u_r \mapsto u^R = u_r \ldots u_1$ is a bijection between the objects of $RLD_\Sigma$ and $RRD_\Sigma$. We have the following theorem.

**Theorem 3.** An ideal language $I$ is strongly connected if and only if it has a reset left regular decomposition. Moreover $RLD_\Sigma$ and $SCSA_\Sigma$ are equivalent categories via the two functors $A, I$ defined by:

- $A : RLD_\Sigma \to SCSA_\Sigma$ such that,
  $$A(\{I_i\}_{i \in F}) = (\{I_i\}_{i \in F}, \Sigma, \eta)$$
  with $\eta(I_i, a) = I_j$ for $a \in \Sigma$ if and only if $I_i a \subseteq I_j$, and if $f : \{I_i\}_{i \in F} \to \{J_i\}_{i \in H}$ then $A(f)$ is the homomorphism $\varphi : A(\{I_i\}_{i \in F}) \to A(\{J_i\}_{i \in H})$ defined by $\varphi(I_i) = J_m$ where $I_i \subseteq J_m$;

- $I : SCSA_\Sigma \to RLD_\Sigma$, such that, for $\mathcal{A} = (Q, \Sigma, \delta)$,
  $$I(\mathcal{A}) = \{I(\mathcal{A})_q\}_{q \in Q}$$
  where $I(\mathcal{A})_q = \{u \in \Sigma^* : \delta(Q, u) = q\}$, and if $\varphi : \mathcal{A} \to \mathcal{B}$ is an arrow between $\mathcal{A} = (Q, \Sigma, \delta)$ and $\mathcal{B} = (T, \Sigma, \xi)$, then $I(\varphi)$ is the arrow defined by $f : Q \to T$ which sends $q \mapsto \varphi(q)$.

The following corollary characterizes the ideals on a unary alphabet which are strongly connected.

**Corollary 4** Let $I$ be an ideal over a unary alphabet $\Sigma = \{a\}$. Then $I$ is strongly connected if and only if $I = \Sigma^*$.

Therefore, we can assume henceforth that the ideals considered are taken over an alphabet $\Sigma$ with more than one element.

Given a strongly connected ideal language $I$ with $\text{Syn}(\mathcal{B}) = I$ for some strongly connected synchronizing automaton $\mathcal{B} = (Q, \Sigma, \delta)$, there is an obvious way to calculate the associated reset left regular decomposition $I(\mathcal{B})$. It is well known that $I$ is recognized by the power automaton of $\mathcal{B}$ defined by $\mathcal{P}(\mathcal{B}) = \langle 2^Q, \Sigma, \delta, Q, \{q\} : q \in Q\rangle$, where $2^Q$ denotes the set of subsets of $Q$, the initial state is the set $Q$ and the final set of states is formed by all the singletons. Thus, for each $q \in Q$ we can associate the DFA $\mathcal{P}(\mathcal{B})_q = \langle 2^Q, \Sigma, \delta, Q, \{q\} \rangle$ and so we can calculate the associated reset left regular decomposition by $I(\mathcal{B}) = \{L(\mathcal{P}(\mathcal{B})_q)\}_{q \in Q}$. Consider, as an example, the Cerny’s series $\mathcal{C}_n = \{\{1, \ldots, n\}, \{a, b\}, \delta_n\}$, where $a$ acts like a cyclic permutation $\delta_n(i, a) = i + 1$ for $i = 1, \ldots, n - 1$ and $\delta_n(n, a) = 1$, while $b$ fixes all the states except the last one: $\delta_n(i, b) = i$ for $i = 1, \ldots, n - 1$ and $\delta_n(n, b) = 1$ (see Fig. 1). For example, in the case of $\mathcal{C}_4$ the associated reset left regular
Theorem 6. We devote the rest of the paper to sketch the proof of the following main result:

\[ L[\mathcal{P}(\mathcal{C})_1] = ((a^*b)(b + ab + a^4)(a^3b + (a^2b(b + a^2)^*ab))(b + ab^*a^3) + ((ab^*ab)(b + a^2)^*ab)^*a^2b)(b + ((ab^*ab^*)(a(a + b)))^*) \]
\[ L[\mathcal{P}(\mathcal{C})_2] = L[\mathcal{P}(\mathcal{C})_1]ab^* \]
\[ L[\mathcal{P}(\mathcal{C})_3] = L[\mathcal{P}(\mathcal{C})_1]ab^*ab^* \]
\[ L[\mathcal{P}(\mathcal{C})_4] = L[\mathcal{P}(\mathcal{C})_1]ab^*ab^*a. \]

In general, for \( \mathcal{C}_n \) it is not difficult to see that \( |\delta_n(\{1, \ldots, n\}, u)| = 1 \) and \( |\delta_n(\{1, \ldots, n\}, u)| > 1 \) for some word \( u \in \{a, b\}^* \) and a letter \( x \in \{a, b\} \) if and only if \( \delta_n(\{1, \ldots, n\}, u) = \{n, 1\} \) and \( x = b \). Thus, if \( |\delta_n(Q, w)| = 1 \), then there is a prefix \( w'b \) of \( w \) with \( \delta_n(Q, w') = \{n, 1\} \). Therefore, it is straightforward to check that in this case the decompositions are given by

\[ L[\mathcal{P}(\mathcal{C})_1] = \{w \in \Sigma^* : \delta_n(\{1, \ldots, n\}, w) = \{1\}\} \]
\[ L[\mathcal{P}(\mathcal{C})_\ell] = L[\mathcal{P}(\mathcal{C})_1](ab^*)^{\ell-1} \text{ for } \ell = 2, \ldots, n-1 \]
\[ L[\mathcal{P}(\mathcal{C})_n] = L[\mathcal{P}(\mathcal{C})_1](ab^*)^{n-2}a. \]

By Theorem 3 if \( I \) is strongly connected, then the set \( \mathcal{R}(I) \) of all the reset left regular decompositions of \( I \) is non-empty.

**Proposition 5** Let \( I \) be a strongly connected ideal, the set \( \mathcal{R}(I) \) endowed with \( \{I_i\}_{i \in F} \land \{J_k\}_{k \in H} = \{I_i \cap J_k\}_{(i,k) \in F \times H} \) is a \( \land \)-semilattice.

We devote the rest of the paper to sketch the proof of the following main result:

**Theorem 6.** Let \( I \subseteq \Sigma^* \) be an ideal language, then \( I \) is a strongly connected ideal language.

The proof of this Theorem pass from the notion of reset left regular decomposition. Indeed, instead of trying to build a strongly connected automaton for which \( I \) serves as the set of reset words, we prove that \( I^R \) has a reset right regular decomposition \( \{I_i\}_{i \in F} \). Thus \( \{I_i^R\}_{i \in F} \) is a reset left regular decomposition of \( I \), from which we get that \( I \) is strongly connected from Theorem 3. Therefore we have reduced ourself to prove that any ideal \( J \) has a reset right regular decomposition. Let \( \mathcal{C} = \langle Q, \Sigma, \delta, q_0, s \rangle \) be the automaton with \( n \) states and a sink state \( s \) which recognized \( J \). Note that for such an automaton \( |Q \cdot u| = 1 \) if and only if \( Q \cdot u = \{s\} \). Fix a word \( u \in \Sigma^* \) and a subset \( H \subseteq Q \).

---

Proceedings of ICTCS 2013
Assume \( u = u_1 \ldots u_r \) for \( u_1, \ldots, u_r \in \Sigma \) and \( r = |u| \). For \( 0 \leq i < j \leq r \) we use the standard notation \( u[i,j] \) to indicate the factor \( u_i u_{i+1} \ldots u_j \) if \( i > 0 \), otherwise \( u[0,j] = u_1 \ldots u_j \) with the convention that \( u[0,0] = \epsilon \) and \( u[i,i] = u_i \) if \( i > 0 \). We now introduce a function which is the key in proving that \( J \) has a reset right regular decomposition. Let \( m = \frac{2^t - 1}{n + 1} + 1 \) and let \( Z_m \) be the ring of the integers modulo \( m \). For an integer \( t \geq 1 \), \( [2^Q]_t \) denotes the set of subsets of \( Q \) of cardinality \( t \). Let \( T_t = Z_m([2^Q]_t \cup \Sigma) \) be the free \( Z_m \)-module on \([2^Q]_t \cup \Sigma\). Fix an element \( u \in \Sigma^* \) with \( u = u_1 \ldots u_r \) and \( H \subseteq Q \) with \( |H| > 1 \). Let \( j \) be the biggest index \( 1 \leq j \leq r \) such that \( |H \cdot u[1,j]| > 1 \) and if \( j < n \), then \( |H \cdot u[1,j+1]| = 1 \). The set \( S = H \cdot u[1,j] \) is called the last set of \((H,u)\). Let \( i \) be the index \( 1 \leq i \leq r \) such that \( u[i,j] \) is the maximal factor of \( u \) with \( |S| = |H \cdot u[0,k]| \) for all \( i \leq k \leq j \). The tail of \((H,u)\) is the element of \( Z_m([2^Q]_t \cup \Sigma) \) with \( t = |S| \geq 2 \) defined by

\[
T(H,u) = \begin{cases} 
\sum_{k=1}^{j-1} (H \cdot u[0,k] + u[k+1,k+1]), & \text{if } u[0,j] = u \\
\sum_{k=i}^{j} (H \cdot u[0,k] + u[k+1,k+1]), & \text{otherwise}
\end{cases}
\]

Consider the set \( T = \cup_{t=2}^n T_t \). We give to \( T \) a structure of semigroup by introducing an internal binary operation \( \circ \) defined in the following way. Let \( T_1 \in T_i, T_2 \in T_j \), then

\[
T_1 \circ T_2 = \begin{cases} 
T_{\min(i,j)} & \text{if } i \neq j \\
T_1 + T_2 & \text{otherwise}
\end{cases}
\]

Note that \((T,\circ)\) has a graded structure with respect to the semilattice \(([2,n],\min)\), i.e. \( T_i \circ T_j \subseteq T_{\min(i,j)} \). Let \( u \in \Sigma^* \), the tail map is the function \( \tau_u : 2^Q \rightarrow T \) defined by

\[
\tau_u(H) = \begin{cases} 
T(H,u) & \text{if } |H| > 1 \\
0_n & \text{otherwise}
\end{cases}
\]

for any \( H \subseteq Q \), and \( 0_n \) is the zero of \( T_n \). We denote by Hom\((A,B)\) the set of the maps \( f : A \rightarrow B \). The following lemma, whose proof is an easy consequence of the definitions, is crucial in proving that \( J \) has a reset right regular decomposition.

**Lemma 1.** Consider the map \( \mu : \Sigma^* \rightarrow \text{Hom}(2^Q,T) \) defined by \( \mu(u) = \tau_u \), then Ker\((\mu)\) is a left congruence on \( \Sigma^* \).

In the proof of Theorem 6 it is shown that the equivalence classes \( \{J_i\}_{i \in H} \) of the equivalence relation \((J \times J) \cap \text{Ker} (\mu) \) are right ideals and the previous Lemma is used to prove that condition \( i \) of Definition 2 is fulfilled. The rest of the proof is devoted to prove that \( \{J_i\}_{i \in H} \) satisfies the reset condition \( ii \) of Definition 2. It is evident that an upper bound of the cardinality \( |H| \) of the decomposition \( \{J_i\}_{i \in H} \) is given by \( |\text{Hom}(2^Q,T)| \). Thus by Theorem 3 we have the following

**Corollary 7** Let \( I \) be an ideal language on \( \Sigma \) such that \( I^R \) has state complexity \( n \). Then there is a strongly connected synchronizing automaton \( B \) with \( N \)
states and \( \text{Syn}(\mathcal{B}) = I \) such that:

\[
N \leq m k^2 n^2 \left( \sum_{i=2}^{n} \frac{m(i)}{c} \right)^2
\]

where \( k = |\Sigma| \) and \( m = \left( \frac{2^{2m}}{m} + 1 \right) \).

This Corollary shows a double exponential upper bound for the number of states of the associated strongly connected automaton with respect to the state complexity of the reverse of the ideal language. It is unknown by the authors whether this bound is tight or not. In [1], for instance, it is shown an algorithm that given a principal ideal \( I = \Sigma^* w \Sigma^* \) with \( |w| = n \) in inputs, it returns a strongly connected synchronizing automaton with \( n + 1 \) states. Therefore in this case the bound is linear with respect to the state complexity of \( I^R \). Even more recently [2] it was proven that in the case \( I \) is finitely generated, \( \text{rdc}(I) \leq 2^{||I||} \) and this bound is tight. Therefore, upper and lower bounds of \( \text{rdc}(I) \) is an interesting topic that can (maybe) shed some light on the Černý’s conjecture. For instance, even a lower bound \( \text{rdc}(I) \geq \sqrt{||I||/c} \) for some constant \( c > 0 \) would be a major breakthrough for this conjecture and all the theory of synchronizing automata.

Acknowledgement

The work is partially supported by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT – Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011, CANTE-PTDC/EIA-CCO/101904/2008. The second author also acknowledges the support of the FCT project SFRH/BPD/65428/2009.

References

1. V.V. Gusev, M.I. Maslennikova, and E.V. Pribavkina. Principal ideal languages and synchronizing automata. in V. Halava, J. Karhumaki, Y. Matiyasevich (eds.) RuFiDimII, TUCS Lecture Notes, 17, 2012.
Validating Reconfigurations of Reo Circuits

Maurice H. ter Beek\textsuperscript{1}, Fabio Gadducci\textsuperscript{2}, and Francesco Santini\textsuperscript{3}

\textsuperscript{1} ISTI–CNR, Pisa, Italy
\textsuperscript{2} Department of Computer Science, University of Pisa, Italy
\textsuperscript{3} EPI Contraintes, INRIA, France

Abstract. We formalize dynamic reconfiguration of Reo circuits (multi-party interactions built from primitive channels) by graph transformation and apply it to a critical infrastructure controlling the business process of an e-banking scenario, in which reconfiguration is triggered as soon as the communication buffers reach specific predefined thresholds of congestion.

1 Introduction and Motivations

Reo [1] is a channel-based exogenous coordination language wherein complex coordinators, called connectors, are compositionally constructed from simpler ones. This graphical language has a strong formal basis and promotes loose coupling, distribution, mobility and dynamic reconfigurability. Application designers can use Reo as a “glue code” language for the compositional construction of connectors that orchestrate the cooperative behavior of instances of components or services in a component-based system or a service-oriented application.

We describe our current research on defining graph transformation based reconfigurations for Reo circuits [3]. We apply them to a real-world scenario from the Finance domain: a critical infrastructure controlling the business process of an e-banking system, inspired by a study of real-world requirements at Credit Suisse S.A. in Luxembourg [4], in which reconfiguration is triggered as soon as the communication buffers reach specific predefined thresholds of congestion.

In this scenario, people select shares they want to buy, put them into a basket and pay for them using electronic means like credit cards. Once in a while, an account statement is sent to clients by e-mail, summarizing their past transactions. As a case study, we show two different configurations of this e-banking system and a way to switch from one to the other (stemming from the assumption that the nature of the incoming requests may change over time). The first case represents a redundant yet safer configuration: if a request fails in either one of the two branches of the original configuration, the other branch provides a potential backup. In the second case, we assume that incoming orders are i) not replicated, but instead processed simultaneously by an upper and a lower branch and ii) arbitrarily distributed to the available services, in order to maximize the overall capacity of requests that the IT infrastructure can process.

We control the switching by setting variables to certain values through a graph rewriting model. The configurations are defined in Reo, the modeling and simulation processes in Modelica (http://modelica.org/), a non-proprietary, object-oriented, equation-based language for modeling complex physical systems.

\textsuperscript{Proceedings of ICTCS 2013}
Validating Reconfigurations of Reo Circuits

This paper is an extended abstract of [3]. In §2 we introduce Reo, while our e-banking scenario is introduced in §3. In §4 we define graph transformations for the dynamic reconfiguration described in our scenario and in §5 we evaluate their application. §6 contains our conclusions and ideas for future work. An appendix contains the basic notions on graph rewriting and the specific variant we use.

2 Reo

Reo facilitates the compositional construction of circuits: communication mediums coordinating interacting parties, each built from simple channels. A channel has exactly two ends that each have exactly one type: it is either a source end that accepts data items or a sink end that offers data items. Fig. 1(left) shows six primitive channels available for Reo users; Fig. 2 describes their behavior.

In a circuit, “data items” flow through “channels” (along edges) past “nodes”. Nodes are logical points where execution over different channels is synchronized. A node is either source, sink or mixed, depending on whether all channel ends coinciding on that node are source ends, sink ends or a combination of both. A mixed node non-deterministically selects and takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. The XOR circuit in Fig. 1(right) implements an exclusive router of messages: its intuitive behavior is that data obtained as input through node a is non-deterministically delivered to one of the output nodes b or c.

In this paper we consider Reo’s semantics based on constraint automata [2]; states represent the internal configurations of a circuit and transitions describe its atomic coordination steps. Formally, a transition is defined as a tuple of four elements: a source state, a synchronization constraint, a data constraint and a target state. A synchronization constraint specifies which nodes synchronize—i.e. through which nodes a data item flows—in some coordination.
step; a data constraint specifies which particular data items flow in such a step. Figure 3 shows the constraint automata for the channels and circuits presented in Fig. 1.

Our case study extensively uses a new primitive Reo channel, called lossyNfifo, graphically represented in Fig. 5 as a fifo channel with a dashed incoming arrow. It models an $n$-position buffer, whose informal semantics allows it to start losing data when the buffer is completely full; otherwise, all its slots are filled in order.

In Fig. 4(left) we formalize its behavior (for $n=2$) as a constraint automaton, in which variables $v_1$ and $v_2$ represent the two slots. In states $q_2$ and $q_4$ both are full, but it is still possible to receive elements through port $a$, and then discarding them (modeled by the self-loops on $q_2$ and $q_4$). Operations on port $b$ free one of the slots by consuming the first-stored message (it is a FIFO scheme) and then outputting it from the circuit to the following component connected to port $b$.

3 A Case Study

We model a service landscape which implements and enables the business process of the aforementioned e-banking system. The model is described using Reo, resulting in the circuit in Fig. 5 for “Case 1” below. Routing between services is realized by circuits that internally implement different Reo connectors and that contain buffers (lossyNfifo channels) that can run full. In “Case 1”, incoming requests are dropped and lost. We assume a stream of client orders flowing into the system, each has a certain business value in euro cents, comes

Fig. 3: Constraint automata for the common channels and the circuit in Fig. 1.

Fig. 4: Behavior of a lossyNfifo channel with two slots (left) and behavior of circuit 1 in Fig. 5 considering the two lossyNfifo channels to have one slot (right). The state labels denote the filling status of the two fifo channels: full or empty. The edge labels $In$, $Out_1$ and $Out_2$ stem from those shown in Fig. 5.
at a specific point in time and adds up to a concrete basket until that basket is complete.

**Case 1.** The input stream, produced by a client service, is replicated by circuit b for processing by an upper and a lower branch of pairwise identical services. When a request fails in either branch, the other provides a backup. In each branch, coordination between two consecutive services is modeled by circuit 2. Their output streams are merged, collected and forwarded from time to time to a mail service, which sends out account statements by e-mail, by circuit 3. The basket services handle the bookkeeping of all incoming requests until their corresponding baskets are complete, in which case the whole basket is sent to the payment service. Both services can fail, which is a behavior realized by their respective failure distributions. The collect services collect all successfully paid baskets and flush them out to the mail service, based on a given timetable. The payment services and the mail services use different delay distributions.

Fig. 4(right) shows the behavior of circuit 1 as a constraint automaton in which data constraints are omitted, i.e. only the synchronization constraints on nodes In, Out1 and Out2 are shown. These represent the endpoints where the client, basket and basket1 components perform their offer and accept operations.

**Case 2.** We now assume that incoming orders are no longer replicated and simultaneously processed by two branches, but they are arbitrarily distributed to the available services to maximize the overall capacity of requests that can be processed. The main difference w.r.t. “Case 1” is the use of different Reo circuits to coordinate the services being used. In fact, services behave in exactly the same way. In Fig. 6, the complete Reo circuit diagram for “Case 2” is shown.

We now implement a sort of switch that either selects model “Case 1” or “Case 2” for processing incoming requests. Different loads of requests (i.e. shares in this scenario) may lead to the loss of a large percentage of purchases, thus lowering the trust of customers in the offered service. From a theoretical point of view, a switch enables the dynamic reconfiguration of the Reo circuits.
Graph transformation was adopted early on for reconfiguring Reo specifications. We use a simple variant (cf. Appendix) of an elaborate proposal [5, 6]. A rewriting rule is specified by two graph morphisms, denoting how the left-hand side is rewritten into the right-hand side, while using the intermediate graphs to track the connection between the items on both sides. We only consider graphical representations of circuits and enrich their edges with a predicate, preserved by the graph morphisms, representing a QoS measure: the signal to start reconfiguring.

In rule 1, depicted in Fig. 7, the three graphs (circuits) composing it are arranged horizontally and the morphisms are suggested by the position of each item. The only ambiguity concerns the isolated nodes in the intermediate graph, which are mapped into the rightmost and leftmost node of the right-hand side. A predicate $f$, signaling the possible congestion of the channel, is easily obtained as an abstraction of the channel behavior by stating, e.g., that more than a certain amount of slots of the buffer are actually in use (filled). When the threshold is exceeded, the rule becomes enabled and it may be applied. It tends to replace the (first) component of the duplicated branch to turn it into a single branch with twice the buffering capacity. The need of reconfiguring also the connections between components is achieved by temporarily switching off and on the basket components, as long as they are empty: this is modeled by deleting and recreating the component after the step, while a predicate $e$ on the basket component signals that the rule can be applied only after the baskets were indeed emptied.

The result of applying rule 1 to the circuit of Fig. 5 is depicted in Fig. 8. The reconfiguration has changed the behavior of the overall circuit, even if the semantics of each channel is preserved. The initial part of the circuit has now passed to a linear treatment of the client input and this is reflected in the overall behavior. One of the lossyNfifo channels verifies the predicate $f$ and this fact enables the application of the rule. The baskets are still empty, while the predicates on the other edges are ignored: they are the same before and after the reconfiguration steps and thus irrelevant for the enabling of the rule. Subsequent applications of rule 2 (Fig. 9) and rule 3 (Fig. 10) will turn
the circuit into that of Fig. 6.

5 Simulation and Validation

Dymola (http://www.3ds.com/products/catia/portfolio/dymola) is a tool supporting graphical compositions of Modelica models and their fast simulation with symbolic pre-processing. Figure 11(left) shows such a model of “Case 3”, which implements the rewriting that switches from “Case 1” to “Case 2” whenever needed, i.e. when the maximum filling percentage among all lossyNfifo channels exceeds a (designer’s choice) warning threshold. Circuit 7 models the Reo coordination between Cases 1 and 2. Our simulation below starts delivering data to “Case 1”, after which it switches to “Case 2”, as formalized in §4. The “Rewriting Rule” block models rewriting “Case 1” to “Case 2”: when at least one of the lossyNfifo channels in “Case 1” is filled for more than a threshold percentage (the triggering condition), then circuit 7 waits for the complete emptying of all lossyNfifo channels before it resumes routing the incoming requests to “Case 2”.

The two outputs from the “Rewriting Rule” block are needed to i) open/close the dispensing of requests from circuit 7 and ii) decide which of the two cases circuit 7 is routing requests to. When switching cases this block thus uses these two discretely valued parameters to first close (open = 0) the output flow of messages to “Case 1”, and then, once all pending messages are served, switch the second flag (case1or2) to start routing messages to “Case 2” (reopening the output flow to it, i.e. open = 1). Its single input, whose value is continuously sampled, conveys the maximal filling status (in percentages) among all lossyNfifo channels in “Case 1”; the threshold is set to this value.

Fig. 11(right) shows a chart obtained by simulating “Case 3” with Dymola until time 420s. To limit the time we set the threshold to 10%, meaning the triggering condition is fired when the maximal filling status among all lossyNfifo channels is at least 10%. The chart shows a curve for each variable fifoCongestion, case1or2 and open. The first represents the only input of “Rewriting Rule”, while the other two represent its two outputs (cf. Fig. 11(left)). The point indicated by a star represents time 135s, when
the 10% threshold is reached and circuit 7 stops routing messages to "Case 1", i.e. open becomes equal to 0 (it is now closed). When all lossyNfifo channels in "Case 1" are empty, at time 392s (indicated by two stars), circuit 7 starts routing messages again (i.e. open = 0), this time to "Case 2" (i.e. from having value 1, now case1or2 becomes 0). Note that at time 200s, the collect services flush out their stored baskets to circuit 3, which is why fifoCongestion reaches 16%.

6 Discussion and Future Work

The scenario in this paper exemplifies the usefulness of (a simple variant of) graph transformation and its relevance in the context of Reo. The predicates used could be made more expressive, e.g. by associating values from a bounded partial order or a semiring, as in soft constraint satisfaction problems, to allow a more complex notion of rule matching. This could lead to symbolic graphs [7].

We could think of alternative reconfiguration rules, like for rule 1 in Fig. 12: The basket service connectors are included in the intermediate graphs and no predicate is required to hold. Intuitively, this corresponds to a kind of on-the-fly reconfiguration by which the state of the basket need not be empty, yet it is preserved during rewriting. This solution seems to increment flexibility, but the graph morphism on the left would no longer be injective, requiring a more difficult variant of the standard operational description of the rewriting step to preserve the uniqueness of the result of a rule application after a match is chosen.
Appendix: Graph Transformation

The generic name *algebraic graph transformation* indicates a class of formalisms adopted in the literature for the local manipulation of graphical structures. This is accomplished by choosing a set of rules, by precisely specifying the kind of objects to manipulate and by denoting how a rule can be applied to an object.

The best known proposal is probably the so-called DPO approach. It is summed up by the two squares in Fig. 13. Assuming to have just ordinary graphs (as in Reo’s graphical representation), the rule is specified by two graph morphisms, $l : L \rightarrow K$ and $r : K \rightarrow R$ (the former has to be injective, too), denoting how the left-hand side graph $L$ is rewritten into the right-hand side graph $R$, while the intermediate graphs are used to keep track of the connection between the items of the left-hand side and those of the right-hand side.

$$
\begin{array}{c}
\begin{array}{ccc}
L & \xrightarrow{l} & K \\
\downarrow{m} & & \downarrow{r} \\
G & \xrightarrow{l(K)} & R \\
\end{array}
\end{array}
$$

Fig. 13: A rule application in the DPO approach.

The idea is that both squares must be ‘minimal’. The DPO approach achieves this by requiring the squares to be what in categorical terms is called a *pushout*: e.g. $H$ is obtained as the disjoint union of $D$ and $R$, with the proviso that the items which already occur in $K$ are actually collapsed. Operationally, the application of a rule to a graph $G$ consists of three steps. First, the match morphism $m : L \rightarrow G$ is chosen, providing there is an occurrence of $L$ in $G$. Second, all items of $G$ matched by $L \setminus l(K)$ are removed, and those items identified by $l \circ m$ are coalesced, leading to context graph $D$. Third, the items of $R \setminus r(K)$ are added to $D$, further coalescing those nodes identified by $r$, obtaining the derived graph $H$. 

Proceedings of ICTCS 2013
For our e-banking scenario, we considered a simple variant of graphs adopted early on for modeling reconfiguration in Reo, in which each edge is actually equipped with a predicate and the morphisms between graphs actually preserve such predicates. All the relevant properties of the DPO approach (e.g., the uniqueness of the result of a rule application for a given match morphism) continue to hold also for our variant.
A complete classification of the expressiveness of interval logics of Allen’s relations over dense linear orders

Luca Aceto\textsuperscript{1}, Dario Della Monica\textsuperscript{1}, Anna Ingólfsdóttir\textsuperscript{1}, Angelo Montanari\textsuperscript{2}, and Guido Sciavicco\textsuperscript{3}

\textsuperscript{1} ICE-TCS, School of Computer Science, Reykjavik University, Iceland
\textsuperscript{2} Dept. of Mathematics and Computer Science, University of Udine, Italy
\textsuperscript{3} Dept. of Information Engineering and Communications, University of Murcia, Spain

1 Introduction

Interval reasoning (where time intervals, rather than time instants, are the primitive ontological entities) naturally arises in various fields of computer science and AI, ranging from hardware and real-time system verification to natural language processing, from constraint satisfaction to planning [2, 3, 6, 9–11]. The variety of binary relations between intervals in a linear order was first studied by Allen [2]. In [7], Halpern and Shoham introduced and systematically analyzed the (full) logic of Allen’s relations, called HS, that features one modality for each Allen’s relation. The undecidability of HS over most classes of linear orders motivated the search for (syntactic) HS fragments offering a good balance between expressiveness and decidability/complexity. A comparative analysis of the expressive power of the variety of HS fragments naturally sets the scene for such a search. This analysis is far from being trivial, because some HS modalities are definable in terms of others and such inter-definabilities may depend on the class of linear orders in which the logic is interpreted. Many classes of linear orders are of practical interest, including the class of all linear orders and the class of all dense (resp., discrete, finite) linear orders. In [5], Della Monica et al. gave a complete characterization of all expressively different subsets of HS modalities over all linear orders. Unfortunately, such a classification cannot be easily transferred to any other class of linear orders (proving a specific undefinability result amounts to providing a counterexample based on concrete linear orders belonging to the considered class). In this paper, we give a complete classification of the expressiveness of HS fragments over all dense linear orders. Undefinability results are essentially based on counterexamples referring to the linear order of \( \mathbb{R} \). However, the proposed constructions can be modified to deal with specific sub-classes of the class of all dense linear orders, e.g., the linear order of \( \mathbb{Q} \). As a final result, we show that there are exactly 966 expressively different HS fragments over (all) dense linear orders (over all linear orders, they are 1347), out of 4096 distinct subsets of HS modalities.

* This is an extended abstract of a paper that has been accepted for publication in the proceedings of TIME 2013 [1].

Proceedings of ICTCS 2013
For every modalities are defined as followas standard for the xoolean operators and propositionsc The semantics of the atomic proposition over F if p are defined by the grammar these relationsa we associate the fragment modality for each wllen’s relation [m] c With every subset set corresponding to the inverse relation each over thema thus associating a modality structures as Kripke structures and wllen’s relations as accessibility relations that isa tions strict intervals in a linear ordera often called diferent relations between two if intervalsc If we exclude equalitya there are fg di terval where a, b ∈ D, and a ≤ b. An interval is called a point (resp., strict) interval if a = b (resp., a < b). In this paper, we restrict ourselves to strict intervals. If we exclude equality, there are 12 diferent relations between two strict intervals in a linear order, often called Allen’s relations [2]: the six relations $R_A, R_L, R_B, R_E, R_D,$ and $R_O$ depicted in Figure 1 and the inverse ones, that is, $R_X = (R_X)^{-1}$, for each $X ∈ \{A, L, B, E, D, O\}$. We treat interval structures as Kripke structures and Allen’s relations as accessibility relations over them, thus associating a modality $\langle X \rangle$ with each Allen’s relation $R_X$. For each $X ∈ \{A, L, B, E, D, O\}$, the transpose of modality $\langle X \rangle$ is modality $\langle X \rangle^T$, corresponding to the inverse relation $R_X^T$ of $R_X$.

HS is a multi-modal logic with formulae built from a finite, non-empty set $\mathcal{AP}$ of atomic propositions, the propositional connectives $\lor$ and $\land$, and a modality for each Allen’s relation [7]. With every subset $\{R_{X_1}, \ldots, R_{X_k}\}$ of these relations, we associate the fragment $X_1X_2\ldots X_k$ of HS, whose formulae are defined by the grammar: $\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \langle X_1 \rangle \varphi \mid \ldots \mid \langle X_k \rangle \varphi$, where $p ∈ \mathcal{AP}$. The other propositional connectives and constants (e.g., $\land, \neg, \lor$) can be derived in the standard way, as well as the dual modalities (e.g., $[A] \varphi ≡ \neg[A] \neg \varphi$).

For a fragment $\mathcal{F} = X_1X_2\ldots X_k$ and a modality $\langle X \rangle$, we write $\langle X \rangle \in \mathcal{F}$ if $X ∈ \{X_1, \ldots, X_k\}$. Given two fragments $\mathcal{F}_1$ and $\mathcal{F}_2$, we write $\mathcal{F}_1 ⊆ \mathcal{F}_2$ if $\langle X \rangle ∈ \mathcal{F}_1$ implies $\langle X \rangle ∈ \mathcal{F}_2$, for every modality $\langle X \rangle$. Finally, for a fragment $\mathcal{F} = X_1X_2\ldots X_k$ and a formula $\varphi$, we write $\varphi \in \mathcal{F}$, or, equivalently, we say that $\varphi$ is an $\mathcal{F}$-formula, meaning that $\varphi$ belongs to the language of $\mathcal{F}$.

The (strict) semantics of HS is given in terms of interval models $M = \langle \mathcal{I}(\mathbb{D}), V \rangle$, where $\mathbb{D}$ is a linear order, $\mathcal{I}(\mathbb{D})$ is the set of all (strict) intervals over $\mathbb{D}$, and $V$ is a valuation function $V : \mathcal{AP} ↦ 2^{\mathcal{I}(\mathbb{D})}$, which assigns to every atomic proposition $p ∈ \mathcal{AP}$ the set of intervals $V(p)$ on which $p$ holds. The truth of a formula on a given interval $[a, b]$ in an interval model $M$ is defined as standard for the Boolean operators and propositions. The semantics of the modalities are defined as follows:

- $M, [a, b] \models \langle X \rangle \psi$ iff there exists an interval $[c, d]$ such that $[a, b]R_X[c, d]$ and $M, [c, d] \models \psi$, for each modality $\langle X \rangle$.

For every $p ∈ \mathcal{AP}$ and $[a, b] ∈ \mathcal{I}(\mathbb{D})$, we say that $[a, b]$ is a $p$-interval if $M, [a, b] \models p$. By $M, [a, b] \not\models \psi$, we mean that it is not the case that $M, [a, b] \models \psi$. Formulae of HS can be interpreted in several interesting classes of interval models over

---

**Fig. 1.** Allen’s interval relations and the corresponding HS modalities.

### 2 Preliminaries

Let $\mathbb{D} = \langle D, < \rangle$ be a linear order. An interval over $\mathbb{D}$ is an ordered pair $[a, b]$, where $a, b ∈ D$ and $a ≤ b$. An interval is called a point (resp., strict) interval if $a = b$ (resp., $a < b$). In this paper, we restrict ourselves to strict intervals. If we exclude equality, there are 12 diferent relations between two strict intervals in a linear order, often called Allen’s relations [2]: the six relations $R_A, R_L, R_B, R_E, R_D,$ and $R_O$ depicted in Figure 1 and the inverse ones, that is, $R_X = (R_X)^{-1}$, for each $X ∈ \{A, L, B, E, D, O\}$. We treat interval structures as Kripke structures and Allen’s relations as accessibility relations over them, thus associating a modality $\langle X \rangle$ with each Allen’s relation $R_X$. For each $X ∈ \{A, L, B, E, D, O\}$, the transpose of modality $\langle X \rangle$ is modality $\langle X \rangle^T$, corresponding to the inverse relation $R_X^T$ of $R_X$.

HS is a multi-modal logic with formulae built from a finite, non-empty set $\mathcal{AP}$ of atomic propositions, the propositional connectives $\lor$ and $\land$, and a modality for each Allen’s relation [7]. With every subset $\{R_{X_1}, \ldots, R_{X_k}\}$ of these relations, we associate the fragment $X_1X_2\ldots X_k$ of HS, whose formulae are defined by the grammar: $\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \langle X_1 \rangle \varphi \mid \ldots \mid \langle X_k \rangle \varphi$, where $p ∈ \mathcal{AP}$. The other propositional connectives and constants (e.g., $\land, \neg, \lor$) can be derived in the standard way, as well as the dual modalities (e.g., $[A] \varphi ≡ \neg[A] \neg \varphi$).

For a fragment $\mathcal{F} = X_1X_2\ldots X_k$ and a modality $\langle X \rangle$, we write $\langle X \rangle \in \mathcal{F}$ if $X ∈ \{X_1, \ldots, X_k\}$. Given two fragments $\mathcal{F}_1$ and $\mathcal{F}_2$, we write $\mathcal{F}_1 ⊆ \mathcal{F}_2$ if $\langle X \rangle ∈ \mathcal{F}_1$ implies $\langle X \rangle ∈ \mathcal{F}_2$, for every modality $\langle X \rangle$. Finally, for a fragment $\mathcal{F} = X_1X_2\ldots X_k$ and a formula $\varphi$, we write $\varphi \in \mathcal{F}$, or, equivalently, we say that $\varphi$ is an $\mathcal{F}$-formula, meaning that $\varphi$ belongs to the language of $\mathcal{F}$.

The (strict) semantics of HS is given in terms of interval models $M = \langle \mathcal{I}(\mathbb{D}), V \rangle$, where $\mathbb{D}$ is a linear order, $\mathcal{I}(\mathbb{D})$ is the set of all (strict) intervals over $\mathbb{D}$, and $V$ is a valuation function $V : \mathcal{AP} ↦ 2^{\mathcal{I}(\mathbb{D})}$, which assigns to every atomic proposition $p ∈ \mathcal{AP}$ the set of intervals $V(p)$ on which $p$ holds. The truth of a formula on a given interval $[a, b]$ in an interval model $M$ is defined as standard for the Boolean operators and propositions. The semantics of the modalities are defined as follows:

- $M, [a, b] \models \langle X \rangle \psi$ iff there exists an interval $[c, d]$ such that $[a, b]R_X[c, d]$ and $M, [c, d] \models \psi$, for each modality $\langle X \rangle$.

For every $p ∈ \mathcal{AP}$ and $[a, b] ∈ \mathcal{I}(\mathbb{D})$, we say that $[a, b]$ is a $p$-interval if $M, [a, b] \models p$. By $M, [a, b] \not\models \psi$, we mean that it is not the case that $M, [a, b] \models \psi$. Formulae of HS can be interpreted in several interesting classes of interval models over
linear orders (in short, classes of linear orders) such as the classes of all, dense, and discrete linear orders.

The following definition formalizes the notion of definability of modalities in terms of others.

**Definition 1 (Inter-definability).** A modality \( \langle X \rangle \) of HS is definable in an HS fragment \( F \) relative to a class \( C \) of linear orders, denoted \( \langle X \rangle \in_C F \), if \( \langle X \rangle p \equiv_C \psi \) for some \( F \)-formula \( \psi \) over the atomic proposition \( p \), for some \( p \in AP \). In such a case, the equivalence \( \langle X \rangle p \equiv_C \psi \) is called an inter-definability equation (or simply inter-definability) for \( \langle X \rangle \) in \( F \) relative to \( C \). We write \( \langle X \rangle \in_C F \) if \( \langle X \rangle \) is not definable in \( F \) over \( C \).

Notice that smaller classes of linear orders inherit the inter-definabilities holding for larger classes of linear orders. Formally, if \( C_1 \) and \( C_2 \) are classes of linear orders such that \( C_1 \subset C_2 \), then all inter-definabilities holding for \( C_2 \) are also valid for \( C_1 \). However, more inter-definabilities can possibly hold for \( C_1 \). On the other hand, undefinability results for \( C_1 \) hold also for \( C_2 \). In the rest of the paper, we will omit the class of linear orders when it is clear from the context (e.g., we will simply say \( \langle X \rangle p \equiv \psi \) and \( \langle X \rangle \in F \) instead of \( \langle X \rangle p \equiv_C \psi \) and \( \langle X \rangle \in_C F \), respectively).

It is known from [7] that, in the strict semantics, all HS modalities are definable in the fragment containing modalities \( \langle A \rangle \), \( \langle B \rangle \), and \( \langle E \rangle \), and their transposes \( \langle \overline{A} \rangle \), \( \langle \overline{B} \rangle \), and \( \langle \overline{E} \rangle \). In this paper, we compare and classify the expressiveness of all HS fragments relative to the class of all dense linear orders. Formally, let \( F_1 \) and \( F_2 \) be any pair of such fragments. We say that \( F_2 \) is at least as expressive as \( F_1 \) (denoted \( F_1 \preceq F_2 \)) if each modality \( \langle X \rangle \in F_1 \) is definable in \( F_2 \). The notions of being strictly less expressive (\( F_1 \prec F_2 \)), equally expressive (\( F_1 \equiv F_2 \)), and expressively incomparable (\( F_1 \not\equiv F_2 \)) are defined accordingly. Now, it is possible to define the notion of optimal inter-definability, as follows.

**Definition 2 (Optimal inter-definability).** A definability \( \langle X \rangle \in F \) is said optimal if \( \langle X \rangle \not\in F' \) for any fragment \( F' \) such that \( F' \prec F \).

In order to show non-definability of a given modality in an HS fragment, we use a standard technique in modal logic, based on the notion of bisimulation and the invariance of modal formulae with respect to bisimulations (see, e.g., [4,8]). The important property of bisimulations used here is that any \( F \)-bisimulation preserves the truth of all formulae in \( F \), that is, if \( ([a,b],[a',b']) \in Z \) and \( Z \) is an \( F \)-bisimulation, then \( [a,b] \) and \( [a',b'] \) satisfy exactly the same formulae in \( F \). Thus, in order to prove that a modality \( \langle X \rangle \) is not definable in \( F \), it suffices to construct a pair of interval models \( M = \langle I(D), V \rangle \) and \( M' = \langle I(D'), V' \rangle \), and an \( F \)-bisimulation \( Z \) between them, relating a pair of intervals \( [a,b] \in I(D) \) and \( [a',b'] \in I(D') \), such that \( M,[a,b] \vDash \langle X \rangle p \), while \( M',[a',b'] \not\vDash \langle X \rangle p \). In this case, we say that \( Z \) breaks \( \langle X \rangle \).

The problem. As we already pointed out, every subset of the set of the 12 modalities corresponding to Allen’s relations gives rise to a logic, namely, a fragment of HS. There are \( 2^{12} \) (the cardinality of the powerset of the set of
that the union of all equations for interdefinabilities for the operators linear orders. In what follows, we first prove that Table 1 depicts a set of optimal interdefinabilities and we provide bisimulations based on discrete structures. As a consequence, the above results for linear orders are based on dense structures, apart from those for modalities based on discrete structures. In what follows, we want to be able to decide how they relate to each other with respect to expressiveness (that is, whether is strictly less expressive than , is strictly more expressive than , and are expressively equivalent, or are incomparable).

In order to do so, all we need to do is to provide the complete set of optimal interdefinabilities between HS modalities. Indeed, provided with such a set, it is immediate to decide which relation exists between any two given fragments with respect to their expressive power.

The class of all linear orders. The problem we address in this paper has been solved for the class of all linear orders in [5], where the complete set of optimal interdefinabilities in Table 1a has been identified.

All the bisimulations used in [5] to solve the problem for the class of all linear orders are based on dense structures, apart from those for and , which are based on discrete structures. As a consequence, the above results for all modalities but immediately extend to all classes of dense linear orders. In what follows, we identify a new set of optimal interdefinabilities holding for and over classes of dense linear orders, and we prove it to be complete (for the modalities and )

3 The class of all dense linear orders

From now on, we focus our attention on the class of all dense linear orders, and we provide bisimulations based on . However, it is possible to extend our results to sub-classes of the class of all dense linear orders (that might not include ), by providing bisimulations based on different (suitable) dense linear orders. In what follows, we first prove that Table 1b depicts a set of interdefinabilities for the operators and (Lemma 1). Then, we show that the union of all equations for and shown in Table 1a and Table 1b

<table>
<thead>
<tr>
<th>(L)p ≡ (A)(A)p</th>
<th>(L) ≡ A</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L)p ≡ (A)(A)p</td>
<td>(L) ≡ A</td>
</tr>
<tr>
<td>(A)p ≡ (E)(E)p</td>
<td>(A) ≡ E</td>
</tr>
<tr>
<td>(D)p ≡ (E)(E)p</td>
<td>(D) ≡ E</td>
</tr>
<tr>
<td>(L)p ≡ (E)(E)p</td>
<td>(L) ≡ E</td>
</tr>
<tr>
<td>(E)p ≡ (E)(E)p</td>
<td>(E) ≡ E</td>
</tr>
</tbody>
</table>

(a) The complete set of optimal interdefinabilities for the class of all linear orders.

<table>
<thead>
<tr>
<th>(L)p ≡ (O)(O)(O)p</th>
<th>(L) ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L)p ≡ (O)(O)(O)p</td>
<td>(L) ≡ B</td>
</tr>
<tr>
<td>(L)p ≡ (O)(O)(O)p</td>
<td>(L) ≡ B</td>
</tr>
<tr>
<td>(L)p ≡ (O)(O)(O)p</td>
<td>(L) ≡ B</td>
</tr>
<tr>
<td>(L)p ≡ (O)(O)(O)p</td>
<td>(L) ≡ B</td>
</tr>
</tbody>
</table>

(b) A set of inter-definabilities for and over the class of all dense linear orders.
constitutes the complete set of optimal inter-definabilities for those operators (Theorem 1).

**Lemma 1.** Table 1b depicts a set of inter-definabilities for the operators \( \langle L \rangle \) and \( \langle L' \rangle \).

The rest of the paper is devoted to establishing our main result, that is, to prove that Table 1a and Table 1b depict a complete set of optimal inter-definabilities for the operator \( \langle L \rangle \). This means that we cannot define \( \langle L \rangle \) by means of any other optimal equation. It is immediate to verify, by symmetry, that the same result holds for the operator \( \langle L' \rangle \).

As a first step, we need to identify all maximal HS fragments not containing, as definable (according to the inter-definabilities of Table 1a and Table 1b), the operator \( \langle L \rangle \). Given the large number of inter-definabilities, it is not immediate to detect all such fragments. For this purpose, we used the algorithm presented in [1]. The algorithm, run on the list of inter-definabilities in Table 1a and Table 1b, and on modality \( \langle L \rangle \) as input parameters, returned the three maximal fragments \( \text{OBEDO}, \text{BEDALEDO}, \text{BALBEDO} \). In the light of the inter-definabilities in Table 1a, we can replace these three fragments with equivalent fragments featuring the smallest set of modalities, namely, \( \text{OBEDO}, \text{BEAED}, \text{BAED} \), respectively. Now, in order to establish the optimality of the set of inter-definabilities, for each such fragment \( F \), we provide an \( F \)-bisimulation that breaks \( \langle L \rangle \). (Due to lack of space, we only provide the proof for the first of the three bisimulations presented in what follows. For the other proofs, we refer the interested reader to [1].) In what follows, thanks to the next proposition, in our proofs we can safely assume that for each interval \( [a, b] \) and Allen’s relation \( R_X \), there exists an interval \([c, d]\) such that \([a, b]R_X[c, d]\).

**Proposition 1.** Let \( \mathbb{D} \) be a dense linear order without least and greatest elements, and let \([a, b] \in \mathbb{D} \). Then, there exists an interval \([c, d] \in \mathbb{D} \) such that \([a, b]R_X[c, d]\), for each \( X \in \{ A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O} \} \).

An \( \text{OBEDO} \)-bisimulation that breaks \( \langle L \rangle \). Consider the two interval models \( M \) and \( M' \), defined as \( M = M' = (\mathbb{I}(\mathbb{R}), V) \), where \( V(p) = \{ [-a, a] \mid a \in \mathbb{R} \} \) (observe that no interval \([c, d]\), with \( c \geq 0 \), satisfies \( p \)). Moreover, let \( Z = \{ ([a, b], [a', b']) \mid -a \sim b \quad \text{and} \quad -a' \sim b' \text{ for some } \sim \in \{ <, =, > \} \} \) (see Fig. 2a).

**Lemma 2.** \( Z \) is a \( \text{OBEDO} \)-bisimulation.

**Proof.** Local condition. Consider a pair \(([a, b], [a', b'])\) of \( Z \)-related intervals. The following chain of double implications hold: \( M, [a, b] \models p \iff -a = b \) iff (by the definition of \( Z \)) \(-a' = b' \) iff \( M, [a', b'] \models p \).

**Forward condition.** Consider the three intervals \([a, b], [a', b']\), and \([c, d]\) such that \([a, b]Z[a', b']\) and \([a, b]R_X[c, d]\) for some \( X \in \{ O, \overline{B}, \overline{E}, \overline{O} \} \). We need to exhibit an interval \([c', d']\) such that \([a', b']R_X[c', d']\) and \([c, d]Z[c', d']\). We distinguish three cases.

- If \(-a > b \quad \text{and} \quad -a' > b' \), then, as a preliminary step, we show that the following facts hold: (i) \( a < 0 \) and \( a' < 0 \); (ii) \( |a| > |b| \) and \( |a'| > |b'| \).

---

Proceedings of ICTCS 2013
We only show the proofs for \( a < 0 \) and \( |a| > |b| \) and we omit the ones for \( a' < 0 \) and \( |a'| > |b| \), which are analogous. As for the former claim above, it is enough to observe that, if \( a \geq 0 \), then \( a \geq 0 \Rightarrow -a > b \), which implies \( b < a \), leading to a contradiction with the fact that \([a, b]\) is an interval (thus \( a < b \)). Notice that, as an immediate consequence, we have that \( |a| = -a \) holds. As for the latter claim above, firstly we suppose, by contradiction, that \( |a| = |b| \) holds. Then, \(-a = |a| = |b| \) holds and this implies either \( b = -a \), contradicting the hypothesis that \(-a > b \), or \( b = a \), contradicting the fact that \([a, b]\) is an interval. Secondly, we suppose, again by contradiction, that \( |a| < |b| \) holds. Then, by the former claim, we have that \( 0 < -a = |a| < |b| \) holds, which implies \( b \neq 0 \). Now, we show that both \( b < 0 \) and \( b > 0 \) lead to a contradiction. If \( b < 0 \), then \( |b| = -b \), and thus it holds \(-a < -b \), which amounts to \( a > b \), contradicting the fact that \([a, b]\) is an interval. If \( b > 0 \), then \( |b| = b \), and thus it holds \(-a < b \), which contradicts the hypothesis that \(-a > b \). This proves the two claims above. Now, we distinguish the following sub-cases.

- If \( X = O \), then \([c, d]\) is such that \( a < c < b < d \). We distinguish the following cases.
  * If \(-c > d \), then take some \( c' \) such that \( a' = c' < -|b'| = 0 \) (notice also that \( c' < -|b'| \leq b' \) trivially holds), and \( d' \) such that \( b' < d' < |c'| = c' \) (the existence of such points \( c', d' \) is guaranteed by the density of \( \mathbb{R} \)). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).
  * If \(-c = d \), then take some \( c' \) such that \( a' < c' < -|b'| < 0 \), and \( d' = c' \) (the existence of such a point \( c' \) is guaranteed by the density of \( \mathbb{R} \)). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).
  * If \(-c < d \), then take \( c' \) such that \( a' < c' < -|b'| < 0 \), and any \( d' > c' \) (the existence of such a point \( c' \) is guaranteed by the density of \( \mathbb{R} \)). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).

- If \( X = \mathcal{B} \), then \([c, d]\) is such that \( a = c < b < d \). We distinguish the cases below.
  * If \(-c > d \), then take \( c' = a' \) and \( d' \) such that \( b' < d' < a' = -c' \) (the existence of such a point \( d' \) is guaranteed by the density of \( \mathbb{R} \)). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).
  * If \(-c = d \), then take \( c' = a' \) and \( d' = c' \) (the existence of such points \( c', d' \) is guaranteed by the density of \( \mathbb{R} \)). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).
  * If \(-c < d \), then take \( c' = d' \) and any \( d' > c' = -a' > b' \). The interval \([c', d']\) is such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).

- If \( X = \mathcal{E} \), then \([c, d]\) is such that \( c < a < b = d \). Notice that \( |a| = -c > -a = |a| \) holds, because \( c < a < 0 \). Thus \(-c > -a > b = d \) also holds. Then, take \( d' = b' \) and any \( c' < a' \). We have that \(-c > -a' > b' = d' \). The interval \([c', d']\) is therefore such that \([a', b']R_{\mathbb{R}}[c', d']\) and \([c, d]Z[c', d']\).

- If \( X = O \), then \([c, d]\) is such that \( c < a < d < b \). Notice that \( |a| = -c > -a = |a| \) holds, because \( c < a < 0 \). Thus \(-c > -a > b > d \) also
holds. Then, take some $d'$ such that $a' < d' < b'$ and any $c' < a'$ (the existence of such a point $d'$ is guaranteed by the density of $\mathbb{R}$). Thus, it holds $-c' > -a' > b' > d'$. The interval $[c', d']$ is therefore such that $[a', b']R_{\bar{Z}}[c', d']$ and $[c, d]Z[c', d']$.

- If $-a = b$ and $-a' = b'$, then we have that $a < 0$ (resp., $a' < 0$) and $b > 0$ (resp., $b' > 0$). Indeed, if $a \geq 0$ held, then $b = -a \leq 0 \leq a$ would also hold, contradicting the fact that $[a, b]$ is an interval (and thus $b > a$). From $a < 0$ and $-a = b$, it immediately follows that $b > 0$. The facts that $a' < 0$ and $b' > 0$ can be shown analogously. Notice also that, from $-a = b$ and $-a' = b'$, it follows that $|a| = |b|$ and $|a'| = |b'|$. Now, we distinguish the following sub-cases.

1. If $X = O$, then $[c, d]$ is such that $a < c < b < d$. Notice that $-c \leq |c| < |a| = |b| < d$ holds. Then, take $c' = 0$ and any $d' > b'$. We have that $-c' < d'$. The interval $[c', d']$ is such that $[a', b']R_{O}[c', d']$ and $[c, d]Z[c', d']$.

2. If $X = \bar{B}$, then $[c, d]$ is such that $a < c < b < d$. Notice that $-c \leq |c| = |a| = |b| < d$ holds. Then, take $c' = a'$ and any $d' > b'$. We have that $-c' = -a' = b' < d'$. The interval $[c', d']$ is such that $[a', b']R_{\bar{B}}[c', d']$ and $[c, d]Z[c', d']$.

3. If $X = \bar{E}$, then $[c, d]$ is such that $c < a < b = d$. Notice that $|c| = -c > -a = |a|$ holds, because $c < a < 0$. Thus $-c > -a = b = d$. Then, take $d' = b'$ and any $c' < a'$. We have that $-c' > -a' = b' = d'$. The interval $[c', d']$ is such that $[a', b']R_{\bar{E}}[c', d']$ and $[c, d]Z[c', d']$.

4. If $X = \bar{O}$, then $[c, d]$ is such that $c < a < d < b$. Notice that $|c| = -c > -a = |a|$ holds, because $c < a < 0$. Thus $-c > -a = b > d$ also holds. Then, take $d' = b'$ and any $c' < a'$. We have that $-c' > d'$. The interval $[c', d']$ is such that $[a', b']R_{\bar{O}}[c', d']$ and $[c, d]Z[c', d']$.

- If $-a < b$ and $-a' < b'$, then the proof proceeds symmetrically to the case when $-a > b$ and $-a' > b'$. More precisely, the argument used there for modalities $\langle \bar{O} \rangle$ and $\langle \bar{E} \rangle$ applies now to modalities $\langle O \rangle$ and $\langle \bar{B} \rangle$, and vice versa. The details are omitted.

**Backward condition.** Since the relation $Z$ is symmetric, the forward condition implies the backward condition, as follows. Consider a pair $([a, b], [a', b'], Z)$ of $Z$-related intervals and an interval $[c', d']$ such that $[a', b']R_{X}[c', d']$, for some $X \in \{ O, \bar{B}, \bar{E}, \bar{O} \}$. We need to find an interval $[c, d]$ such that $[a, b]R_{X}[c, d]$ and $[c, d]Z[c', d']$. By symmetry, $([a', b'], [a, b]) \in Z$, as well. By the forward condition, we know that for every interval $[c', d']$ such that $[a', b']R_{X}[c', d']$, for some $X \in \{ O, \bar{B}, \bar{E}, \bar{O} \}$, there exists an interval $[c, d]$ such that $[a, b]R_{X}[c, d]$ and $[c', d']Z[c, d]$. By symmetry $[c, d]Z[c', d']$ also holds, hence the backward condition is fulfilled, too.

**Corollary 1.** The modality $\langle L \rangle$ is not definable in the fragment $\texttt{BEAED}$ (and in any of its sub-fragments) over the class of all dense linear orders.

**A BEAED-bisimulation that breaks $\langle L \rangle$.** In order to define a BEAED-bisimulation that breaks $\langle L \rangle$, we will make use of the function $f : \mathbb{R} \rightarrow \{ x \in \mathbb{R} \mid x < 1 \}$, defined as: $f(x) = x - 1$ if $x \leq 1$ and $f(x) = 1 - \frac{1}{2}$ otherwise.
Lemma 3. \( f \) is a monotonically increasing bijection from \( \mathbb{R} \) to \( \{x \in \mathbb{R} \mid x < 1\} \) such that \( f(x) < x \) for every \( x \in \mathbb{R} \).

The bisimulation that breaks \( \langle L \rangle \) is defined as follows. We consider two interval models \( M \) and \( M' \), defined as \( M = \langle \mathbb{I}(\mathbb{R}), V \rangle \) and \( M' = \langle \mathbb{I}(\mathbb{R}), V' \rangle \), respectively, where \( V(p) = \{[a, b] \mid a = f(b)\} \) and let \( Z = \{([a, b], [a', b']) \mid a \sim f(b), a' \sim f(b') \text{ for some } \sim \in \{<, =, >\}\} \) (see Fig. 2b).

Lemma 4. \( Z \) is a \( \text{BEAED} \)-bisimulation.

Corollary 2. The modality \( \langle L \rangle \) is not definable in the fragment \( \text{BEAED} \) (and in any of its sub-fragments) over the class of all dense linear orders.

A \( \text{BABE} \)-bisimulation that breaks \( \langle L \rangle \). Consider the two interval models \( M \) and \( M' \), defined as \( M = \langle \mathbb{I}(\mathbb{R}), V \rangle \) and \( M' = \langle \mathbb{I}(\mathbb{R}), V' \rangle \), respectively, where \( V(p) = \{[a, b] \mid a, b \in \mathbb{Q} \text{ or } a, b \in \mathbb{R} \setminus \mathbb{Q}\} \) and \( V'(p) = \{[a', b'] \mid a' \leq 0 \text{ and } (a', b' \in \mathbb{Q} \text{ or } a', b' \in \mathbb{R} \setminus \mathbb{Q})\} \). Moreover, let \( Z = \{([a, b], [a', b']) \mid a' \leq -1 \text{ and } M, [a, b] \models p \text{ iff } M', [a', b'] \models p\} \).

Lemma 5. \( Z \) is a \( \text{BABE} \)-bisimulation.

Corollary 3. The modality \( \langle L \rangle \) is not definable in the fragment \( \text{BABE} \) (and in any of its sub-fragments) over classes of dense linear orders.

Theorem 1. Table 1a and Table 1b depict a complete set of optimal inter-definabilities for the modality \( \langle L \rangle \).

References


Proceedings of ICTCS 2013
Two Gray Codes for $q$-ary $k$-generalized Fibonacci Strings

Antonio Bernini$^1$, Stefano Bilotta$^1$, Renzo Pinzani$^1$, and Vincent Vajnovszki$^2$

$^1$ Dipartimento di Matematica e Informatica “Ulisse Dini”, Università degli Studi di Firenze, Viale G.B. Morgagni 65, 50134 Firenze, Italy
$^2$ L2EI, Université de Bourgogne, BP 47 870, 21078 Dijon Cedex, France

1 Introduction

The famous $k$-generalized Fibonacci sequence $\{F_n^{(k)}\}_{n \geq 0}$ is defined by

$$F_n^{(k)} = F_{n-1}^{(k)} + F_{n-2}^{(k)} + \ldots + F_{n-k}^{(k)} \quad \text{for } n \geq k,$$

with initial conditions (as in [4])

$$F_0^{(k)} = 0, F_1^{(k)} = 0, \ldots, F_{k-2}^{(k)} = 0, F_{k-1}^{(k)} = 1.$$

A combinatorial interpretation of these numbers is the enumeration of binary strings avoiding $k$ consecutive 1’s [4], called $k$-generalized Fibonacci strings.

If an alphabet $A$ with cardinality $q > 1$ is used, then the enumeration of the strings of length $n$ avoiding a pattern constituted by $k$ consecutive occurrences of a symbol of $A$ is given by

$$F_n^{(k),q} = (q - 1) \left( F_{n-1}^{(k),q} + F_{n-2}^{(k),q} + \ldots + F_{n-k}^{(k),q} \right) \quad \text{for } n \geq k,$$

with

$$F_0^{(k),q} = 1, F_1^{(k),q} = q, \ldots, F_{k-2}^{(k),q} = q^{k-2}, F_{k-1}^{(k),q} = q^{k-1}.$$

This sequence is a particular case of the weighted $k$-generalized Fibonacci sequence (studied and used in [5] and [6]) where all the weights are equal to $q - 1$. Similarly to the binary case, we denote $q$-ary $k$-generalized Fibonacci strings the strings on an alphabet of cardinality $q$ and avoiding $k$ consecutive occurrences of a symbol of the alphabet $A$.

In this work we provide two different Gray codes for the $q$-ary $k$-generalized Fibonacci strings. The first one is based on a direct recursive construction starting from all the strings of length less than $k$. The second one works on the elements of an already existing Gray code [7] defined for binary strings avoiding $k$ consecutive 0’s.

2 Basic Notation and Definitions

If $L$ is a set of strings over an alphabet $A$, then $L$ denotes a list where the strings of $L$ are listed following a certain criterion. If the Hamming distance [3] between two successive elements of $L$ is bounded by a constant, than $L$ is called Gray code list.

The notation we are going to use is defined below:
- if $\alpha$ and $\beta$ are two successive elements of $L$, then $d_H(\alpha, \beta)$ denotes their Hamming distance;
- $\text{first}(L)$ and $\text{last}(L)$ are the first and the last element of $L$, respectively;
- $L'$ denotes the list obtained reading from right to left the elements of $L$;
- if $u$ is an element of $A^*$, then $u \cdot L$ is a new list where each element has the form $u \omega$, where $\omega$ is any element of $L$;
- if $L$ and $L'$ are two lists, then $L \circ L'$ is the concatenation of the two lists, obtained by appending the elements of $L'$ after the elements of $L$.

The well known Binary Reflected Gray Code (BRGC) [2] can be generalized to an alphabet of cardinality greater than 2 [1, 8]. If $A = \{a_0, a_1, \ldots, a_{q-1}\}$, then the list $G^n_q$ of the words in $A^n$ is given by:

$$
G^n_q = \begin{cases} 
\emptyset & \text{if } n = 0 , \\
 a_0 \cdot G^n_{q-1} \circ a_1 \cdot G^n_{q-1} \circ \ldots \circ a_{q-1} \cdot G^n_{q-1} & \text{if } n > 0 , 
\end{cases}
$$

where $G^n_{q-1}$ is $G^n_{q-1}$ if $q$ is even or $G^n_{q-1}$ if $q$ is odd. It is proved [1] that $G^n_q$ is a Gray code list with Hamming distance 1.

**Proposition 1.** If $q$ is odd, then last($G^n_q$) = $a_{q-1}^n$ and first($G^n_q$) = $a_0^n$. In particular

$$
\text{last}(G^n_q) = \text{last}(G^n_{q-1})a_{q-1} = a_{q-1}\text{last}(G^n_{q-1})
$$

and

$$
\text{first}(G^n_q) = \text{first}(G^n_{q-1})a_0 = a_0\text{first}(G^n_{q-1}) .
$$

**Proposition 2.** If $q$ is even, then last($G^n_q$) = $a_{q-1}a_0^{n-1}$ and first($G^n_q$) = $a_0^n$. In particular

$$
\text{last}(G^n_q) = a_{q-1}a_0^{n-1} = \text{last}(G^n_{q-1})a_0
$$

and

$$
\text{first}(G^n_q) = a_0^n = \text{first}(G^n_{q-1})a_0 = a_0\text{first}(G^n_{q-1}) .
$$

### 3 Direct Recursive Construction

We propose a first construction for a *prefix partitioned* Gray code for the $q$-ary $k$-generalized Fibonacci strings of length $n$, where the Hamming distance between two adjacent strings is 1. From now on, we simply refer to “Gray code” as a list where two successive elements have Hamming distance equal to 1. Moreover, with abuse of notation and for the sake of simplicity, we can suppose $A = \{0, 1, \ldots, q-1\}$. For our topics we are going to consider the avoidance of the pattern $0^k$, but the reader can easily check that all our arguments are still right for $i^k$, with $i \in A, i \neq 0$. The definition we propose is a recursive definition and depends on the parity of $q$. 

Proceedings of ICTCS 2013
Proposition 3. Let

\[ I^{(k)}_{n,q} = \begin{cases} \overline{G}_n^q & \text{if } 0 \leq n < k, \\ \Theta_{n,1} \circ \Theta_{n,2} \circ \ldots \circ \Theta_{n,k} & \text{if } n \geq k. \end{cases} \]

with

\[ \Theta_{n,j} = 0^{j-1}(q-1) \cdot \overline{I}^{(k)}_{n-j,q} \circ \ldots \circ 0^{j-1} \cdot \overline{I}^{(k)}_{n-j,q} \circ \overline{I}^{(k)}_{n-j,q}, \quad \forall j = 1, \ldots, k, \]

where \( I^{(k)}_{n-j,q} \) is \( I^{(k)}_{n-j,q} \) if \( q \) is even and \( I^{(k)}_{n-j,q} \) if \( q \) is odd. Then, \( I^{(k)}_{n,q} \) is a Gray code for the class of the \( k \)-generalized Fibonacci strings on the \( q \)-ary alphabet \( A \).

We propose the construction of the Gray codes for \( n \leq 4 \), \( k = 3 \), \( A = \{0,1,2\} \). We have:

\[ I^{(3)}_{0,3} = \emptyset; \]
\[ I^{(3)}_{1,3} = (2, 1, 0); \]
\[ I^{(3)}_{2,3} = (22, 21, 20, 10, 11, 12, 02, 01, 00); \]
\[ I^{(3)}_{3,3} = (222, \ldots, 200, 100, \ldots, 122, 022, 021, 020, 010, 011, 012, 002, 001); \]
\[ I^{(3)}_{4,3} = (2222, \ldots, 2001, 1001, \ldots, 1222, 0222, \ldots, 0200, 0100, \ldots, 0122, \]
\[ \quad 0022, 0021, 0020, 0010, 0011, 0012). \]

The reader can easily check that \( I^{(3)}_{i,3} \), for each \( i \) with \( 1 \leq i \leq 4 \), is a Gray code.

4 Replacing Bits

A different approach is now proposed. It starts from the definition of the Gray code for \( k \)-generalized Fibonacci binary strings [7], then we extend the binary alphabet into the \( q \)-ary alphabet \( A = \{0,1,\ldots,q-1\} \), producing a technique replacing bits, leading to a \textit{trace partitioned} Gray code.

Before going along in our discussion, we recall a tool for manipulating the characters of a string. If \( \beta \) is a binary string of length \( n \) such that \( |\beta|_1 = t \) (the number of 1’s in \( \beta \)), we say that \( \varepsilon(\beta) \) is a list of \( |G^{(q)}_n| \) strings, where the \( i \)-th string is obtained by replacing the \( t \) 1’s of \( \beta \) by the \( t \) symbols (read from left
to right) of the \(i\)-th string in \(G^q\). For example, if \(\beta = 01011\) (the trace) and \(\Sigma = \{1, 2\}\), then

\[
G^2 = (111, 112, 122, 121, 221, 222, 212, 211)
\]

and

\[
\varepsilon(\beta) = (01011, 01012, 01022, 01021, 02021, 02022, 02012, 02011).
\]

Note that \(\text{first}(\varepsilon(\beta)) = \beta\) and all the elements of \(\varepsilon(\beta)\) have the same trace.

We observe that \(\varepsilon(\beta)\) is essentially the list \(G^q\) where each element has been expanded inserting some 0’s, each time in the same positions. Since \(G^q\) is a Gray code and the insertions of the 0’s does not change the Hamming distance between two successive element of \(\varepsilon(\beta)\) (which is 1), we have the following:

**Proposition 4.** The list \(\varepsilon(\beta)\) is a Gray code.

The Gray code we are going to consider as the starting point of our argument is the one defined in [7], where the author deals with binary strings avoiding \(k\) consecutive 1’s. Since we are interested in the avoidance of \(k\) consecutive 0’s, we recall, for the sake of clearness, the definition in [7] adapted according to our needs which cause some slight differences with respect to the original definition in [7].

The Gray code list \(F_n^{(k)}\) for the \(n\)-length bitstrings avoiding \(k\) consecutive 0’s is defined by

\[
F_n^{(k)} = \left\{ \begin{array}{ll}
C_n & \text{if } 0 \leq n < k, \\
1 \cdot F_{n-1}^{(k)} \circ 01 \cdot F_{n-2}^{(k)} \circ 001 \cdot F_{n-3}^{(k)} \circ \cdots \circ 0^{k-1} \cdot F_{n-k}^{(k)} & \text{if } n \geq k,
\end{array} \right.
\]

where

\[
C_n = \left\{ \begin{array}{ll}
\lambda & \text{if } n = 0 \\
1 \cdot C_{n-1} \circ 0 \cdot C_{n-1} & \text{if } n \geq 1.
\end{array} \right.
\]

It is proved [7] that \(|F_n^{(k)}| = f_{n,k}\), where \(f_{n,k}\) is the \(n\)-th number of the \(k\)-Fibonacci sequence.

Let \(A = \{0, 1, \ldots, q - 1\}\) and let \(F_{n,q}^{(k)}\) be the subset of words of \(A^n\) avoiding \(k\) consecutive 0’s. The aim is the construction of a Gray code for \(F_{n,q}^{(k)}\). Our definition moves from the idea of considering the expansion \(\varepsilon(\alpha_i)\) (or its reverse \(\overline{\varepsilon(\alpha_i)}\)), according to the alphabet \(A' = \{1, 2, \ldots, q - 1\}\), of each element of \(F_{n,q}^{(k)} = (\alpha_1, \alpha_2, \ldots, \alpha_{f_{n,k}})\), and then concatenating them opportunely, in dependence of the parity of \(q\).

---

*Proceedings of ICTCS 2013*
4.1 The Case of \( q \) Even

In this case the construction of a Gray Code list for \( F_{n,q}^{(k)} \) is straightforward: just consider the expansions of the elements \( \alpha_i \), for \( i = 1, 2, \ldots, f_{n,k} \), according to the alphabet \( A' \), and concatenate them, taking \( \varepsilon(\alpha_i) \) and \( \varepsilon(\alpha_{i+1}) \) alternatively, in order to obtain a list \( F_{n,q}^{(k)} \) we prove to be a Gray Code list. The following proposition, indeed, holds:

**Proposition 5.** If \( q \) is even, then the list

\[
F_{n,q}^{(k)} = \varepsilon(\alpha_1) \circ \varepsilon(\alpha_2) \circ \ldots \circ \varepsilon'(\alpha_{f_{n,k}}),
\]

where \( \varepsilon'(\alpha_{f_{n,k}}) = \varepsilon(\alpha_{f_{n,k}}) \) if \( f_{n,k} \) is odd and \( \varepsilon'(\alpha_{f_{n,k}}) = \varepsilon(\alpha_{f_{n,k}}) \) if \( f_{n,k} \) is even, is a Gray Code list with Hamming distance 1.

4.2 The Case of \( q \) Odd

The preceding proposition can not be applied in general when \( q \) is odd: the reader can easily check that for \( n = 4, k = 3, \) and \( A = \{0, 1, 2\} \), the concatenation of \( \varepsilon(\alpha_i) \) and \( \varepsilon(\alpha_{i+1}) \), alternatively, does not work. A possible solution could be the following:

- The elements \( \alpha_i \), for \( i = 1, 2, \ldots, f_{n-j,k} \), can be grouped together according to their prefix (1 or 01 or 001 or \ldots or 0\(^{k-1}\)) derived from the definition of \( F_{n}^{(k)} \). Formalizing, let

\[
\alpha_i^{(j)} \in 0^j \cdot F_{n-j}^{(k)}, \text{ with } j = 1, 2, \ldots, k.
\]

- Then, we consider the following expansions \( \Gamma_j \) of each group:

\[
\Gamma_j = \varepsilon(\alpha_{1}^{(j)}) \circ \varepsilon(\alpha_{2}^{(j)}) \circ \varepsilon(\alpha_{3}^{(j)}) \circ \varepsilon(\alpha_{4}^{(j)}) \circ \ldots \circ \varepsilon'(\alpha_{f_{n-j,k}}^{(j)}), \text{ with } j = 1, 2, \ldots, k,
\]

where \( \varepsilon'(\alpha_{f_{n-j,k}}^{(j)}) = \varepsilon(\alpha_{f_{n-j,k}}^{(j)}) \) if \( f_{n-j,k} \) is odd and \( \varepsilon'(\alpha_{f_{n-j,k}}^{(j)}) = \varepsilon(\alpha_{f_{n-j,k}}^{(j)}) \) if \( f_{n-j,k} \) is even.

The following proposition holds:

**Proposition 6.** If \( q \) is odd, then the list

\[
F_{n,q}^{(k)} = \Gamma_1 \circ \Gamma_2 \circ \ldots \circ \Gamma_k
\]

is a Gray Code list with Hamming distance 1.
For the sake of clearness, we propose the right construction of the Gray code for the case $n = 4$, $k = 3$, $A = \{0, 1, 2\}$ and $A' = \{1, 2\}$. We have:

$$\mathcal{F}_4^{(3)} = \left\{ 1001, 1011, 1010, 1110, 1101, 1100, 0100, 0101, 0111, 0110, \overline{0010}, \overline{0011} \right\} ;$$

$$\mathcal{G}_0^2 = \emptyset ;$$
$$\mathcal{G}_1^2 = \{1, 2\} ;$$
$$\mathcal{G}_2^2 = \{11, 12, 22, 21\} ;$$
$$\mathcal{G}_3^2 = \{1111, 1112, 1122, 1211, 1212, 1221, 2211, 2212, 2222, 2221, 2121, 2122, 2112, 2111\} ;$$

$$\Gamma_1 = \left\{ 1001, 1002, 2002, 2001, 2011, \ldots, 1011, 1010, \ldots, 2110, \ldots, 1110, \right\}$$
$$\left\{ 1111, 2111, 2101, \ldots, 1101, 1100, \ldots, 2100 \right\} ;$$

$$\Gamma_2 = \{0100, 0200, 0201, \ldots, 0101, 0111, \ldots, 0211, 0210, \ldots, 0110\} ;$$

$$\Gamma_3 = \{0010, 0020, 0021, \ldots, 0011\} .$$

The reader can easily check that $\mathcal{F}_4^{(3)} = \Gamma_1 \circ \Gamma_2 \circ \Gamma_3$ is a Gray code with Hamming distance 1.

References

Commutative consensual counter languages

Stefano Crespi Reghizzi and Pierluigi San Pietro
DEIB, Politecnico di Milano and CNR-IEIIT
stefano.crespireghizzi@polimi.it pierluigi.sanpietro@polimi.it

Abstract. We show that the family obtained by closure under union and catenation of semilinear commutative languages can be specified by means of parallel matching finite-state computations, within the model of consensually regular languages CREG. To recognize a word, the consensual model uses a regular language (or DFA), the base, on a doubled alphabet; several base language words must match (in a precise sense) to accept a given word. Such recognition device is a multi-counter machine storing linear-sized multisets of DFA states, hence operating in NLOGSPACE. For every commutative language given by its Parikh vectors, the result is a regular set, consensually generating the language. By carefully assigning to different numeric congruence classes the computations that pertain to each component language, we are able to define the catenations and unions of such languages by means of the same consensual device. The result strengthens related studies on the specification of various multi-counter languages by means of consensual regular expressions.

1 Introduction

This work combines the traditional line of research on commutative languages with the recent model of consensual languages, introduced by the authors [1, 2] to represent language processes that interact and re-enforce each other. Consider the alphabet (called internal) obtained by uniting the terminal alphabet with a marked (dotted) copy. Two or more words on the internal alphabet strongly match (metaphorically, they provide consensus to each other) if they coincide when the marks are ignored, and in every position exactly one of the words has an undotted letter. Thus a regular set—the base—over the internal alphabet specifies another language over the terminal alphabet, called a Consensually Regular Language (CREG); a word is accepted if a corresponding set of matching words is in the base. Clearly, the REG (the regular languages) is included in CREG. The family CREG is known to be in NLOGSPACE, to include non-semilinear languages, and to be incomparable with the family of context-free languages. Moreover, the descriptional complexity of a regular language (i.e., the size of its minimal NFA) can be exponentially larger than the size of the minimal DFA for the base language. Also, strictly locally testable bases consensually define all, and only, regular languages.

To recognize a word of length $n$ using the DFA of the base, up to $n$ matching computations run in parallel, hence the instantaneous configuration is a multiset of states, which can be represented by a finite set of counters whose sum is bounded by $n$; thus consensual recognizers are a type of NLOGSPACE multi-counter real-time machine.

A semilinear property SLIP language is the finite union of linear property (LIP) languages (e.g., [4]). We deal with SLIP/LIP commutative languages (COM-SLIP/COM-LIP). COM-SLIP coincide with the commutative closure
of regular languages, and can be recognized by nondeterministic, blind multi-counter machines (see [5] among others). For each letter the machine has a finite set of counters that accumulate the letter count; when the input is finished, the machine nondeterministically checks (by decrementing counters) that the totals satisfy one of the linear equalities defining the language. Clearly, zero checks are performed only at the end, hence the counters are blind. The approach can be extended to the closure of COM-SLIP under union and catenation product, to be denoted by \textit{UP-COM-SLIP}, but the resulting machine structure is uninteresting, because it performs all relevant calculations at the end by means of spontaneous moves that decrement the counters. Our research instead leads to a \textsc{Nlogspace} nondeterministic multi-counter machine operating in real-time, which can be composed with similar machines to recognize any language obtained by unions and products.

The UP-COM-SLIP family, which in general is non-commutative, has received little attention in the past (to the best of our knowledge). To hint to a potential application, we observe that UP-COM-SLIP offers a rather suitable schema for parallel computation systems, such as the “bulk synchronous parallel computer” of Valiant [7]. There, when all threads in a parallel computational phase, which we suggest to model by a commutative language, terminate, the next phase can start; the sequential composition of such phases can be represented by language catenation. Then the composition of alternative subsystems corresponds to language union.

A COM-LIP language is very easy to be consensually defined (examples follow), but not so for the unions and catenations of such languages. This is because the union of two base languages may produce spurious cross-matching, hence more than the union of the corresponding consensual languages. To block cross-matching, we carefully “discipline” the bases so that each base carries a sort of personal “signature” and the signatures of different bases never match. As signatures we use different residues of the same numeric congruence, in such a way that they are invariant under circular shifts.

The results and constructions may be useful when dealing with counter machines, because they offer a more abstract specification method based on regular expressions. Similar constructions are used in [3] to obtain specifications of deterministic partially blind multi-counter machines and their unions.

\textit{Related work.} Commutative SLIP languages have been the object of many studies, but we are not aware of any previous study on the closure of COM-SLIP under union and catenation. Our disciplined specifications of COM-SLIP have some similarity with the use in [6] of Kari’s “scattered deletion” operation.

Sect. 2 lists the basic definitions and introduces the CREG model. Sect. 3 contains the constructions and lemmas that permit to obtain a disciplined form for CREG base languages, suitable to composition by union and catenation. The conclusion mentions unanswered questions and future developments.

\section{First definitions and properties}

The terminal alphabet is $\Sigma = \{a_1, \ldots, a_k\}$ and the empty word is $\epsilon$. The number of occurrences of $a \in \Sigma$ in word $x$ is $|x|_a$, and $|x|$ is the length of $x$. The
Definition 2 (Consensual language). 
Consensual languages are defined by closure application of 1 or more languages at position \( w \), each position \( \Pi \). The families of commutative languages enjoy the SLIP property. The closure of COM-SLIP under union and catenation yields the family denoted by UP-COM-SLIP.

Consensual Languages. We present the elements of consensual language theory [1, 2]. Let \( \bar{\Sigma} \) be the marked (dotted) copy of \( \Sigma \). For each \( a \in \bar{\Sigma} \) we denote by \( \dot{a} \) the set \( \{a, \dot{a}\} \). The alphabet \( \bar{\Sigma} = \Sigma \cup \dot{\Sigma} \) is named internal, because of its use only within the match functions. To express a sort of agreement between words over the internal alphabet, we introduce a binary relation called match, over \( \bar{\Sigma}^* \). In our metaphor, such matching words provide mutual consensus on the validity of the corresponding word over \( \Sigma \), thereby justifying the name “consensual” of the language family.

Definition 1. The partial, symmetrical, and associative binary operator, match, \( @: \bar{\Sigma} \times \bar{\Sigma} \rightarrow \bar{\Sigma} \), is defined, first for all \( a \in \Sigma \), then for all words \( w, w' \in \bar{\Sigma}^n \), \( n \geq 0 \) as:

\[
\begin{align*}
\alpha @ \dot{a} &= \dot{a} @ \alpha = \alpha \\
\dot{a} @ \dot{a} &= \dot{a} \\
\text{undefined, otherwise} &
\end{align*}
\]

Thus, the match is undefined if \( |w| \neq |w'| \), or if, in some position \( i \), \( w(i) @ w'(i) \) is undefined, i.e., when both letters are in \( \Sigma \) or the two letters are not in \( \dot{a} \) for some \( a \in \Sigma \). E.g.: \( \dot{a} \dot{b} \dot{b} \dot{b} \dot{b} \) while \( \alpha \dot{a} \dot{b} \dot{b} = \dot{a} \dot{b} \dot{b} \alpha \dot{a} \dot{b} \dot{b} \) is undefined.

The match of a finite nonempty set of internal words \( w_1, \ldots, w_m \) is denoted by \( w = w_1 @ w_2 @ \ldots @ w_m \), and is a partially defined function. The number \( m \) is called the degree of the match. The match result is further qualified as strong if \( w \in \Sigma^* \), or as weak otherwise. By Def. 1, if \( w \) is a strong match, in each position \( 0 \leq i \leq |w| - 1 \), exactly one word, say \( w_h \), is unmarked, i.e., \( w_h(i) \in \Sigma \), and \( w_j(i) \in \dot{\Sigma} \) for all \( j \neq h \); we say that word \( w_h \) places the letter at position \( i \) and the other words consent to it. The match is extended to two or more languages via \( L' @ L'' = \{w' @ w'' \mid w' \in L', w'' \in L''\} \) and its repeated application is defined by \( L^{10} = L, L^{i0} = L @ L^{(i-1)0}, i \geq 2 \).

Definition 2 (Consensual language.). The closure under match, or \( @ \)-closure, of a language \( L \subseteq \Sigma^* \) is \( L^{\bar{a}} = \bigcup_{i \geq 1} L^{i0} \). Let \( B \subseteq \Sigma^* \). The consensual
language with base $B$ is defined as $\mathcal{C}(B) = B^\alpha \cap \Sigma^*$. The family of consensually regular languages, denoted by CREG, is the collection of all languages $\mathcal{C}(B)$ such that $B$ is regular.

Example. The COM-LIP language $L_1 = \text{com}(\{a^n b^n \mid n \geq 1\}) = \mathcal{C}(B_1)$ with $B_1 = \text{com}(\{ab\}) \sqcup \Sigma^*$, Similarly $L_2 = \text{com}(\{a^n b^{2n} \mid n \geq 1\})$ is generated by the base $B_2 = \text{com}(\{abb\}) \sqcup \Sigma^*$. The languages $L_1 \cup L_2$ and $L_1 \cdot L_2$ are in CREG, but they are not defined by the respective bases $B_1 \cup B_2$ and $B_1B_2$. For instance $\mathcal{C}(B_1 \cup B_2)$ contains also spurious “cross-matching” strings such as $ababb = ababb \circ ababb$. The sequel shows the translation of COM-SLIP and their unions and products to CREG.

3 Consensual Definitions of COM-LIP unions and products

We have seen that the naive implementation of COM-LIP languages results in spurious cross-matching when two or more base languages are united (or catenated). Although it is unknown whether CREG is closed under union and catenation, we present here a new technique for the COM-LIP family that eliminates cross-matching by assigning a sort of unique “signature” to the words of each base to be united or catenated, so that the match of two words with different signatures is undefined. This vague idea is implemented by assigning specific positions to the dotted and undotted letters within a word $w$ over $\tilde{\Sigma}$. For each $\tilde{a}$, we consider the projection of $w$ on $\tilde{a}$ and the numbered positions of $a$ and $\hat{a}$. Let $m$ be an integer. By prescribing that for each base language, each undotted letter $a$ may only occur in a position $j$ characterized by a specified value of the residue $j \mod m$, we discipline the bases so that they become suitable for union and catenation. We show the idea by means of the previous example. For brevity, we consider in $L_1$ and $L_2$ only strings of length multiple of three, without renaming the languages. Then $L_1$ is specified by the “disciplined” base $D_1 = \tilde{E}_1 \sqcup E_1$ where $E_1 = (\hat{a}^* \hat{a} \hat{a} a \hat{a}^*) \sqcup (\hat{b}^* \hat{b} \hat{b} \hat{b}^*)$ and $E_1 = (a \hat{a} \hat{a})^+ \sqcup (\hat{b} \hat{b} \hat{b})^+$. Every word in $L_1$ is the match of exactly one word in $E_1$ and of one or more words in $\tilde{E}_1$. Similarly, a disciplined base of $L_2$ is $D_2 = \tilde{E}_2 \sqcup E_2$ where $\tilde{E}_2 = (\hat{a}^* \hat{a} \hat{a} a \hat{a}^*) \sqcup (\hat{b}^* \hat{b} \hat{b} \hat{b}^*)$ and $E_2 = (a \hat{a} \hat{a})^+ \sqcup (\hat{b} \hat{b} \hat{b})^+$. Then $(L_1 \cup L_2) = \mathcal{C}(D_1 \cup D_2)$. Spurious strong match is impossible with the disciplined bases because $E_1$ and $E_2$ never match and either one is required for a strong match; moreover, $\tilde{E}_1$ and $E_2$ (and symmetrically $\tilde{E}_2$ and $E_1$) cannot match. Of course, things are more complex in general and we need several definitions.

Definition 3. Let $m \geq 2$, called module, and let $R \subseteq \{1, \ldots, m - 1\}$ be a nonempty set of numbers called residues. For every $a \in \Sigma$, define the finite sets in $\tilde{\alpha}^m$:

$$R_m(a) = \{ \hat{a} a^{-1} \hat{a} a^{m-r-1} \mid r \in R \}, \quad \tilde{R}_m(a) = \{ \hat{a} a^{-1} \hat{a} a^{m-r-1} \mid r \in R \}.$$  

Define the languages in $\tilde{\Sigma}^*$: $R_m = \{ x \mid \forall a \in \Sigma, \pi_a(x) \in (R_m(a))^* \}$ and $\tilde{R}_m = \{ x \mid \forall a \in \Sigma, \pi_{\hat{a}}(x) \in (\tilde{R}_m(a) \cup \hat{a}^m)^* \}$.  

Proceedings of ICTCS 2013
\[ \dot{R}_m(a) \text{ and } R_m(a) \text{ are identical by interchanging } a \text{ and } \dot{a} \text{ (i.e., by a sort of "complement"). E.g., if } m = 5, R = \{2, 3\}, \text{ then } R_5(a) = \{\dot{a}a\dot{a}a, a\dot{a}a\dot{a}\}, \text{ while } \dot{R}_5(a) = \{\dot{a}a\dot{a}a, \dot{a}\dot{a}a\dot{a}\}. \]

Moreover, \( R_m@R_m = \emptyset \) and \( R_m@\dot{R}_m = (\dot{R}_m)^@ = \dot{R}_m. \)

Given \( x, y \in \dot{\Sigma}^+ \), define the letter-by-letter catenation as: \( x \odot y = \{ z \in \dot{\Sigma}^+ \mid \forall a \in \Sigma. \pi_a(z) = \pi_a(x) \pi_a(y) \}. \)

**Definition 4.** Let \( m \geq 2 \) and \( R \) be a non empty subset of \( \{1, \ldots, m-1\} \), and let \( F \subseteq \Sigma^* \) be a finite commutative language. A base language \( E \subseteq \dot{\Sigma}^* \) is called \((m, R)\)-decomposed if there exist two languages, denoted by \( E_m \) and \( \dot{E}_m \), such that:

\[
E = E_m \cup \dot{\Sigma}^* \cdot \dot{E}_m \cdot \dot{\Sigma}^* , \quad E_m \subseteq R_m \odot F, \quad \text{and } \dot{E}_m \subseteq \dot{R}_m.
\]

By definition, it follows that every word \( w \in C(E) \) is the match of exactly one word \( z \in E_m \) and of zero, one, or more words in \( \dot{\Sigma}^* \cdot \dot{E}_m \cdot \dot{\Sigma}^* \) as next stated.

**Property 1.** If \( E \subseteq \dot{\Sigma}^+ \) is \((m, R)\)-decomposed, for some \( m, R, E_m, \dot{E}_m \) as in Def. 4, then, for every \( x \in \dot{\Sigma}^+ \), \( x \in C(E) \) if, and only if, \( x \in E_m @ (\dot{\Sigma}^* \cdot \dot{E}_m \cdot \dot{\Sigma}^*)^@ \).

A disciplined form for COM-LIP languages. Consider a COM-LIP language \( L \) over \( \Sigma = \{a_1, \ldots, a_k\} \), defined by the constant vector \( c = [c_1 \ldots c_k] \) and by a nonempty set \( P \) of periods \( p = [p_1 \ldots p_k] \). We may assume that each \( p_i \) is an even number, because the case of odd numbers can be obtained by suitable unions of COM-LIP languages, i.e., it is included in the family UP-COM-LIP, which will be proved to be CREG. Let \( m \geq 2 \) be an integer, called module. For all \( i, 1 \leq i \leq |\Sigma| \), if there exists \( p_i \in P \) with \( p_i \neq 0 \), then assume W.l.o.g. that component \( c_i \) of vector \( c \) is such that \( c_i > m \cdot p_i \) (the words not verifying this condition can be included in a finite set \( L_F \subseteq L \)). Let \( 0 < r < m \) be another integer, named residue, and define \( R \) as the singleton set \( R = \{r\} \). Next, we define two regular languages, parameterized by the module \( m \) and the residue \( r \), and the disciplined (regular) base \( D \) of \( L \).

\[
X(P, m, r) = \bigcup_{p \in P} \{ x \in \dot{R}_m \mid \forall a_i \in \Sigma. |x|_a_i = p_i \}
\]

\[
W(P, m, r) = \{ w \in \dot{\Sigma} \mid \forall a_i \in \Sigma. \pi_a(w) = c_i + \sum_{0 \leq h_p < m} p_i \} \quad (2)
\]

\[
D(L, m, r) = \dot{\Sigma}^* \cdot X(P, m, r) \cdot \dot{\Sigma}^* \cup R_m \odot W(P, m, r) \quad (3)
\]

**Comments.** Eq. (1): in each word \( x \in X(P, m, r) \subseteq \dot{R}_m \), for every \( 1 \leq i \leq k \), there are exactly \( p_i \) occurrences of \( a_i \), for some \( p \in P \); since \( x \) is also in \( \dot{R}_m \), its projection on \( a_i \) is the catenation of segments of length \( m \), where exactly \( p_i/2 \) segments have the form \( a_i a_i^{r-1} a_i^{m-r-1} \) (i.e., there is exactly a pair of undotted letters \( a_i \), occupying positions \( 1 \) and \( r + 1 \) in the segment), while each of the remaining segments is \( \tilde{a}_i^r \). For instance, consider the simpler case of \( P = \{p\} \). Given a sentence \( z \in R_p \) and sentences \( x_1, \ldots, x_n \in X(P, m, r) \), \( n \geq 1 \), if \( z \odot x_1 \odot \ldots \odot x_n \) is a strong match \( y \), then for each \( a_i \in \Sigma \), \( y \) has
exactly \(m \cdot n \cdot p_i/2\) occurrences of \(a_i\). Hence (since if \(p_i \neq 0\) then \(p_i \geq 2\) since \(p_i\) is even), \(X(\mathcal{P}, m, r)^@\otimes R_m\) can only define words whose projections on \(a_i\) has length multiple of \(m \cdot p_i\).

Eq. (2): \(W(\mathcal{P}, m, r)\) is a finite commutative language, composed of all words \(w \in L - L_F\) such that, for every \(a_i \in \Sigma\), the projections on \(a_i\) have lengths \(c_i, c_i + p_i, \ldots, c_i + (m - 1)p_i\). Hence, \(X(\mathcal{P}, m, r)^@ \otimes (R_m \odot W(\mathcal{P}, m, r))\) possibly defines also words whose projection on \(a_i\) is not a multiple of \(m \cdot p_i\).

Property 2. Let \(R = \{r\}\). Language \(D(L, m, r)\) is \((m, R)\)-decomposed (Def. 4) into \(E_m \cup \Sigma^* \cdot E_m \cdot \Sigma^*\), with \(E_m = R_m \odot W(\mathcal{P}, m, r)\) and \(E_m = X(\mathcal{P}, m, r)\).

It is possible to prove that this disciplined base consensually defines \(L\), where \(m\) and \(R\) can be arbitrary, as long as they verify Def. 4.

A language in the union and catenation of COM-SLIP languages can be described by an expression combining COM-LIP languages with union and catenation. However, since union and catenation distribute, this expression can be assumed to be the union of the catenation of COM-LIP languages. It is possible to show that also the catenation of COM-LIP languages in disciplined forms can consensually be described by a base in decomposed form. Then, it is possible to show that, just by selecting \(m\) large enough, given two bases \(B_1, B_2\) in decomposed forms, each describing the catenation of COM-LIP languages, one can always modify the bases so that the union \(B_1 \cup B_2\) does not have spurious matches, i.e., \(\mathcal{C}(B_1 \cup B_2) = \mathcal{C}(B_1) \cup \mathcal{C}(B_2)\). Hence, also the union of the catenation of COM-LIP languages is still in decomposed form:

**Theorem 1.** The closure under union and catenation of the COM-SLIP family is strictly included in the family of consensually regular languages:

\(\text{UP-COM-SLIP} \subseteq \text{CREG}\).

The inclusion is strict because CREG includes non-SLIP languages. The proof of Th. 1 requires a few more definition and lemmata. The marking function \(\text{dot} : \Sigma \to \hat{\Sigma}\) is defined as \(\text{dot}(x) = x\) if \(x \in \hat{\Sigma}\), and as \(\dot{\text{dot}}(x) = \dot{a}\) if \(x = a \in \Sigma\); it is naturally extended to words and languages. The unmarking function, \(\text{undot}\), is defined symmetrically.

**Lemma 1.** Let \(L\) be a COM-LIP language specified by the constant and periods \(\langle c, \mathcal{P}\rangle\). Then for all \(m \geq 2\) and for every \(r, 1 \leq r \leq m - 1\), there exists a finite language \(L_F\) and a disciplined base \(D(L, m, r)\) such that 

\(L = L_F \cup \mathcal{C}(D(L, m, r))\).

**Proof.** (Sketch) Let \(m > r\) and \(D = D(L, m, r)\), \(X = X(\mathcal{P}, m, r)\), \(W = W(\mathcal{P}, m, r)\) be defined as in Eq. (3). For the sake of simplicity, here we only prove the case of \(\mathcal{P} = \{p\}\). The case \(L_F \cup \mathcal{C}(D) \subseteq L\) is obvious from Eq. (3), thus we only need to prove the inclusion \(L \subseteq L_F \cup \mathcal{C}(D)\). If \(u \in L - L_F\) (if \(u \in L_F\) the claim is obvious), there exists a row vector \(n = [n_1, \ldots, n_k]\), of \(|p| = k\) elements, such that \(u\) is in \(c \ast n \ast p\). Let \(a_i \in \Sigma\) and let \(\alpha_i = \pi_{\tilde{a}_i}(w)\). By definition, \(|\alpha_i| = c_i + n_i p_i\). Consider the case \(p_i > 0\): it is \(\alpha_i = a_i^{n_i p_i} a_i^{c_i}\). Let \(h = n_i \text{ div } m\). Hence, \(n_i - h\) is a (possibly null) multiple of \(m\). Then \(\alpha_i = a_i^{(m-h)p_i} a_i^{c_i+h-p_i}\). Clearly, there exists \(z \in \pi_{\tilde{a}_i}(R_m)\).  

---

Proceedings of ICTCS 2013
such that $undot(z) = a_i^{(m-h)p_i}$ (i.e., $|z|$ is a multiple of $m$), while by definition of $W$, $a_i^{(m-h)p_i} \in W$. Therefore, there exists $z' \in \pi_a(R_m \cap W)$ such that $undot(z') = a_i$. This is true also in the case $p_i = 0$, since then $\alpha_i = a_i^0$ (just take $z' = w$). In this case, $\alpha_i = \pi_a(w)$. If $p_i > 0$, by definition of $R_m$, word $z$ has exactly two occurrences of $a_i$ in every segment of length $m$. The total number of such segments is $|z|/m$, i.e., $(n_i-h)/p_i$, $|z|$, with $n_i-h$ being a multiple of $m$. Hence, $|z|/m$ is a multiple of $p_i$. Since every word in $X$ has exactly $p_i$ occurrences of $a_i$, then $z' \in X^\alpha$ includes a word $v$ such that $\pi_a(v) = \alpha_i$. The above reasoning holds for all $a_i \in \Sigma$, hence there exists $z' \in R_m \cap W$ such that $undot(z') = u$, and $(z') \in X^\alpha$ includes $u$ itself. □

**Lemma 2.** Let $R, R' \subseteq \{1, \ldots, m-1\}$ be incompatible. For all $y \in R_m, y' \in \Sigma^+$, if there exists $a \in \Sigma$ such that a factor of $\pi_a(y')$ is in $R'_m(a)$, then $y \circ y'$ is undefined.

**Lemma 3.** For $m \geq 2$, let $E', E'' \subseteq \Sigma^+$ be, respectively, $(m, R')$- and $(m, R'')$-decomposed, for some $R', E'_m, E'_m$ and some $E''_m, E''_m, R''$ satisfying Def. 4. If $R'$ and $R''$ are incompatible, then set $R = R' \cup R''$ is such that language $E = E' \cup E''$ is $(m, R)$-decomposed into $E_m, E_m$, with:

1. $E_m = \left( \hat{\Sigma}^* E'_m \hat{\Sigma}^* \right) \cup \left( \hat{\Sigma}^* E''_m \hat{\Sigma}^* \right)$ and $E_m = E'_m \cup E''_m$

2. $C(E) = \left( C(E') \cup C(E'') \right)$

**Lemma 4.** The catenation of COM-LIP languages can be consensually defined using an $(m, R)$-decomposed base, for some $m \geq 2, R \subseteq \{1, \ldots, m-1\}$.

**Proof.** Let $L$ be a language obtained by catenation of $k > 0$ COM-LIP languages $L_1, L_2, \ldots, L_k$. By Lm. 1, for all $1 \leq i \leq k$, $L_i = C(B_i)$, where $B_i$ is the union of a finite language $F_i$ with a $(m_i, \{r_i\})$-decomposed language $E_i$, for some $m_i \geq 2, 1 \leq r_i \leq m_i - 1$. Since $m_i, r_i$ can be chosen arbitrarily large, assume each $m_i = m$, for some $m > k$, and $r_i = i$. The proof that $L_1 L_2 \ldots L_k$ is consensual is by induction on $k$. The induction hypothesis is that the catenation of $k-1$ languages as above can be consensually defined using an $(m, R)$-decomposed base, for some $m \geq 2, R \subseteq \{1, \ldots, m-1\}$. The base $k = 1$ is just Lm. 1. Consider $k > 1$. Let $L' = L_1 L_2 \ldots L_{k-1}$ and $L'' = L_k$. Let $E'' = E_k$ and $E'' = R_k$. Hence, $L'' = C(E'') \cup F_k$. By induction hypothesis, $L'$ is defined as $C(E') \cup F'$ where $F' \subseteq \Sigma^*$ is a finite set, $F'$ is an $(m, R')$-decomposed base, for some $m \geq 2, R' = \bigcup_{1 \leq i \leq k} \{i\}$. Let $E$ be the $(m, R)$-decomposed language defined by: $E_m = E'_m \cup E''_m$ and $E_m = E'_m \cdot E''_m$. We claim $C(E) \cup (F' \cdot F_k) = L' \cdot L''$, from which the main statement follows.

**Case** $(C(E')) \cup F'^* \cdot (C(E'') \cup F_k) \subseteq C(E) \cup F' \cdot F_k$. Let $x \in (C(E') \cup F') \cdot (C(E'') \cup F_k)$. The cases where $x \in F' \cdot F_k \cup C(E') \cdot F_k \cup F' \cdot C(E'')$ are obvious. If $x \in C(E') \cdot C(E'')$, then $x = x' x''$ with $x' \in C(E')$, $x'' \in C(E'')$. Hence, $x'$ (resp., $x''$) is the strong match of one $w' \in E'_m$ (resp., $w'' \in E''_m$) with a finite number of words $w'_1, w'_2, \ldots$ in $E'_m = \hat{\Sigma}^* E'_m \hat{\Sigma}^* \subseteq \hat{\Sigma}^* E'_m \hat{\Sigma}^*$ (resp., words $w''_1, w''_2, \ldots$ in $E''_m = \hat{\Sigma}^* E''_m \hat{\Sigma}^* \subseteq \hat{\Sigma}^* E''_m \hat{\Sigma}^*$. Let $p = |w'|, q = |w''|$. For each $i$, all words in $w'_i \hat{\Sigma}^*$ and all words in $\hat{\Sigma}^* w''_i$ are in $\hat{\Sigma}^* E'_m \hat{\Sigma}^*$, hence also all

---

*Proceedings of ICTCS 2013*
words \(w'_1 \cdot \dot{\text{dot}}(w'''), w'_2 \cdot \dot{\text{dot}}(w'''), \ldots\) and all words in \(w\). But since \(w''w''' \in E_m\), in \(E\) it is possible to define a strong match yielding \(x'x'' = x\), i.e., \(w''w'''@\dot{\text{dot}}(w'_1 \cdot \dot{\text{dot}}(w'''))@\dot{\text{dot}}(w'_2 \cdot \dot{\text{dot}}(w'''))@\ldots@\dot{\text{dot}}(w'_m \cdot \dot{\text{dot}}(w'''))@\ldots\), which is defined and is the catenation of \(w''\@\dot{\text{dot}}(w'_1 \@\dot{\text{dot}}w'_2 \@\ldots = x'\) with \(w''\@\dot{\text{dot}}(w'_2 \@\dot{\text{dot}}w'_3 \@\ldots = x''\).

**Case** \(C(E') \cup F \subseteq (C(E') \cup F') \cdot (C(E'') \cup F_k)\). Let \(x \in C(E')\) (the case \(x \in F\) is obvious). \(E\) is \((m, R)\)-decomposed and there exist \(w \in E_m\) and \(1 \leq n \leq |x|\) words \(w_1, w_2, \ldots, w_n\) such that \(w_1 \in E_m\) and \(w_2, \ldots, w_n \in E_m\) and such that \(w_1 \@ w_2 \@ \ldots \@ w_n = x\). We again assume that each \(w_i \not\in \Sigma^+\), \(1 \leq i \leq n\), since otherwise the removal of \(w_i\) does not affect the match result \(x\).

By definition of \(E\), \(w_1 \in E_m\) can be decomposed as the catenation of two words \(w'_1 \in E_m', w''_1 \in E_m''\), while each \(w_i, 2 \leq i \leq n\), is either in \(E_m'\) or in \(E_m''\). Assume, by contradiction, that \(x \not\in C(E') \cdot C(E'')\). Since \(x\) is the match of a word \(w_1\) with other words in \(E\), then \(x = \dot{\text{undot}}(w_1)\). Hence, since \(x \not\in L'L''\), either \(\dot{\text{undot}}(w'_1) \not\in L'\) or \(\dot{\text{undot}}(w''_1) \not\in L''\). We consider only the former case, the other being analogous. Let \(q = [w'_1]\). Strong match \(w'_1@w_1(1,q)@w_2(1,q)@\ldots@w_n(1,q)\) is defined, resulting in a word \(x' \not\in L'\). Hence, there exists \(j, 2 \leq j \leq n\), such that \(w_j(1,q) \not\in \Sigma^*E_m'\Sigma^*\), with \(w_j(1,q) \not\in \Sigma^+\). There are two cases: either \(w_j \in \Sigma^*E_m'\Sigma^*\) or \(w_j \in \Sigma^*E_m''\Sigma^*\).

We consider only the latter case, the other one being analogous. Let \(a \in \Sigma\) be the leftmost terminal symbol in \(w_j(1,q)\), occurring at a position \(p \leq q\). Let word \(\beta = \pi_0(a, p, q)\), with \(\beta(1) = a\). We claim that \(|\beta| > m\) (actually, \(|\beta| \geq m + k\), i.e., \(\beta\) must include at least another occurrence of \(a\), at position \(k + 1 < m\) of \(\beta\): \(\beta(k + 1) = a\) (with \(k \in R^\alpha\)). In fact, by definition of the consensual base for a COM-LIP language, every \(w' \in E_m\) \(\pi_0(a, w')\) has a suffix of the form \(a^*a^+m\): if \(|\beta| \leq m\) then at least one of the occurrences of \(a\) in \(\beta\) would be at a position \(s, p < s < q\), in \(w_j(1,q)\): but \(w'_1(s) = a\), hence \(w'_1@w_j(1,q)\) would be undefined. Hence, \(|\beta| > m\). Then, \(\beta\) has a prefix in \(R_m'(a)\), i.e., also \(\pi_0(a, w_1(1,q))\) has a factor in \(R_m'(a)\): by Lm. 2, with \(y = w'_1\) and \(y'' = w_j(1,q)\), then \(w'_1@w_j(1,q)\) is undefined, therefore also \(w_1@w_j\) is undefined, a contradiction.

**Proof of Theorem 1.** A language in the union and catenation closure of COMSLIP languages can be described, by distributivity, by an expression \(\eta\) defined as the union of the catenation of COM-LIP languages. Let \(L(\eta)\) be the language defined by \(\eta\). By Lm. 4, the catenation of COM-LIP languages may be described as the union of a finite language with a suitable \((m, R)\)-decomposed base; by Lm. 3, Part (2), the union of these languages is still consensual. The proof that \(L(\eta)\) is consensual is by structural induction on the height \(k\) of \(\eta\). The inductive hypothesis is that if every expression \(\eta\) as above has height \(k\) then for all \(m \geq k + 1\), for all \(R \subseteq \{1, \ldots, m - 1\}\), with \(|R| \geq k\) there exists a \((m, R)\)-decomposed language \(E \subseteq \Sigma^+\) such that \(C(E) = L(\eta)\). The thesis then follows immediately. The base case \(k = 1\) is Lm. 1. For some \(k > 1\) let \(eta\) be an expression of height \(k\). Hence, \(\eta = \eta' \cup \eta''\) or \(\eta = \eta' \cdot \eta''\), for some expression \(\eta'\), \(\eta''\) both of height less than \(k\). By induction hypothesis, for all \(m, m' \geq k\), \(R', R''\) satisfying the hypothesis there exist a \((m', R')\)-decomposed language \(E' \subseteq \Sigma^+\) and a \((m'', R'')\)-decomposed language \(E'' \subseteq \Sigma^+\), such that \(C(E') = L(\eta')\) and \(C(E'') = L(\eta'')\). Among all possible values for \(m', m'', R', R''\), choose \(m' = m''\).
with $m'$ large enough so that it is possible to define $R'$ and $R''$ to be disjoint. By Lm. 3, for $R = R' \cup R''$ and $m = m'$, there exists a $(m, R)$-decomposed language $E$ such that $C(E) = C(E') \cdot C(E'')$, hence $C(E) = L(\eta)$. Moreover, $m, R$ verify the other conditions of the inductive hypothesis. The inclusion is strict, since language $\{ba^1ba^2ba^3 \ldots ba^k | k \geq 1\}$ has a non-SLIP commutative image, but it is in CREG [1].

4 Conclusion

First, we state some unanswered questions, then we outline possible developments. Is the star of a COM-SLIP language in CREG? And then, is the family obtained by closing COM-SLIP under union, catenation and star included in CREG? Another question is whether all commutative languages that are in CREG are SLIP. For the unary alphabet, a positive answer would imply that unary CREG languages are regular. Beyond the present results, it would be interesting to characterize a broader class of multi-counter languages that are in CREG and are closed under union and catenation.

References

Combining Declarative and Procedural Views in the Feature-Oriented Specification and Analysis of Product Families

Maurice H. ter Beek¹, Alberto Lluch Lafuente², and Marinella Petrocchi³

¹ ISTI–CNR, Pisa, Italy maurice.terbeek@isti.cnr.it
² IMT Institute for Advanced Studies Lucca, Italy alberto.lluch@imtlucca.it
³ IIT–CNR, Pisa, Italy marinella.petrocchi@iit.cnr.it

Abstract. We present the recently introduced feature-oriented language FLan as a proof of concept for specifying both declarative aspects of product families, namely constraints on their features, and procedural aspects, namely feature configuration and run-time behaviour. FLan is inspired by the concurrent constraint programming paradigm. A store of constraints allows one to specify in a declarative way all the constraints on features that are commonly used in software product line engineering, including the cross-tree constraints well known from feature models. A standard yet rich set of process-algebraic operators allows one to specify in a procedural way the configuration and behaviour of products. There is a close interaction between these two views: (i) the execution of a process is constrained by its store to forbid undesired configurations; (ii) a process can query a store to resolve design and behavioural choices; (iii) a process can update the store by adding new features.

1 Introduction

The last decades have witnessed a paradigm shift from mass production to mass customization to serve as many individual customer’s needs as possible. Software Product Line Engineering (SPLE) has translated this into a software engineering approach aimed at developing, in a cost effective way, a variety of software-intensive products that share an overall reference model, i.e. that together form a product family. Usually, commonality and variability are defined in terms of features, and managing variability is about identifying variation points in a common family design to encode exactly those combinations of features that lead to valid products. The actual configuration of the products during application engineering then boils down to selecting desired options in the variability model.

Feature models are the most widely used variability model [15]. They provide a compact representation of all products of a family in terms of features, and additional constraints among them. Graphically, features are drawn as nodes of a tree, with the family as its root and relations between these features representing constraints. However, there may be thousands of features, requiring models with thousands of options, which easily leads to anomalies such as superfluous or—worse—contradictory variability information (e.g. so-called false optional features and dead features). There is a large body of literature...
on computer-aided analyses of feature models to extract valid products and to
detect anomalies [4].

None of these analyses consider behavioural variability, though, in the sense
that only the presence of the software implementing the features is considered—
not their ordering in time. Indeed, research on applying formal methods in
SPLE traditionally focusses on modelling and analysing structural rather than
behavioural constraints in product families. However, many software-intensive
systems are embedded, distributed and safety critical, making it important
to be able to model and analyse also their behaviour, as a form of quality
assurance.

Recent years have witnessed a growing interest in specifically considering
also the behavioural variability of product families. This has resulted in, among
others, extensions of Petri nets [13] and a variety of frameworks with an LTS-
like semantics [8,11,10,12,7,1]. As a result, behavioural analysis techniques such
as model checking have become available for the verification of (temporal)
logic properties of product families. Specifying a product family directly in an
operational model is often not feasible. Therefore it can be useful to resort
to high-level formal languages with semantics over those operational models,
as is common in the context of process algebra. Several extensions of CCS
have been proposed to model product families [10,9], but none can combine
behavioural constraints with all common structural constraints known from
feature models [15].

We present here our current research on FLan: a feature-oriented language
for specifying product families by taking structural and behavioural constraints
into account [2]. It is inspired by concurrent constraint programming [14] and
its adoption in process calculi [5]. A store of constraints allows one to specify all
common structural constraints known from feature models in a declarative way,
incl. cross-tree constraints. Moreover, a rich set of process-algebraic operators
allows one to specify in a procedural way both the configuration and behaviour
of products. These declarative and procedural views are closely related: (i) the
execution of a process is constrained by its store, e.g. to avoid introducing
inconsistencies; (ii) a process can query a store to resolve options regarding
the design and behaviour; (iii) a process can update the store by adding new
features. To this aim, the semantics of FLAN unifies static and dynamic feature
selection in an elegant fashion. Inspired by [9], we implemented FLAN in the
executable modelling language Maude (http://maude.cs.uiuc.edu/), whose
rich toolkit allows the application of a variety of formal automated analysis
techniques, from consistency checking to model checking, to product families
specified in FLAN.

This paper is an extended abstract of [2]. It is organised as follows. §2
describes a running example of a family of coffee machines. In §3, we present
the syntax and semantics of FLAN and a specification of the example, while
we refer to [2] for an illustration of its Maude-supported automated analyses.
We report some concluding remarks and list promising future work in §4, while
we refer to [2] for a detailed discussion of related work.
2 A Family of Coffee Machines

We use a popular running example in the style of [1,3,7,9,13]. It describes a (simplified) family of coffee machines in terms of the following requirements.

1. Initially, a coin must be inserted: either a euro, exclusively for European products, or a dollar, exclusively for Canadian products;
2. Upon the insertion of a coin, a choice for sugar must be offered, followed by a choice of beverages;
3. The choice of beverage (coffee, tea, cappuccino) varies, but every product must offer at least one beverage, tea may be offered only by European products, and all products that offer cappuccino must also offer coffee;
4. Optionally, a ringtone may be rung after the delivery of a beverage. However, a ringtone must be rung after serving a cappuccino;
5. After the beverage is taken, the machine returns idle.

These requirements define products by combining structural constraints defining valid feature configurations (e.g. “every product must offer at least one beverage”) with temporal constraints defining valid behaviour (e.g. “a ringtone must be rung after serving a cappuccino”). Behaviour is not captured at all in feature models.

3 FLAN: Syntax and Semantics

The feature-oriented language FLAN we propose here is loosely inspired by the CCS-like process algebra CLASPL [9], but it differs in its treatment of the cross-tree constraints known from feature models and in the separation of declarative and procedural aspects inspired by the concurrent constraint programming paradigm [14] and its adoption in process calculi [5]. FLAN’s core notions are features, constraints, processes and fragments, which can be identified in the syntax of FLAN in Fig. 1. More precisely, $f$ and $g$ range over features and syntactic categories $S$, $P$ and $F$ correspond to constraints, processes, and fragments.

A feature is a term describing specific elements or properties of a product. The universe of features is denoted by $F$. The features of our running example are the coins accepted (i.e. euro and dollar), the products offered (i.e. coffee, tea and cappuccino) and additional elements such as sugar (the capability to regulate the delivery of sugar) and ringtone (the capability to emit a ringtone).

The declarative part of FLAN is represented by a store of constraints which defines both constraints on features extracted from the product requirements and additional information (e.g. about the context wherein the product will operate). Two important notions of constraint stores are (i) the consistency of a store $S$, denoted by $\text{consistent}(S)$, which in our case amounts to logical satisfiability of all constraints forming $S$; and (ii) the entailment $S \vdash c$ of constraint $c$ in store $S$, which in our case amounts to logical entailment. A constraint store is any term generated by $S$ in the grammar of FLAN. The most basic constraint stores are $\top$ (no constraint at all), $\bot$ (inconsistent) and
Fig. 1. The syntax of FLan, where $a \in A$, $p \in P$ and $f, g \in F$ ordinary boolean propositions (generated by $K$). Constraints can be combined by juxtaposition.

We assume that all standard structural constraints known from feature models (optional, mandatory and alternative—or and xor—features) are expressed using boolean propositions (e.g. as explained in [15]). For this purpose, we assume that the universe $P$ of propositions contains a Boolean predicate $\text{has}(\cdot) : F \to B$ that can be used to denote the presence of a feature in a product. Boolean propositions can also be used to represent additional information such as contextual facts. Examples from our running example are $\text{in}(\text{Europe})$ and $\text{in}(\text{Canada})$, respectively used to state the fact that the coffee machine being configured is meant to be used in Europe or in Canada. Boolean propositions can state relations between contextual information and features, like $\text{in}(\text{Europe}) \to \text{has}(\text{euro})$ (i.e. a coffee machine for the European market needs a euro coin slot).

The cross-tree constraints known from feature models (requires and excludes) are instead handled as first-class citizens to emphasise the way we deal with them. A constraint $f \triangleright g$ expresses that feature $f$ requires the presence of feature $g$ while a constraint $f \otimes g$ expresses that features $f$ and $g$ mutually exclude each other’s presence (i.e. they are incompatible). Of course, also these constraints can be encoded as boolean propositions. For instance, $f \otimes g$ and $f \triangleright g$ can equivalently be expressed as $\text{has}(f) \iff \neg \text{has}(g) \text{ and } \text{has}(f) \to \text{has}(g)$, respectively. We use indeed such logical encoding to reduce consistency checking and entailment to logical satisfiability (and hence exploit Maude’s SAT solver).

We also consider a class of action constraints, reminiscent of Featured Transitions Systems [7], where transitions are subject to the presence of features. For instance, in a coffee machine equipped with a slot for euro coins we will use euro for the action of inserting a euro coin and $\text{do}(\text{euro})$ as a proposition stating the execution of that action. The relations between the action euro and the presence of the corresponding feature euro can be formalised as $\text{do}(\text{euro}) \to \text{has}(\text{euro})$, i.e. the insertion of a euro coin requires the presence of an appropriate coin slot. In general, we assume that each action $a$ may have a constraint $\text{do}(a) \to p$. These act as a sort of guard to allow or forbid the execution of actions (see later).

The constraint store $S$ in Fig. 2 formalises part of the requirements specified in §2 for our running example. It contains both contextual information (e.g. $\text{in}(\text{Europe})$) and action constraints (e.g. $\text{do}(\text{euro}) \to \text{has}(\text{euro})$). For instance, from requirement 1 we extract that euro and dollar are mutually
exclusive features (formalised as dollar $\otimes$ euro), while from requirement 3 we understand that cappuccino requires coffee (formalised as cappuccino $\gg$ coffee).

The procedural part of FLAN is represented by processes of the following type:

- $0$, the empty process that can do nothing;
- $X$, where $X$ is a process identifier (we assume that there is a set of process definitions of the form $X \doteq P$ and we also assume that recursively defined processes are finitely branching, which can be ensured in standard ways, e.g. prefixing every occurrence of a process identifier $X$ with an action or constraining process definitions to be of the form $X \doteq A.P$);
- $A.P$, a process willing to perform the action $A$ and then to behave as $P$;
- $P + Q$, a process that can non-deterministically choose to behave as $P$ or $Q$;
- $P.Q$, a process that must progress first as $P$ and then as $Q$;
- $P | Q$, a process formed by the parallel composition of $P$ and $Q$, which evolve independently.

Note that we distinguish between ordinary actions (from a universe $A$) and the special actions install($f$) (used to denote the dynamic installation of a feature $f$) and ask($K$) (used to query the store). We will see that each action type is treated differently in rules of the operational semantics. In our example we consider the actions euro, dollar (insertion of the respective coin); sugar (sugar selection); coffee, tea, cappuccino (beverage selection); and ringtone (ringtone emission).

A fragment $F$ is a term $[S \parallel P]$, formed by a store of constraints $S$ and a process $P$, which may influence each other, as in concurrent constraint programming [14]: a process may update its store which, in turn, may condition the execution of process actions. The operational semantics of closed fragments (i.e. its
The reduction semantics (inst) is formalised by the transition relation $\rightarrow \subseteq T \times T$ of Fig. 3, with $T$ denoting the set of all terms generated by $F$ in the grammar of Fig. 1. Technically, such reduction relation is defined in SOS style modulo a structural congruence relation $\equiv \subseteq T \times T$, which allows to identify different ways to denote the same fragment. We consider the least congruence on fragments closed w.r.t. commutativity and associativity of non-deterministic and parallel composition of processes $(P \parallel Q) \equiv Q \parallel P$, $(P + Q) \equiv (P + Q) + R$, $P | Q \equiv Q | P$ and $P | (Q | R) \equiv (P | Q) | R$; associativity of sequential composition of processes $(P; (Q; R)) \equiv (P; Q); R$; identity of non-deterministic choice, sequential and parallel composition of processes $(P+0) \equiv P$, $P \equiv P$, $P; 0 \equiv P$ and $P | 0 \equiv P$; and expansion of recursive process definitions $(P \equiv P^{\ast} / f)$ if $X \equiv Q$. This choice is not accidental: all can be naturally and efficiently treated by Maude, so our semantics enjoys several nice properties: (1) it is (efficiently) executable; (2) each semantic rule of Fig. 3 corresponds to exactly one conditional rewrite rule in Maude’s implementation of FLAN (cf. [2]); (3) the number of reduction rules is small, so semantics and implementation are compact and easy to read.

Rules INST and ACT are very similar, both allowing a process to execute an action if certain constraints are satisfied. In particular, rule INST forbids inconsistencies due to the introduction of new features. Note that rule INST can be seen as a particular instance of the rule for the tell operation of concurrent constraint programming [14] instantiated as tell(has(f)). Rule ACT forbids inconsistencies with respect to action constraints. A typical case of action constraint is $do(a) \rightarrow has(f)$, i.e. action $a$ is subject to the presence of feature $f$. Rule ASK formalises the semantics of the usual ask(·) operation as known from concurrent constraint programming [14]. It allows to block a process until a proposition can be derived from the store. Rule OR is quite straightforward. It allows the process to evolve as any of the branches. It is worth remarking that non-determinism can be solved at the procedural level (by relying on ask(·) actions) or at the declarative level (by using a non-deterministic choice that may be solved by the constraint store), thus providing a lot of flexibility to fragment designers (as illustrated in [2]). Rules SEQ, formalising the usual sequential composition, and PAR, formalising an interleaving parallel composition, are standard.

Note the different ways in which feature $f$ can be selected in configurations. First, this can be done in an explicit and declarative way by

\begin{align*}
\text{(INST)} \quad \frac{\text{consistent}(S \ \text{has}(f))}{S \parallel \text{install}(f).P \rightarrow [S \ \text{has}(f)] P} \\
\text{(ASK)} \quad \frac{S \vdash K}{S \parallel \text{ask}(K).P \rightarrow [S] P} \\
\text{(ACT)} \quad \frac{S \vdash (\text{do}(a) \rightarrow K)}{S \parallel a.P \rightarrow [S] P}
\end{align*}

\begin{align*}
\text{(OR)} \quad \frac{[S \parallel P] \rightarrow [S'] P'}{[S \parallel P + Q] \rightarrow [S'] P'} \\
\text{(SEQ)} \quad \frac{[S \parallel P] \rightarrow [S'] P'}{[S \parallel P; Q] \rightarrow [S'] P'; Q} \\
\text{(PAR)} \quad \frac{[S \parallel P] \rightarrow [S'] P'}{[S \parallel P; Q] \rightarrow [S'] P'; Q}
\end{align*}

Fig. 3. The reduction semantics of FLAN
including the proposition has($f$) in the initial store. This would be the case of features that the system designer is sure to be mandatory for all the family’s products. Second, the presence of feature $f$ can be obtained in an implicit and declarative way, meaning that $f$ may be derived as a consequence of further constraints. This would be the case of features that apparently seem not to be mandatory to the system designer, but that are indeed enforced by the constraints (e.g. in a store containing the constraints $g \Rightarrow f$ and has($g$) the presence of $f$ can be inferred). Third, feature $f$ can be dynamically installed in a procedural way during process execution. This is a key aspect of our approach as it enables the designer to delay feature configuration decisions and to specify them procedurally. FLAN’s concurrent constraint approach allows to combine these three declarative and procedural forms of feature configuration in an elegant and consistent way.

Figure 2 shows a specification of the coffee machine. Fragment $F$ is formed by store $S$ and the sequential composition of processes $D$, specifying an initial design phase, and $R$, specifying the run-time behaviour of the coffee machine. The store $S$ is made of two parts: constraints derived from the requirements specification ($S_1$) plus some contextual information and initial configurations ($S_2$). Note that the action constraints are quite simple (all are of the form $do(f) \rightarrow has(g)$) but recall that they could be more sophisticated if needed. For instance, one could specify the constraint on action cappuccino as $do(cappuccino) \rightarrow has(cappuccino) \land has(ringtone)$ thus requiring not only the presence of the corresponding feature but also that of the ringtone feature.

The configuration process $D$ is quite simple. It is just formed by the parallel composition of the installation of some of the features that the coffee machine may exhibit. This specifies a sort of race between features and may be thought of as independent designers competing to install the features they are responsible for. The semantics of FLAN ensures that all executions will end up with a consistent configuration if the process begins with a consistent store. For instance, the semantics will forbid the installation of mutually exclusive features. Process $R$ describes the coffee machine’s run-time operation. Depending on the country it is meant for, the machine may either accept a euro or a dollar. This is implemented as a non-deterministic choice that will be consistently solved at run-time due to the presence of the action constraints $do(euro) \rightarrow has(euro)$ and $do(dollar) \rightarrow has(dollar)$, which will forbid the use of actions euro or dollar if the corresponding feature has not been installed. After that, it may ($P_2$) or it may not ($P_3$) deliver sugar. The next step is beverage selection and delivery, which may be followed by a ringtone ($P_3$) or not, after which it returns idle. Note that $D$ and $R$ are not pure configuration and run-time processes. Indeed, feature ringtone is not installed by $D$ but by $R$, i.e. the feature ringtone is installed dynamically and it can be thought of as, e.g., a software module. This is an interesting example of a partial configuration process, where some non-mandatory features are not installed and products are only partially configured, and a run-time configurable process that installs features when needed.
4 Conclusion

The concurrent constraint programming paradigm adopted in FLan provides a flexible mechanism for separating and (when necessary) combining declarative and procedural aspects. For instance, design decisions can be delayed until runtime, which is very convenient for software product families that allow features to be added while the system operates. Also, the run-time specification can be relieved from design decisions like feature constraints, thus resulting in lightweight, understandable specifications. Moreover, FLAN’s implementation in Maude allows to exploit this framework’s rich analysis toolset (as done in [2]).

We envisage several potentially interesting extensions of FLAN. First, we can adopt further primitives and mechanisms from the concurrent constraint programming tradition. The concurrent constraint π-calculus [5], e.g., provides synchronisation mechanisms typical of mobile calculi (i.e. name passing), a check operation to prevent inconsistencies, a retract operation to remove (syntactically present) constraints from the store and a general framework for soft constraints (i.e. not only boolean). Second, we may equip FLAN with a semantics over known suitable models for product families, like Featured Transition Systems [7] and Modal Transition Systems [8,11,12,1], so that FLAN becomes a high-level language for them and we can exploit their specialised analysis tools (e.g. [6,3]).

References


Proceedings of ICTCS 2013
Polarized Multigames*

Furio Honsell¹, Marina Lenisa¹, Rekha Redamalla²

¹ Dipartimento di Matematica e Informatica, Università di Udine, Italy.
   furio.honsell@comune.udine.it, marina.lenisa@uniud.it
² Birla Science Centre Hyderabad, India.
   rekhar@yahoo.com

Introduction

In traditional game semantics (see e.g. [AJM00,H000]), games are sequential, i.e., at each step, either Player or Opponent moves (global polarization), moreover, only a single move can be performed at each step. More recently, concurrent games (see e.g. [AM99,RW11,CGW12]) have been introduced, where global polarization is abandoned, and multiple moves are allowed. In this paper, we introduce polarized multigames, which are situated half-way between traditional sequential game semantics and concurrent game semantics: global polarization is still present, however multiple moves are possible at each step, i.e. a team of Players/Opponents moves in parallel. More precisely, we define multigames as games where plays are sequences of multimoves, i.e. finite sets of (atomic) moves with the same polarity. The notion of strategy as well as the usual game constructions, such as tensor product, linear implication, and exponential, can be naturally extended to multigames. Moreover, we show that multigames and strategies can be endowed with a structure of a monoidal closed category together with an exponential comonad. The main difference between the category of multigames and categories of traditional games such as those in [AJM00,H000] lies in the fact that, in the tensor product of traditional games, at each step, the current player can move in exactly one component, while in the multigame tensor, by exploiting the parallel nature of multigames, in general the current player can perform a multimove consisting of atomic moves on both components. Similarly, in multigame strategies, at each step, the current player can possibly perform a multimove consisting of atomic moves on both components. As a consequence, the notion of strategy composition on multigames is not a straightforward adaptation of usual composition, but it requires a non-standard parallel application of strategies. Interestingly, the category of polarized multigames turns out to be equivalent to a category of games à la [AJM00] (AJM-games) with a new notion of parallel tensor product, where at each step the current player performs a move in at least one component of the tensor game.

While traditional tensor on games is related to Conway’s disjunctive sum, [Con01], parallel tensor product is inspired by Conway’s selective sum. Categories of coalgebraic games, i.e. possibly non-terminating games generalizing Conway’s games and other notions of games, have been recently studied in [HLR12,HLP13], where disjunctive or selective sum is taken as tensor product.

* Work supported by MIUR PRIN Project CINA 2010LHT4KM.

Proceedings of ICTCS 2013
In particular, in [HLR12], categories of coalgebraic games with disjunctive sum are related to categories of traditional Game Semantics. On the other hand, the categories of AJM-games with parallel tensor product presented in this paper are related to the categories of coalgebraic games with selective sum studied in [HLP13].

Multigames, as they are defined in this paper, can be viewed as a special case of concurrent games of [AM99]. Hence, one may ask what is their interest or to what extent they offer a model of parallelism. A first answer provided in the present paper is that they offer a sufficient level of parallelism for modeling (in a universal way) unary PCF with parallel or. We think that multigames can be also useful to model more complex languages with parallel features, so as to understand concurrency in proof theory (see e.g. [Abr03]). Further experiments are left for future work so as a comparison with more recent notions of concurrent games, e.g. those in [RW11, CGW12].

1 Multigames

A multigame is a game in the [AJM00]-style, where plays are sequences of multimoves, i.e. non-empty finite sets of atomic moves. The notion of strategy on games admits a natural extension on multigames. Similarly, the usual game constructors, such as tensor product, linear implication, and exponential, admit a natural definition on multigames, and they amount to the parallel counterparts of the traditional game constructors.

Definition 1 (Multigames). A multigame has two participants, Player (P) and Opponent (O). A multigame $A$ is a quadruple $(M_A, \lambda_A, P_A, \approx_A)$, where

- $M_A$ is the set of atomic moves of the game.
- $\lambda_A : M_A \to \{O, P\}$ is the labeling function: it tells us if an atomic move is taken by the Opponent or by the Player. We denote by $\overline{\lambda_A}$ the function which exchanges Player and Opponent, i.e. $\overline{O} = P$ and $\overline{P} = O$. We denote by $\overline{\lambda_A}$ the function which exchanges the polarity of moves, $\overline{\lambda_A}(a) = \overline{\lambda_A}(a)$.
- The set of atomic moves $M_A$ determines the set $M_A$ of multimoves of the game, which are non-empty finite sets of atomic moves with the same polarity. We denote by $\lambda_A$ the obvious extension of $\lambda_A$ to multimoves.
- $P_A$ is the set of plays, i.e. a non-empty and prefix-closed subset of the set $M_A^*$, which satisfies the following conditions:
  - $s = \alpha t \Rightarrow \forall a \in \alpha. \lambda_A(a) = O$ (O starts)
  - $(\forall i : 1 \leq i \leq |s|)[\lambda_A(s_{i+1}) = \overline{\lambda_A(s_i)}]$ (alternating)
  - $s \approx_A s' \Rightarrow |s| = |s'|
  - so \approx_A s'\alpha' \Rightarrow s \approx_A s'
  - s \approx_A s' \& s\alpha \in P_A \Rightarrow (\exists \alpha')[s\alpha \approx_A s'\alpha']$

The set of even-length (odd-length) plays will be denoted by $P_A^\text{even}$ ($P_A^\text{odd}$). The empty sequence will be denoted by $\epsilon$.  

In what follows, we will often refer to multimoves simply as moves.

The difference between the above notion of multigame and the standard notion of AJM-game lies in the definition of plays, which are sequences of atomic moves on AJM-games, while on multigames they are sequences of multimoves.

Notice that every multigame can be viewed as a game whose moves are the multimoves, i.e. any multigame $A = (M_A, \lambda_A, P_A, \approx_A)$ induces a game $A_g = (M_A, \lambda_A, P_A, \approx_A)$. However, as we will see, multigame tensor (as well as other game constructions) is not preserved under this mapping.

The notion of strategy naturally extends to multigames:

**Definition 2 (Strategies).** A strategy for the Player on a multigame $A$ is a non-empty set $\sigma \subseteq P_A^{\text{even}}$ of plays of even length such that

- $\sigma = \sigma \cup \text{dom}(\sigma)$ is prefix-closed, where $\text{dom}(\sigma) = \{ t \in P_A^{\text{odd}} \mid (\exists \alpha)[t\alpha \in \sigma]\}$,
- $s\alpha\beta, t\alpha\gamma \in \sigma \Rightarrow \beta = \gamma$ (determinism).

A strategy $\sigma$ for a multigame $A$ is history-free if it satisfies the following property: $s\alpha\beta, t \in \sigma, t\alpha \in P_A \Rightarrow t\alpha\beta \in \sigma$.

The equivalence relation on plays $\approx_A$ can be naturally extended to strategies:

**Definition 3.** Let $\sigma, \tau$ be strategies, $\sigma \approx \tau$ if and only if

- $s\alpha\beta \in \sigma, s'\alpha'\beta' \in \tau, s\approx_A s'\alpha' \Rightarrow s\alpha\beta \approx_A s'\alpha'\beta'$
- $s \in \sigma, s' \in \tau, s\approx_A s'\alpha' \Rightarrow (\exists \beta)[s\alpha\beta \in \sigma] \iff (\exists \beta')[s'\alpha'\beta' \in \tau]$

Multigame Constructions.

**Definition 4 (Tensor Product).** Given multigames $A$ and $B$, the tensor product $A \otimes B$ is the multigame defined as follows:

- $M_{A \otimes B} = M_A + M_B$
- $\lambda_{A \otimes B} = [\lambda_A, \lambda_B]$
- $P_{A \otimes B} \subseteq M_{A \otimes B}$ is the set of plays, $s$, which satisfy the following condition: the projections on each component (written as $s \mid A$ or $s \mid B$) are plays for the games $A$ and $B$ respectively.
- $s \approx_{A \otimes B} s' \iff s \mid A \approx_A s' \mid A, s \mid B \approx_B s' \mid B, (\forall i)[s_i \in M_A \Leftrightarrow s'_i \in M_B]$

Notice that, on the tensor product of multigames, $A \otimes B$, a multimove can contain atomic moves of both $A$ and $B$. As a consequence, $(A \otimes B)_g$ is not isomorphic to $A_g \otimes B_g$. However, as we will see in the Section 3, this isomorphism holds when a new notion of parallel tensor on standard AJM-games is considered.

**Definition 5 (Linear Implication).** Given multigames $A$ and $B$, the linear implication multigame $A \rightarrow B$ is defined as follows:

- $M_{A \rightarrow B} = M_A + M_B$
\[ \lambda_{A \rightarrow B} = [\lambda_{A}, \lambda_{B}] \]

\[ P_{A \rightarrow B} \subseteq \mathcal{M}_{A \rightarrow B}^{A \rightarrow B} \]

is the set of plays, \( s \), which satisfy the following condition:

\[ s \upharpoonright A \text{ and } s \upharpoonright B \text{ are plays for the multigames } A \text{ and } B \text{ respectively.} \]

\[ s \approx_{A \rightarrow B} s' \iff s \upharpoonright A \approx_{A} s' \upharpoonright A, s \upharpoonright B \approx_{B} s' \upharpoonright B (\forall i)[(s_i \in \mathcal{M}_A \Leftrightarrow s'_i \in \mathcal{M}_A) \land (s_i \in \mathcal{M}_B \Leftrightarrow s'_i \in \mathcal{M}_B)] \]

As on the tensor multigame, also on linear implication multimoves can include moves on both components. Similarly, on the exponential multigame \(!A\), moves on finitely many components are possible at each step. We omit the formal definition.

## 2 Categories of Multigames and Strategies

The main difficulty in defining the category of multigames is the definition of strategy composition, which is based on a non-standard parallel composition. The difficulty arises from the fact that a multimove in a strategy between \( A \) and \( B \) can include atomic moves on both \( A \) and \( B \).

Let \( \mathcal{M} \) be the category defined by:

**Objects:** multigames.

**Morphisms:** a morphism between games \( A \) and \( B \) is an equivalence class of strategies \( \sigma : A \rightarrow B \) w.r.t. the relation \( \approx_{A \rightarrow B} \).

**Identity:** the identity \( id_A : A \rightarrow A \) is (the equivalence class of) the copy-cat strategy, defined by: \( id_A = \{ s \in P_{A}^{\text{even}} \mid s \upharpoonright 1 = s \upharpoonright 2 \} \). This definition works thanks to the fact that games are polarized. On the game \( A \rightarrow A \), Opponent can only open on the righthand \( A \) component, then Player proceeds by copying the moves on the lefthand \( A \) component and so on, thus at each step Opponent has exactly one component to move in.

**Composition:** the composition is given by the extension on equivalence classes of the composition of strategies \( \sigma : a \rightarrow B, \tau : B \rightarrow C \), obtained via the swivel-chair strategy and a non-standard parallel application of strategies. The difficulty in defining composition lies in the fact that, in the composition \( \tau \circ \sigma : A \rightarrow C \), there are situations where the Opponent can move in both \( A \) and \( C \) components. In this case, we have to apply \( \sigma \) and \( \tau \) in parallel, and to show that the applications of the two strategies in parallel do not interfere. For lack of space, we omit the proof.

The multigame constructions of tensor product and linear implication can be made functorial, determining a structure of a symmetric monoidal closed category on \( \mathcal{M} \), with the empty multigame \( I = (\emptyset, \emptyset, \{ e \}, \{(e, e)\}) \) as tensor unit.

Strategy composition is closed under history-free strategies, and hence we can consider the corresponding subcategory \( \mathcal{M}_{hf} \).

The standard co-kleisli construction of a cartesian closed category from the exponential easily extends to multigames, getting CCC’s \( K_i(\mathcal{M}) \) and \( K_i(\mathcal{M}_{hf}) \).
3 Categories of Games with Parallel Tensor Product

In this section, we build a new category, where objects are AJM-games, but the usual tensor product is replaced by a new notion of parallel tensor.

In the standard tensor product of games, see e.g. [AJM00], on the game $A \otimes B$, at each step, the player who has the turn can move exactly in one of the two components, $A$ or $B$. On parallel tensor product $A \parallel B$, at each step the player who has the turn can either move in $A$, or in $B$, or in both components.

The category of AJM-games with parallel tensor turns out to be equivalent to the category of multitags introduced in Section 2.

Let us consider AJM-games and (history-free) strategies defined in the usual way. On these games, we introduce the following parallel tensor product:

**Definition 6 (Parallel Tensor Product).** Let $A = (M_A, \lambda_A, P_A, \approx_A)$ and $B = (M_B, \lambda_B, P_B, \approx_B)$ be AJM-games, the tensor product $A \parallel B$ is the game defined as follows:

- $M_{A \parallel B} = M_A + M_B + (M_A^O \times M_B^O) + (M_A^P \times M_B^P)$
- $\lambda_{A \parallel B} = [\lambda_A, \lambda_B, a \mapsto O, a \mapsto P]$, where $a \mapsto O$ (a \mapsto P) denotes the $O$-constant (P-constant) function.
- $P_{A \parallel B} \subseteq M_{A \parallel B}^O$ is the set of plays, $s$, which satisfy the following condition:
  - $s \upharpoonright A \approx_A s' \upharpoonright A, s \upharpoonright B \approx_B s' \upharpoonright B, (\forall i)[(s_i \in M_A \iff s'_i \in M_B)]$

As for parallel tensor, also on parallel linear implication moving on both components at the same time is allowed.

**Definition 7 (Parallel Linear Implication).** Let $A = (M_A, \lambda_A, P_A, \approx_A)$ and $B = (M_B, \lambda_B, P_B, \approx_B)$ be AJM-games, the parallel linear implication game $A \rightarrow_{\parallel} B$ is defined as follows:

- $M_{A \rightarrow_{\parallel} B} = M_A + M_B + (M_A^O \times M_B^O) + (M_A^P \times M_B^P)$
- $\lambda_{A \rightarrow_{\parallel} B} = [\lambda_A, \lambda_B, a \mapsto P, a \mapsto O]$
- $P_{A \rightarrow_{\parallel} B} \subseteq M_{A \rightarrow_{\parallel} B}^O$ is the set of plays, $s$, which satisfy the following condition:
  - $s \upharpoonright A \approx_A s' \upharpoonright A, s \upharpoonright B \approx_B s' \upharpoonright B, (\forall i)[(s_i \in M_A \iff s'_i \in M_B)]$

The above notions of parallel tensor and linear implication give rise to monoidal closed categories $\mathcal{G}^\parallel$ and $\mathcal{G}_{hf}^\parallel$ where objects are AJM-games and morphisms between games $A$ and $B$ are equivalence classes of (history-free) strategies $\sigma : A \rightarrow_{\parallel} B$. Composition is defined similarly as for multitags. We omit the details. The co-kræishi construction can be straightforwardly carried out on the categories $\mathcal{G}^\parallel$ and $\mathcal{G}_{hf}^\parallel$ getting CCC’s $K_{1_{\parallel}}(\mathcal{G}^\parallel)$ and $K_{1_{\parallel}}(\mathcal{G}_{hf}^\parallel)$.
Relating Games to Multigames. The category $G^\circ \ (G^\circ_{h_f})$ of games with parallel tensor product turns out to be equivalent to the category $\mathcal{M} (\mathcal{M}_{h_f})$ of multigames, i.e. there exist functors $F : G^\circ \to \mathcal{M}$ and $G : \mathcal{M} \to G^\circ$, and natural isomorphisms $\eta : G \circ F \to \text{Id}_{G^\circ}$ and $\eta' : \text{Id}_{\mathcal{M}} \to F \circ G$.

Namely, given a multigame $A = (M_A, \lambda_A, P_A, \approx_A)$, in Section 1, we have seen how this induces an AJM-game $A_g = (M_A, \lambda_A, P_A, \approx_A)$, where the moves are the set of multimoves on the multigame $A$. Vice versa, given an AJM-game, $A = (M_A, \lambda_A, P_A, \approx_A)$, one build a multigame $A_m = (M_A, \lambda_A, P_A, \approx_A)$, where $P_A$ denotes the set of plays obtained from the plays in $P_A$ by replacing each move instance $a$ by the singleton multimove $\{a\}$. Clearly, for any game $A$, $(A_g)_m$ is isomorphic to $A$, and for any multigame $A$, $(A_m)_g$ is also isomorphic to $A$.

This allows us to define the object part of functors $F : G^\circ \to \mathcal{M}$ and $G : \mathcal{M} \to G^\circ$. Notice that $F$ and $G$ preserve tensor product on objects, up-to isomorphism.

Functors $F$ and $G$ can be extended to strategies as follows.

For any strategy on multigames $\sigma : A \multimap B$, we can associate a strategy $\sigma_g : A_g \multimap B_g$, where the plays of $\sigma_g$ are obtained from the plays of $\sigma$ by splitting each multimove of $\sigma$ containing atomic moves both in $A$ and in $B$ into a pair of moves on $A$ and $B$, respectively. Vice versa, any strategy on games $\sigma : A \multimap B$ induces a strategy on multigames $\sigma_m : A_m \multimap B_m$, whose plays are obtained from the plays of $\sigma$ by transforming each move instance of $A$ or $B$ into a singleton multimove, and each pair of moves $(a,b)$ as the multimove $\{a,b\}$.

Summarizing, we can define functors $F : G^\circ \to \mathcal{M}$ and $G : \mathcal{M} \to G^\circ$ by: for any AJM-game $A$, $FA = A_m$, for any strategy $\sigma : A \multimap B$, $F(\sigma) = [\sigma_m]$, for any multigame $A$, $GA = A_g$, for any strategy $\sigma : A \multimap B$, $G(\sigma) = [\sigma_g]$.

The functors $F : G^\circ \to \mathcal{M}$ and $G : \mathcal{M} \to G^\circ$ can be shown to be monoidal, and to give an equivalence between the categories $G^\circ$ and $\mathcal{M}$.

An analogous result can be proved for the categories $G^\circ_{h_f}$ and $\mathcal{M}_{h_f}$.

4 A Game Model of Unary PCF with Parallel Or

In this section, we define a game model of unary PCF with parallel or in the category of games $K_t(G^\circ_{h_f})$.

We recall that unary PCF with parallel or is a typed $\lambda$-calculus with a ground type $o$, two ground constants, $\bot, \top$, a sequential composition constant $\land : o \to o \to o$, and a parallel or constant $\lor : o \to o \to o$. Sequential composition examines the two arguments sequentially: if the first argument is $\bot$, then it returns $\bot$, otherwise it returns the second argument. Parallel or examines its arguments in parallel, and it returns $\bot$ only if both are $\bot$. The formal definition of unary PCF is the following.

The class $\text{SimType}$ of simple types over a ground type $o$ is defined by:

$$\text{SimType} \ni A ::= o \mid A \to A.$$  

Raw terms are defined as follows:
\[ A \ni M ::= \bot | T | \land | \lor | x | \lambda x : A . M | MM \]
where \( \bot, T, \land, \lor \) are constants, and \( x \in \text{Var} \).

Well-typed terms are terms typable in typing judgements of the form \( \Gamma \vdash M : A \), where \( \Gamma \) is a type environment, i.e. a finite set \( x_1 : A_1, \ldots, x_k : A_k \).

The rules for deriving typing judgements are the following:

\[
\begin{align*}
\Gamma \vdash \bot : o & \quad \Gamma \vdash T : o & \quad \Gamma \vdash \land : o \to o \to o & \quad \Gamma \vdash \lor : o \to o \to o \\
\Gamma, x : A \vdash x : A & \quad \Gamma \vdash \lambda x : A . M : A \to B & \quad \Gamma \vdash M : A \to B & \quad \Gamma \vdash N : A
\end{align*}
\]

The conversion relation between well-typed terms is the least relation generated by the following rules together with the rules for congruence closure (which we omit):

\[
\begin{align*}
\Gamma \vdash (\lambda x : A . M) N = M[N/x] : B, & \text{ where } \Gamma, x : A \vdash M : B, \text{ and } \Gamma \vdash N : A \\
\Gamma \vdash \land \top M = \land M \top = M : o & \text{ and } \Gamma \vdash \land \bot M = \land M \bot = \bot : o, \text{ where } \Gamma \vdash M : o \\
\Gamma \vdash \lor \top M = \lor M \top = \top : o & \text{ and } \Gamma \vdash \lor \bot M = \lor M \bot = M : o, \text{ where } \Gamma \vdash M : o.
\end{align*}
\]

The Game Model. In the cartesian closed category \( K_{\bot} \left( G^\bot \right) \), simple types are interpreted by the hierarchy of games over the following Sierpinski game:

**Definition 8 (Sierpinski Game).** The game \( \mathcal{O} \) is defined as follows:

\[
\begin{align*}
- M_\mathcal{O} & = \{ q, a \} \\
- \lambda_\mathcal{O}(q) & = Oq \quad \lambda_\mathcal{O}(a) = Pa \\
- P_\mathcal{O} & = \{ \epsilon, q, qa \} \\
- \approx_\mathcal{O} & = id_{P_\mathcal{O}}
\end{align*}
\]

Parallel or \( \lor \) is interpreted by the non-sequential strategy on the game \(!_\mathcal{O} \to_\mathcal{O} !_\mathcal{O} \to_\mathcal{O} \mathcal{O} \), where Opponent opens in the right-hand \( O \)-component, and Player answers with a pair of moves asking both arguments in parallel; then if Opponent answers in at least one argument \( (i.e. \text{ at least one argument is different from } \bot) \), Player provides the final answer in the right-hand component. This strategy is obtained as the even-prefix closure of the plays in the picture.

Using standard methods, one can prove that the theory induced by the game model is the theory of \( \beta\eta \)-normal forms. Moreover, in view of the results in [Lai03], the extensional quotient of the above game model is universal for the observational equivalence of unary PCF (see [Lai03] for more details).
References


Space Complexity in Algebraic Proof Systems

Ilario Bonacina\(^1\) and Nicola Galesi\(^2\)

\(^1\) University of Rome “La Sapienza”, Rome, Italy
bonacina@di.uniroma1.it
\(^2\) University of Rome “La Sapienza”, Rome, Italy
galesi@di.uniroma1.it

Abstract. In [7] we described a combinatorial framework for proving space lower bounds in algebraic proof systems as Polynomial Calculus (PC) and Polynomial Calculus with Resolution (PCR); here we present some aspects of that paper. The framework we build unifies in a clear way under a common language all proofs of space lower bounds known so far for PC/PCR (for formulas such as \(CT_n\), \(PHP_n\), \(BPHP_n\), \(XOR-PHP_n\)) and allowed us to solve to the open problem [1, 20] of proving space lower bounds in PC and PCR for the polynomial encoding of randomly chosen \(k\)-CNF formulas. Then, as proved for Resolution in [12], also in PC and in PCR refuting a random \(k\)-CNF over \(n\) variables requires high space measure of the order of \(\Omega(n)\) where \(n\) is the number of variables (our result holds for \(k \geq 4\)). Our method also applies to the Graph-\(PHP^m_n\), which is a \(PHP^m_n\) defined over a constant (left) degree bipartite expander graphs, and to other families of formulas.

Keywords: Proof Complexity, Space Complexity, Polynomial Calculus, Resolution, Random \(k\)-CNF Formulae

1 Introduction

We are going to briefly present some aspects of our paper [7]. This work belongs to the research field of Proof Complexity: a research field initiated by Cook and Reckhow [16] that studies the complexity of proving propositional tautologies in different logical propositional proof systems. The historical motivation for investigating the complexity of proofs is the P vs NP question. As observed in [16], one way of establishing \(\text{co-NP} \neq \text{NP}\), and hence \(P \neq \text{NP}\), would be to prove that there are no efficient proof systems. This is what is known as the Cook’s Program in Proof Complexity. Among the most studied proof systems there are the logical systems of Resolution [10] and algebraic proof systems as Polynomial Calculus (PC) [15] or Polynomial Calculus with Resolution (PCR) [1].

As remarked by Razborov [21], proof complexity plays the same role in the field of feasible proofs of the role played by the Boolean Circuits/Turing Machines in the field of efficient computations. Hence proof size in Proof Complexity should be viewed as Circuit-Size/Running-Time in Circuit Complexity. It is then no surprise that a notion of Proof Space Measure was introduced also for proof systems [1], as for efficient computations we consider memory occupation as a measure of efficiency. Polynomial Calculus, though being a very well-studied proof system when considering the size and degree complexity of proofs [15, 8, 13, 4], is still at the beginning of the investigation of the space measure [1, 20, 7].

Proceedings of ICTCS 2013
Polynomial Calculus (PC) is a proof system having its algebraic base on the Gröbner Basis Algorithm [15] and PC is surely one of the proof systems that have some hope of producing new insights into the field of SAT-solvers. At the moment there are PC-based solvers, such as PolyBoRi [6], but in general they seem to be an order of magnitude slower than state-of-the-art solvers based on Resolution. Our work contributes to better understand theoretically the space measure in PC/PCR and we think that our discrete combinatorial characterization of the space in algebraic proof systems might open the way to better understand how to encode polynomials and devise algorithms (theorem provers or SAT-solvers) working on polynomials but using discrete combinatorial concepts.

Our work is related also with Finite Model Theory. Our main definition, the notion of \( k \)-extendibility, can be viewed as a first step towards a 2-players game characterization for algebraic proof systems. This property characterizes the winning strategies for an adversary as a class of combinatorial objects. Our main definition should also be compared with the definition given by Atserias for Resolution [5] and the one, given by Esteban, et al. in [18] of winning strategies for getting space lower bounds in \( \text{Res}(k) \), that is a Resolution system on \( k \)-DNF. As was done for random \( k \)-CNF for DATALOG by Asterias [5], our game characterization of boolean reasoning with polynomials can suggest non-expressibility results in stronger logics appropriate to this kind of reasoning.

2 Outline of the results

In [7] we answered to the open problem of proving an \( \Omega(n) \) space lower bound for random \( k \)-CNF in \( n \) variables in PC/PCR [1, 20]. We have done this by developing a new method for proving space lower bounds in PC/PCR that allows us to use semantic arguments, similar to the ones used in Resolution. It is known [3] that in Resolution “space is lower bounded by the width” and hence width lower bounds imply space lower bounds. This connection can be obtained by a characterization of the width and the space through winning strategies of the adversary (Duplicator) in a Spoiler-Duplicator \( k \)-existential game. In the algebraic setting of PC/PCR we characterize (Definition of \( k \)-extendible family) how long an adversary (Duplicator) can answer to a player (Spoiler) downloading polynomials into the memory without falling into a contradiction. This idea is the same as the one used in Resolution both in the characterization of the space by Asterias and Dalmau [3] or by Esteban et al. in [18] where they independently introduced the notion of \( k \)-dynamical satisfiability to study space lower bounds in Resolution or \( \text{Res}(k) \).

Once we have this definition of \( k \)-extendibility then our Main Theorem basically says that a \( k \)-extendible family for an unsatisfiable CNF \( \varphi \) provide a \( k/4 \) space lower bound for the polynomial encoding of \( \varphi \) in the proof systems PC/PCR.

Then we can use this result to prove space lower bounds: we “simply” have to build suitable \( k \)-extendible families for the given family of formulas. Then, to prove an \( \Omega(n) \) space lower bound for random \( k \)-CNFs in \( n \) variables, we just have to exhibit an \( \Omega(n) \)-extendible family for random \( k \)-CNFs \( (k \geq 4) \). We
do this generalizing to double matchings the Matching Game of Ben-Sasson and Galesi [12] (simplified in [5]). Moreover the same construction produce a new space lower bound for the Graph-PHP, which is a formula encoding a Pigeonhole Principle defined over an expander bipartite graph with constant left degree.

In [7] we easily re-obtain all the known space lower bound for PC/PCR known so far [1, 20] by producing fairly simple extendible families. This part of [7] will not be discussed here.

In what follows we sketch some ideas behind the proof of the Main Theorem (see below) and its application to prove a space lower bound for random $k$-CNFs.

Before stating the Main Theorem, let’s just recall informally what is the notion of space we use: we look at proofs in PC/PCR as sequences of memory configurations, i.e. sets of polynomials we simultaneously keep in memory, and we can go from one memory configuration to the other according to the inference rules of the calculus. The space of each memory configuration is the number of distinct monomials in it, and the space of a proof $\pi$ is the maximum space of a memory configuration appearing in $\pi$.

**Theorem 1 (Main Theorem).** Let $P$ be an unsatisfiable set of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ and $I$ a proper ideal in that ring. Suppose that there exists a non-empty $k$-extendible family of admissible configurations $F$ for $P$ with respect to $I$. Then $Sp_{PCR}(\varphi \vdash 1) \geq k/4$, i.e. the minimum space to refute $P$ in $PCR$ is at least $k/4$.

Informally speaking, the proof of the Main Theorem goes as follows: suppose that we have a $k$-extendible family for $P$ and, by contradiction, that there exists a refutation $\pi$ of $P$ in PC/PCR where the space used is $< k/4$. The proof $\pi$ is a sequence of memory configurations $(C_0, \ldots, C_s)$. We prove by induction on the index $i$ of the memory configuration that there exists an object $(Q_i, H_i) \in F$ and a set of assignments $Z_i \subseteq H_i$ such that $Z_i$ satisfy $C_i$ and the set $Z_i$ has some good properties to make the induction work.

In the case of an axiom download we maintain the induction properties by $k$-extendibility.

In the case of an erasure (or an inference) of some polynomial from the memory we are able to maintain the inductive property by a Localy Lemma (basically the same of [1]) and by properties of the $k$-extendible families.

So we obtain the contradiction that the final (unsatisfiable) memory configuration is satisfiable. For more detail of the definitions, and for the statement of the Localy Lemma, see [7].

### 2.1 An $\Omega(n)$-extendible family for random $k$-CNFs

It is well-known that in circuit complexity simple counting arguments show that a random function is hard to compute. In studying the complexity of a given proof system it is natural to ask what is the proof complexity of a tautology taken at random. However we do not have a definition of what is a random tautology. Still, in some cases, if we restrict our attention only
to certain kinds of tautologies we can deduce information on their random behavior. An easy calculation shows that for a high enough constant, with high probability a random $k$-CNF formula is unsatisfiable. Let us introduce the definition of a random CNF.

Let $n, m$, and $k$ be positive natural numbers and let $X = \{x_1, \ldots, x_n\}$ be a set of variables. Let $F(n, m, k)$ be the set of all $k$-CNF formulas on $X$ with exactly $m$ clauses each defined on $k$ literals on distinct variables. Alternatively, $F(n, m, k)$ can be described as the result of repeating $m$ times independently the following experiment: choose exactly $k$ variables from $X$, and negate each variable independently with probability $1/2$. The ratio $m/n$ is denoted by $\Delta$, and is called the clause density. Usually, $\Delta$ is fixed to a constant and therefore is determined by $n$. We are interested in studying the asymptotic properties of a randomly chosen formula $F \sim F(n, m, k)$ as $n$ approaches infinity. It is well known that when the clause density exceeds a certain constant $\theta_k$ that only depends on $k$, a randomly chosen formula is almost surely unsatisfiable. The question of the existence and value of a satisfiability threshold constant is an important open problem in combinatorics, and for more information on this subject see e.g. [2]. In this work we are interested only in the region in which $F$ is unsatisfiable with high probability, then we always consider fixed $\Delta$, then $F(n, m, k)$ can be made dependent only on $n$, $\Delta$ and $k$ and denoted as $F(n, \Delta, k)$.

The proof size of unsatisfiable random CNFs has been widely studied in proof complexity. Chvatal and Szemeredi in their seminal paper [17] showed that with high probability, any random $k$-CNF over $n$ variables and $\Delta n$ clauses for $\Delta = O(1)$, requires exponentially long Resolution proofs to be refuted. The importance of their work was in showing that in fact Resolution is a very weak proof system, because in some sense almost all unsatisfiable $k$-CNF require exponential size proofs to be refuted. Their lower bound was later improved and simplified by Beame and Pitassi in [11], and finally improved up to a ratio by Beame, Karp, Pitassi and Saks in [9], and reformulated in terms of a general technique based on the width by Ben-Sasson and Wigderson in [14].

The degree complexity in PC/PCR of refuting random $k$-CNF was established by Ben-Sasson and Impagliazzo in [13] for polynomials over fields with characteristic different from 2 and then for any field by Alekhnovich and Razborov in [4]. These works proved that with high probability refuting (unsatisfiable) random $k$-CNF in PC/PCR requires linear degree.

Concerning space it is known that in Resolution random $k$-CNF for $\Delta = O(1)$ requires space $\Omega(n)$. This was a result of Ben-Sasson and Galesi answering to a question posed in [1, 19]. Regarding PC/PCR both [1, 20] posed the question of proving linear lower bounds for the space of refuting random $k$-CNF in these proof systems. In [7] we answer to this question, for the case $k \geq 4$. The precise statement we gave is the following.

**Theorem 2.** Let $k \geq 4$ be any integer, $\epsilon > 0$ any constant and $\Delta \geq 1$. Let $\varphi \sim F(n, \Delta, k)$. There exists a constant $c = c_{k, \Delta, \epsilon}$, $c \geq 1$, such that with high probability

$$Sp_{PCR}(\varphi \vdash_{sem} 1) \geq \frac{n}{4c}.$$
To obtain this result, thanks to the Main Theorem, it is sufficient to build a suitable extendible family for random $k$-CNFs. Here we want to sketch some ideas behind that construction.

First we use a double matching game on bipartite graphs to identify at each step of the proofs what are the variables involved in a possible “good” set of assignments that implies the memory configuration. This is not new, since Ben-Sasson and Galesi in [12] where doing exactly this for Resolution. But instead of multiple matchings they had simple matchings and instead of our family of assignments they have single assignments. This game is played on bipartite graphs $G = (U \cup V, E)$ and has two players, Challenger and Extender. Challenger asks at each step of the game a vertex in $U$ and Extender has to respond with a double matching in the graph $G$ involving that vertex and coherent with his previous answers. Challenger can renounce to some vertex, and Extender has to respond accordingly. If Extender can’t respond to some move then Challenger wins.

Then we move from that matching game to a $\Omega(n)$-extendible family of assignments. This second part is a quite straightforward: it is the translation of a winning strategy for Extender into a $k$-extendible family that basically is a winning strategy for another kind of game closely related with the double matching game.

The winning strategy for Extender is based on dynamically maintaining a property, the $(r, s)$-double matching property.

**$(r, s)$-double matching property:** Let $r \leq s$ two integers and $G = (U \cup V, E)$ a bipartite graph. Given two subsets $A \subseteq U$ and $B \subseteq V$ we say that $(G, A, B)$ has the $(r, s)$-double matching property if $|A| \leq r$, $|B| = 2|A|$ and for every $C \subseteq U \setminus A$, if $|C| = s - |A|$ then there exists a 2-matching of $C$ into $V \setminus B$.

This property will allow us to identify dynamically for each memory configuration a set of initial clauses of the random $k$-CNF we are satisfying (in addition to the actual memory configuration we satisfy). The idea behind this definition is to focus on the extension of an existing double matching, i.e. how in the double matching game the player Extender can continue the game, hiding all the details on how he and Challenger arrived to that configuration of the game but focusing only on the current configuration. In the previous definition the sets $A$ and $B$ play the role of the actual configuration of the game: we are not interested in how is constructed the double matching inside the sets $A$ and $B$ to extend. The aim of that definition is to guarantee Extender that no matters how he and Challenger arrived to a configuration, he can always make his move. Clearly this is a game very close to the Matching Game developed in [12, 5].

The only important missing detail of that construction is that we are able to maintain that $(r, s)$-double matching property provided that the graph where the game is played has good expansion properties. And this is the case for the graph we build upon random $k$-CNFs. The expansion property we use to maintain that property is stronger than the ones usually used (see [12, 5])
because we need to provide the existence of double matchings. The precise notion of expansion we use is the following.

\((s, \epsilon)\)-BIPARTITE EXPANSION: Let \(G = (U \cup V, E)\) a bipartite graph. We say that \(G\) is an \((s, \epsilon)\)-bipartite expander if

\[\forall A \subseteq U, \ |A| \leq s \implies |N_G(A)| \geq (2 + \epsilon)|A| .\]

Notice that the standard expansion property used for example in Resolution to achieve lower bounds for random \(k\)-CNFs has a \((1 + \epsilon)\) instead \((2 + \epsilon)\) in the previous definition. Due to this stronger requirement on the expansion we obtain our lower bound for random \(k\)-CNF for \(k \geq 4\).

Putting all this together we are able to build an \(\Omega(n)\)-extendible family for random \(k\)-CNFs in \(n\) variables, hence an \(\Omega(n)\) space lower bound for that families of formulas due to the Main Theorem.

3 Some open problems

We saw two natural complexity measures in algebraic proof systems: the degree and the space. It is not known if they are related and how. It is conjectured that “space is lower bounded by the degree” so, if we believe in this conjecture, it is reasonable to expect that some technique could provide both degree and space lower bounds. At the moment this is not the case: the techniques used to prove degree lower bounds \([15, 8, 13, 4]\) and space lower bounds \([1, 20, 7]\) are way different. Our paper can be seen as a preliminary first step in unifying these techniques.

Anyway we have still some basic open problems: for example it is not known any space lower bound for formulas in which there is a lot of “unit propagation”, for example for some versions of the Pigeonhole Principle. The lower bound provided by our Main Theorem in such cases becomes trivial. As already said, the investigation about space complexity in the algebraic setting is still at its beginning.

References

Covering Pairs in Directed Acyclic Graphs

Niko Beerenwinkel\textsuperscript{1}, Stefano Beretta\textsuperscript{2}, Paola Bonizzoni\textsuperscript{2}, Riccardo Dondi\textsuperscript{3},
and Yuri Pirola\textsuperscript{2}

\textsuperscript{1} Dept. of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland, 
niko.beerenwinkel@bsse.ethz.ch
\textsuperscript{2} DISCo, Univ. degli Studi di Milano-Bicocca, Milan, Italy, 
\{beretta,bonizzoni,pirola\}@disco.unimib.it
\textsuperscript{3} Dip. di Scienze Umane e Sociali, Univ. degli Studi di Bergamo, Bergamo, Italy, 
riccardo.dondi@unibg.it

\textbf{Abstract.} In this paper we study a constrained version of the Minimum Path Cover problem motivated by applications in software testing and in the analysis of Next-Generation Sequencing data in bioinformatics. Given a directed acyclic graph and a set of pairs of vertices, the problem asks for a path that contains the maximum number of pairs. We investigate the complexity of the problem and we show that it is not only NP-hard, but also W[1]-hard when the parameter is the size of the solution, i.e. the number of covered pairs. On the positive side, we give a fixed-parameter algorithm when the parameter is the number of overlapping pairs, a natural parameter in some applications of the problem.

1 Introduction

The Minimum Path Cover (MinPC) problem is a well-known problem in graph theory. Given a directed acyclic graph (DAG), the problem asks for a minimum-cardinality set of paths $\Pi$ such that each vertex of $G$ belongs to at least one path of $\Pi$. The problem can be solved in polynomial time with an algorithm based on the well-known Dilworth’s theorem for partially ordered sets [2], which allows to relate the size of a minimum path cover to that of a maximum matching in a bipartite graph obtained from the input DAG.

The Minimum Path Cover problem has important applications in several fields ranging from bioinformatics [5, 8, 1] to software testing [7]. In bioinformatics, for example, the Minimum Path Cover problem is used in the reconstruction of the set of different transcripts originated from the same gene starting from a large set of their short fragments (called short reads) [8, 1]. In particular, each fragment is represented by a single vertex and two vertices are connected if the alignments of the corresponding reads on the genomic sequence overlap. Clearly, paths on such a graph represent putative transcripts and a minimum-cardinality set of paths “covering” all the vertices represents a set of transcripts which the observed reads are likely originated from. Another application of the minimum path cover problem in bioinformatics is that of assessing the structure of a viral population by reconstructing the (highly similar) genetic sequences of its individuals starting from a set of short reads [5].

The Minimum Path Cover problem has been applied in the past to software testing. Each procedure to be tested is modeled by a graph where vertices correspond to single instructions and two vertices are connected if the
corresponding instructions are executed sequentially. The test of the procedure should test each instruction at least once, hence a minimum path cover of the graph represents a minimum set of execution flows that allows to test all the instructions. Clearly, not all the execution flows are possible. Hence, Ntafos and Hakimi [7] proposed a constrained variant of the minimum path cover problem, with the introduction of a set of required pairs, meaning that a feasible solution must include a path containing both vertices of the pair.

The notion of required pairs is also important for the applications in bioinformatics we mentioned, as short reads are often sequenced in pairs (paired-end reads) and must align to a single genetic sequence at a certain (fixed) distance. This kind of data provides valuable information which, in principle, could greatly improve the accuracy of the reconstruction. However, it is often used only to filter out reconstructed sequences that do not meet such constraints, instead of exploiting it directly in the reconstruction process.

Since the general problem of covering a given set of required pairs is NP-complete [7], a natural greedy approach to the problem adds at each step a path that covers the maximum number of required pairs. This leads to a natural combinatorial problem, the Maximum Required Pairs with Single Path (MaxRPSP) problem, that, given a directed acyclic graph and a set of required pairs, asks for a path that covers the maximum number of required pairs. In this paper we focus on the fixed-parameter tractability of this problem. After giving some preliminary definitions, we show in Section 3 that MaxRPSP is $W[1]$-hard when the parameter is the size of the solution, i.e. the number of covered required pairs. We complement this result by giving in Section 4 a fixed-parameter algorithm where the parameter is the maximum number of overlapping required pairs. The maximum number of overlapping required pairs (formally defined in Section 4) is a natural parameter to study the complexity of the problem, as in some applications (especially in bioinformatics) can be bounded by a low value.

2 Preliminaries

In this section we introduce the basic notions used in the rest of the paper. We consider both directed and undirected graphs.

We denote an undirected graph as $G = (V, E)$ where $V$ is the set of vertices and $E$ is the set of (undirected) edges, and a directed graph as $D = (N, A)$ where $N$ is the set of vertices and $A$ is the set of (directed) arcs. We denote an edge of $G = (V, E)$ as $\{v, u\} \in E$ where $v, u \in V$. Moreover, we denote an arc of $D = (N, A)$ as $(v, u) \in A$ where $v, u \in N$.

Given a directed graph $D = (N, A)$, a path $\pi$ from vertex $v$ to vertex $u$, referred to as $vu$-path, is a sequence of vertices $\langle v_1, \ldots, v_n \rangle$ such that $(v_i, v_{i+1}) \in A$, $v = v_1$ and $u = v_n$. We say that a vertex $v$ belongs to a path $\pi$ if $v \in \pi$. Given a set of $N' \subseteq N$ of vertices, a path $\pi$ covers $N'$ if $N' \subseteq \pi$.

Now, we are able to give the definition of the combinatorial problem we are interested in.
Problem 1. Maximum Required Pairs with Single Path (MaxRPSP)

**Input:** a directed acyclic graph \( D = (N, A) \), a source \( s \in N \), a sink \( t \in N \) and a set \( R = \{ [v_x, v_y] \mid v_x, v_y \in N, v_x \neq v_y \} \) of required pairs.

**Output:** an \( st \)-path \( \pi \) that covers a set \( R' = \{ [v_x, v_y] \mid v_x, v_y \in \pi \} \subseteq R \) of maximum cardinality.

MaxRPSP is related to a combinatorial problem which has been studied in the context of program testing [7], where it is shown to be NP-hard. More precisely, given a directed acyclic graph \( D = (N, A) \), a source \( s \in N \), a sink \( t \in N \) and a set \( R = \{ [v_x, v_y] \mid v_x, v_y \in N, v_x \neq v_y \} \) of required pairs, the Minimum Required Pairs Cover problem asks for a minimum cardinality set \( \Pi = \{ \pi_1, \ldots, \pi_n \} \) of directed \( st \)-paths such that every required pair \([v_x, v_y] \in R\) belongs to at least one \( st \)-path \( \pi_j \in \Pi \), i.e. \( v_x, v_y \in \pi_j \).

3 MaxRPSP Parameterized by the Maximum Number of Covered Required Pairs

In this section, we investigate the parameterized complexity of MaxRPSP when parameterized by the size of the solution, that is the maximum number of required pairs covered by a single path, and we prove that the problem is \( W[1]\)-hard. Notice that this result implies the NP-hardness of MaxRPSP, since a \( W[1]\)-hard problem is also an NP-hard problem [3, 6].

We prove the \( W[1]\)-hardness of MaxRPSP via a parameterized reduction from the Maximum Clique (MaxClique) problem (for details on parameterized reduction, we refer the reader to [3, 6]). We recall that, given an undirected graph \( G = (V, E) \), MaxClique asks for a subset \( C \subseteq V \) of maximum size that induces a clique in \( G \).

Here, we consider the two decision versions of the MaxClique and MaxRPSP, \( h \)-Clique and \( k \)-RPSP respectively, parameterized by the sizes of the respective solutions. For example, given an undirected graph \( G = (V, E) \), the \( h \)-Clique problem consists of deciding if there exists a clique \( C \subseteq V \) of size \( h \), i.e. \( |C| = h \). We recall that \( h \)-Clique is known to be \( W[1]\)-hard [4].

First, we start by showing how to construct an instance of \( k \)-RPSP starting from an instance of \( h \)-Clique. Given an (undirected) graph \( G = (V, E) \) with \( n \) vertices \( v_1, \ldots, v_n \), we construct the associated directed acyclic graph \( D = (N, A) \) as follows. The set \( N \) of vertices is defined as follows:

\[
N = \{ v_i^z \mid v_i \in V, z \in 1, \ldots, h \} \cup \{ s, t \}
\]

Informally, \( N \) consists of two distinguished vertices \( s, t \) and by \( h \) copies \( v_i^1, \ldots, v_i^h \) of every vertex \( v_i \) of \( G \).

The set of arcs \( A \) is defined as follows:

\[
A = \{ (v_i^z, v_j^{z+1}) \mid \{ v_i, v_j \} \in E, 1 \leq z \leq h - 1 \} \cup \{ (s, v_i^1), (v_i^h, t) \mid v_i \in V \}
\]

Informally, we connect the consecutive copies associated with vertices adjacent in \( G \), the source vertex \( s \) to all the vertices \( v_i^1 \), with \( 1 \leq i \leq n \), and all the vertices \( v_i^h \), with \( 1 \leq i < n \), to the sink vertex \( t \).
Fig. 1. Example of the directed acyclic graph $D = (N, A)$ associated with an instance $G = (V, E)$ of the $h$-Clique problem. Each gray box highlights an independent set $I_z$ composed of one copy of the vertices in $V$. Edges $(v_i^x, v_j^y)$, $(v_i^x, v_j^{y+1})$, and $(v_i^n, v_j^{n+1})$ are some of the directed edges in $A$ associated with edges $\{v_i, v_j\}, \{v_i, v_j\}, \{v_i, v_n\} \in E$.

The set $R$ of required pairs is defined as follows:

$$R = \{[v_i^x, v_j^y] \mid \{v_i, v_j\} \in E, 1 \leq x < y \leq h\}$$

Informally, for each edge $\{v_i, v_j\}$ of $G$ there is a required pair $[v_i^x, v_j^y]$, $1 \leq x < y \leq h$, between every two different copies associated with $v_i, v_j$.

By construction, the vertices in $N$ (except for $s$ and $t$) are partitioned into $h$ independent sets $I_z = \{v_i^z \mid 1 \leq i \leq n\}$, with $1 \leq z \leq h$, each one containing a copy of every vertex of $V$. Moreover, the arcs of $A$ only connect two vertices of consecutive subsets $I_z$ and $I_{z+1}$, with $1 \leq z < h - 1$. Figure 1 presents an example of the directed graph $D$ associated with an undirected graph $G$.

Now, we are able to prove the main properties of the reduction.

Lemma 1. Let $G = (V, E)$ be an undirected graph and $(D = (N, A), R)$ be the associated instance of $k$-RPSP. Then: (1) starting from an $h$-clique in $G$ we can compute in polynomial time an st-path $\pi$ in $D$ that covers $\binom{h}{2}$ required pairs of $R$; (2) starting from an st-path $\pi$ in $D$ that covers $\binom{h}{2}$ required pairs we can compute in polynomial time an h-clique in $G$.

Proof. (1) Starting from an $h$-clique $C$ in $G$ we show how to compute a path $\pi$ in $D$ that covers $\binom{h}{2}$ required pairs of $R$. Let $C = \{v_{i_1}, \ldots, v_{i_h}\}$ be a clique of $G$ and let $\langle v_{i_1}, \ldots, v_{i_h} \rangle$ be an arbitrary ordering of $C$. Let $\pi_C = (s, v_{i_1}, \ldots, v_{i_h}, t)$ be a sequence of vertices obtained by selecting the vertex $v_{i_z}^z$ for each independent set $I_z$, $z \in 1, \ldots, h$ (in addition to the vertices $s$ and $t$). Since $C$ is a clique of $G$, by construction of $D$, every pair of vertices $\langle v_{i_z}^z, v_{i_{z+1}}^{z+1} \rangle$ is connected by

---

Proceedings of ICTCS 2013
an arc, hence \( \pi_C \) is an st-path of \( D \). Moreover, the path \( \pi_C \) covers exactly \( \binom{h}{2} \) required pairs of \( R \) because, by construction of \( R \), there exists a pair between any two copies of vertices adjacent in \( G \). More precisely, since the clique \( C \) has all the possible edges among its \( h \) vertices, the number of required pairs covered by the path \( \pi_C \) is \( \binom{h}{2} \).

(2) Let \( \pi \) be an st-path in \( D \) that covers a set \( R' \subseteq R \) of \( \binom{h}{2} \) required pairs, we show how to compute in polynomial time an \( h \)-clique \( C \) in \( G \). Notice that, by construction of \( D \), the st-path \( \pi \) must contain exactly one vertex \( v_i^x \), \( 1 \leq i \leq n \) and \( 1 \leq x \leq h \), for each independent set \( I_x \) of \( D \). By construction of set \( R \), each vertex \( v_i^x \) of \( \pi \) appears in at most \( h - 1 \) required pairs of \( R' \). Hence, the total number of required pairs covered by path \( \pi \), which contains exactly \( h \) inner vertices \( v_i^x \), is at most \( \frac{h(h-1)}{2} = \binom{h}{2} \). Let \( C \) be the set \( \{ v_i^x \mid v_i^x \in \pi \setminus \{s,t\} \} \). We claim that \( C \) is an \( h \)-clique. First, we prove that \( C \) contains \( h \) vertices. Suppose that this is not the case, hence \( C \) consists of less than \( h \) vertices. Then, there exist two vertices \( v_i^x \) and \( v_i^y \) of \( \pi \) that correspond to the same vertex \( v_i \) of \( C \), that is \( i' = i'' = i \). Since \( (v_i^x, v_i^y) \) is not a required pair, i.e. \( [v_i^x, v_i^y] \notin R \), it follows that each of \( v_i^x \), \( v_i^y \) appears in at most \( h - 2 \) required pairs of \( R' \). As a consequence, the total number of required pairs covered by the path \( \pi \) is strictly less than \( \binom{h}{2} \), violating the initial hypothesis that \( \pi \) covers \( \binom{h}{2} \) required pairs of \( R \). Hence \( C \) contains \( h \) vertices. As all the internal vertices of \( \pi \) (i.e., all its vertices but \( s \) and \( t \)) represent distinct vertices of \( G \), all the required pairs covered by \( \pi \) represent distinct edges of \( G \). The only undirected graph with \( h \) vertices and \( \binom{h}{2} \) edges is the complete graph, hence \( C \) is an \( h \)-clique of \( G \).

The W[1]-hardness of \( k \)-RPSP follows easily from Lemma 1.

**Theorem 1.** \( k \)-RPSP is W[1]-hard when parameterized by the number of required pairs covered by an st-path.

**Proof.** The result follows from Lemma 1 and from the W[1]-hardness of \( h \)-Clique when parameterized by \( h \) [4].

\( \square \)

## 4 MaxRPSP Parameterized by the Maximum Number of Overlapping Pairs

In this section we propose a parameterized algorithm for the MaxRPSP problem, where the parameter is the number of overlapping required pairs. First, we introduce some preliminary definitions. Given an instance of the MaxRPSP problem \( \langle D = (N, A), R \rangle \), then for ease of exposition we will represent the required pairs of \( R \) as \([v_i^1, v_i^2]\) and, whenever no confusion arises, we will refer to that required pair as i-pair. Moreover, we say that two required pairs \([v_i^1, v_i^2], [v_j^1, v_j^2]\) \( R \) overlap, if there exists a path \( \pi \) in \( D \) such that the four vertices appear in \( \pi \) in one of the following orders (assuming that vertex \( v_i^1 \) appears before \( v_i^2 \) in \( \pi \)):

- \( \langle v_i^1, v_i^2, v_j^1, v_j^2 \rangle \) (alternated)
- \( \langle v_i^1, v_j^1, v_i^2, v_j^2 \rangle \) (nested)

\( \square \)
Given a required pair \([v_i^1, v_i^2]\), we define the set of vertices:

\[
OP([v_i^1, v_i^2]) = \{v_j \mid v_j \text{ appears before } v_i^2 \text{ and } j\text{-pair overlaps with } i\text{-pair}\} \cup \{v_i^1\}
\]

We also define a required pair \([v_i^1, v_i^2]\) as maximal if there is no required pair
\([v_j^1, v_j^2]\) such that \([v_i^1, v_i^2]\) is nested in \([v_j^1, v_j^2]\). Property 1 on maximal required pairs directly follows from the previous definition.

**Property 1.** Given two maximal required pairs \([v_i^1, v_i^2], [v_j^1, v_j^2] \in R\) in a path \(\pi\) of \(D\), then exactly one of the following cases holds (assuming that vertex \(v_i^1\) appears before \(v_j^1\) in \(\pi\)):

- alternated: \([v_i^1, v_j^1, v_i^2, v_j^2]\);
- not overlapped: \([v_i^1, v_i^2, v_j^1, v_j^2]\).

Now, we present the parameterized algorithm (where the parameter \(h\) is the maximum number of overlapped required pairs) for the MaxRPSP problem. The algorithm is based on dynamic programming. We assume that the required pairs are ordered and that according to this order if a maximal pair \([v_i^1, v_i^2]\) is before a maximal pair \([v_j^1, v_j^2]\) (that is \(i < j\)), then \(v_i^1\) is not reachable from \(v_j^1\) (i.e., either because there is a path from \(v_i^1\) to \(v_j^1\) or because there is no path between them) or \(v_i^2\) is not reachable from \(v_j^2\). Intuitively, we impose that the order of the required pairs is compatible with the topological order of the vertices.

Let \(P[v_i^2, S]\) denote the maximum number of required pairs covered by a path \(\pi\) ending at vertex \(v_i^2\) and such that the set \(S \subseteq OP([v_i^1, v_i^2])\) is contained in \(\pi\). Given two sets \(S\) and \(S'\) of vertices such that \(S \subseteq OP([v_i^1, v_i^2])\) and \(S' \subseteq OP([v_j^1, v_j^2])\), we say that \(S\) is in agreement with \(S'\) if \(S \cap (OP([v_i^1, v_i^2]) \cap OP([v_j^1, v_j^2])) = S' \cap (OP([v_i^1, v_i^2]) \cap OP([v_j^1, v_j^2]))\). Informally, \(S\) and \(S'\) must contain the same subset of vertices of \(OP([v_i^1, v_i^2]) \cap OP([v_j^1, v_j^2])\).

In the following we present the recurrence to compute \(P[v_i^2, S]\). For ease of exposition we only focus on vertices that appear as the second vertex of a required pair. In fact, paths that do not end in such vertices are not able to cover new required pairs. Furthermore, for simplicity, we consider the source of the graph \(s\) as the second vertex of a fictitious required pair (with index 0) which is not overlapped with any other required pair. Such a fictitious required pair does not contribute to the total number of required pairs covered by the path.

The recurrence is:

\[
P[v_i^2, S] = \max_{\{v_j^1, v_j^2\} \text{ is not nested in } [v_i^1, v_i^2] \text{ and } j \neq i; S' \text{ in agreement with } S; \text{ if a path from } v_j^1 \text{ to } v_j^2 \text{ containing all vertices in } S' \setminus S'} \{P[v_j^2, S'] + |OV([v_i^1, v_i^2], S \setminus S')|\}
\]

where \(OV([v_i^1, v_i^2], S \setminus S') = \{[v_h^1, v_h^2]\} | [v_i^1, v_i^2] \text{ is nested in } [v_h^1, v_h^2] \text{ and } v_h^1 \in S \wedge v_h^2 \in S \setminus S'\). Notice that each pair is assumed to be nested in itself.

The base case of the recurrence is \(P[s, \emptyset] = 0\).

The correctness of the recurrence derives from the following two lemmas.

---

**Proceedings of ICTCS 2013**
Lemma 2. If $P[v_i^2, S] = k$, then there exists a path $\pi$ in $D$ ending in $v_i^2$, such that every vertex in $S$ belongs to $\pi$ and the number of required pairs covered by $\pi$ is $k$.

Lemma 3. Let $\pi$ be a path in $D$ ending in $v_i^2$ and covering $k$ required pairs. Let $S$ be the set of all the vertices belonging to required pairs covered by $\pi$ and overlapping with $[v_i^1, v_i^2]$. Then $P[v_i^2, S] \geq k$.

Proof (of Lemma 2). We prove the lemma by induction on the index $i$. It is easy to see that the base case holds. Assume that the lemma holds for the indexes of value less than $i$, we prove that the lemma holds for $i$. Let $P[v_i^2, S] = k$. By Eq. (1), there exists a vertex $v_{j}^2$ with $j < i$, such that $P[v_{j}^2, S'] = k_1$ for some set $S'$ in agreement with $S$. Assume that $|\text{Ov}([v_i^1, v_i^2], S \setminus S')| = k_2$, with $k_1 + k_2 = k$. By induction hypothesis, since $P[v_i^2, S'] = k_1$, there exists a path $\pi'$ ending in $v_i^2$, containing every vertex in $S'$, and such that $\pi'$ covers $k_1$ required pairs. Furthermore, the $k_2$ covered required pairs have at least one vertex in $S \setminus S'$, hence the vertices of such required pairs belong to a path $\pi''$ which starts in $v_i^2$ and ends in $v_i^2$ (path $\pi''$ exists by hypothesis). But then, the path obtained by the concatenation of $\pi'$ and $\pi''$ covers $k_1 + k_2$ pairs.

Proof (of Lemma 3). We prove the lemma by induction on the index $i$. It is easy to see that the base case holds. Assume that the lemma holds for the indexes of value less than $i$, we prove that the lemma holds for $i$. Let $\pi$ be a path that covers $k$ required pairs and ends in $v_i^2$ and let $S$ be the set of vertices that belong to the required pairs covered by $\pi$ and overlapped with $[v_i^1, v_i^2]$. We claim that $P[v_i^2, S] \geq k$. Consider the rightmost vertex $v_{j}^2$ of $\pi$ such that $v_{j}^2$ belongs to a required pair covered by $\pi$ and not nested in the $i$-pair. Decompose path $\pi$ in two parts: one, $\pi'$, from $s$ to $v_{j}^2$, and the other one, $\pi''$, from $v_{j}^2$ to $v_i^2$. Let $S'$ be the set of vertices that belong to the required pairs covered by $\pi$ and overlapped with $[v_i^1, v_i^2]$. Let $k'$ be the number of required pairs covered by $\pi'$ and $k''$ be the number of the remaining pairs covered by $\pi$ (hence $k = k' + k''$). First, notice that $k'' = |\text{Ov}([v_i^1, v_i^2], S \setminus S')|$. By induction hypothesis we have that $P[v_i^2, S'] = k_1$ for some $k_1 \geq k'$. By construction, $S'$ is in agreement with $S$ and the subpath of $\pi$ from $v_{j}^2$ to $v_i^2$ contains all the vertices in $S \setminus S'$. As a consequence, by Eq. (1), $P[v_{j}^2, S]$ is at least $k_1 + k'' \geq k' + k'' = k$, which concludes the proof.

Let $h$ be the maximum number of overlapping required pairs in $D$, it follows that the number of possible subsets $S$ is bounded by $O(2^{2h})$. Then each entry $P[v_i^2, S]$ requires time $O(2^{2h}n)$ to be computed, and, since there exist $O(2^{2h}n)$ entries, the recurrence requires time $O(4^{2h}n^2)$. From Lemma 2 and Lemma 3, it follows that an optimal solution for MaxRPSP can be obtained by looking for the maximum of the values $P[v_i^2, S]$. Hence the overall time complexity of the algorithm is bounded by $O(4^{2h}n^2)$.

References

On Two Different Forms of Inefficiency in Network Design

Pietro Cenciarelli, Daniele Gorla, and Ivano Salvo

Sapienza University of Rome, Dpt. of Computer Science
{cenciarelli,gorla,salvo}@di.uniroma1.it

Abstract. Through the years, several models have emerged for studying the behaviour of network traffic, protocols, energy consumption, and so on. Formal models turn out to be a useful tool for reasoning about network design. In this paper, we consider two network models, one designed for nodes with depletable energy (e.g., Smartdust) and one for selfish routing. We compare two notions of inefficiency and show unexpected relationships between them.

1 Introduction

Through the years, several formal models have emerged for studying network design ([3,7,9], just to cite a few). Our investigation started in [5] with a model for nodes with depletable energy (e.g., Smartdust). To study energy efficiency and load balancing of multi-hop communication in ad-hoc networks, we considered a simple model where networks are oriented graphs equipped with a function that labels each node with a natural number representing its depletable charge. Node charge decreases during the life of a network, as the result of information passing, until a dead state (i.e., a state where no information can be passed anymore) is reached. For investigating the communication service provided by a network, we introduced a notion of equivalence on networks (inspired by trace equivalence in the theory of concurrency) and we provided a characterization of such equivalence, by showing that two networks are equivalent if they have identical maximum and minimum inhibiting flow.

Networks in which the minimum inhibiting flow is strictly less than the maximum flow, are in some sense inefficient, because choosing one routing or another can provide different communication services. Network designers can improve network performance by removing node connections, in particular, as we show in this paper, by removing paths within a network cut. This somehow resembles a counterintuitive but well-known situation, called Braess’s Paradox [2,4] in the setting of selfish routing. Braess’s paradox occurs when the equilibrium cost may be reduced by raising the cost of an edge or, equivalently, by removing such an edge.

We compare these two notions of inefficiency and show unexpected similarities between them. In particular, we present a graph-theoretic characterization of weak graphs, i.e. graphs in which the minimum inhibiting flow is strictly less than the maximum flow for some labelling of nodes. Then, we relate weak graphs to vulnerable graphs, i.e. graphs in which the Braess’s Paradox occurs.
2 Weak Graphs

We consider st-graphs, i.e. graphs with a distinguished source vertex s and sink vertex t. A path in an st-graph is a (possibly cyclic) directed walk from s to t. A cut in an st-graph is a minimal set T of vertices such that p ∩ T = ∅ for all paths p. Since we only consider source-to-sink paths, we can assume that every vertex belongs at least to one such path.

A depletable channel [5] (just channel for short) is an st-graph G endowed with a function η associating with each vertex a non-negative integer representing its depletable charge. When no confusion arises, we write just η for a channel (G, η) and call G its underlying graph. The set P(η) is the set of paths of η.

A flow for η is a function φ : P(η) → N such that φ(v) ≤ η(v), for every v ∈ V. Here, φ(v) denotes the amount of v’s charge consumed by φ; i.e., φ(v) = ∑p∈P(η) rvp · φ(p), where rvp denote the number of times in which node v is repeated in path p (0, if v /∈ p). The value of φ is #p∈P(η) φ(p). We denote by max η the maximum value of a flow for η. We call η a dead channel if max η = 0. A flow φ is said to inhibit η if η after φ is dead. We denote by min η the smallest value of an inhibiting flow in η.

Definition 1. We call weak an st-graph G when min η ≠ max η for some channel η whose underlying graph is G.

Two typical examples of channels are depicted in Fig. 1. Here, channel (1) is not weak, since every routing of messages always ensures the delivery of 5 information units (we assume that every information unit consumes one charge unit of every traversed node). By contrast, channel (2) is weak because there are flows that can deliver 5 units (the same as in channel (1)), but there are also flows that can only deliver 4 units (by sending information units along the vertical edge from the node labelled 2 to the node labelled 4).

Theorem 1. A graph is weak if and only if there exist a cut T and a directed walk a ∼ b with a, b ∈ T.

Proof. If. We define η(v) = 1 for all v ∈ T and we let η(v) be sufficiently large on all other vertices, so that max η = |T|. We may assume without loss of generality that the walk r : a ∼ b is acyclic and such that r ∩ T = {a, b}. Similarly, by the minimality of T, there exist (acyclic) directed walks p : s ∼ a and q : b ∼ t such that p ∩ T = {a} and q ∩ T = {b}. Then, a flow φ of one unit along the path s ∼ a ∼ b ∼ t is feasible and, it leaves the channel with a charge-to-node assignment θ such that max θ ≤ |T| − 2. Then, we can combine φ with a maximum flow for θ to obtain a dead network after a flow of value max θ + 1 < max η. Thus, G is weak.
Only if. Let \( \eta \) be such that \( \eta \) is inhibited by some flow \( \phi \) of value \( n < \max \eta \) and call \( \zeta \) the resulting (dead) channel. There exists a cut \( T \) in the graph such that \( \zeta(v) = 0 \) for all \( v \in T \). Since maximal flow corresponds to minimal cut, \( \max \eta \leq \sum_{v \in T} \eta(v) \). By definition \( \zeta(v) = \eta(v) - \sum_{v \in E} \phi(p) \) and hence \( \eta(v) = \sum_{v \in E} \phi(p) \). Suppose no directed walk exists between any two vertices of \( T \). Since \( T \) is a cut, all paths must include precisely one vertex in \( T \). Hence \( \sum_{v \in T} \sum_{v \in E} \phi(p) = \sum_{p} \phi(p) \). Summing up, we have the absurd:

\[
\max \eta \leq \sum_{v \in T} \eta(v) = \sum_{v \in T} \sum_{v \in E} \phi(p) = \sum_{p} \phi(p) = n < \max \eta. \]

It is worth remarking that every cyclic graph is weak, as a trivial corollary of Theorem 1. Indeed, every node belongs to some cut \( T \) and thus, if we have a cycle on \( v \), we trivially have a directed walk \( v \sim v \) with \( v \in T \).

3 Vulnerable Graphs

In this section, we essentially follow the presentation in [8]. As for channels, we consider oriented st-graphs. Here, a flow for a graph \( G = (V,E) \) is a function \( \phi : P(G) \rightarrow \mathbb{R}^+ \). A flow induces a unique flow on edges: for any edge \( e \in E \), \( \phi(e) = \sum_{p \in P(G); e \in p} \phi(p) \). Notice that in this model, since we do not have capacities on edges (like in the standard flow networks) or charges on nodes (like in the model presented in Section 2), a flow is simply a function assigning non-negative reals to paths, without any further constraint.

For each edge \( e \in E \) a latency function \( l_e : \mathbb{R}^+ \rightarrow \mathbb{R}^+ \) assigns to each edge a latency that depends on the flow on it. The latency of a path \( p \) is the sum of the latencies of the edges in the path, \( l_p(\varphi) = \sum_{e \in p} l_e(\varphi(e)) \).

Given a graph \( G \), a real number \( r \in \mathbb{R}^+ \) and a latency function \( l \), we call the triple \( (G,r,l) \) an instance. A flow \( \varphi \) is feasible for \((G,r,l)\) if the value of \( \varphi \) is \( r \).

A flow \( \varphi \) feasible for \((G,r,l)\) is at Nash equilibrium (or is a Nash flow) if, for all pairs of paths \( p,q \in P(G) \) such that \( \varphi(p) > 0 \), we have \( l_p(\varphi) \leq l_q(\varphi) \). In particular, this implies that, if \( \varphi \) is a Nash flow, all paths to which \( \varphi \) assigns a positive flow have the same latency. We will denote with \( L(G,r,l) \) the latency of all paths with positive flow at the Nash equilibrium.

Braess’s paradox [2,4] originates when the latency at the Nash equilibrium decreases because of removing edges (or equivalently, by raising the latency function on edges): an instance \((G,r,l)\) suffers from the Braess’s paradox if there is a subgraph of \( G \) with a lower latency. In Fig. 2 we show a minimal example of Braess’s paradox. A Nash flow of value 1 assigns all the flow to the path along the dashed arrows in the picture. The latency in such a case is 2. In Fig. 3, we show the optimal subgraph: in this case, a Nash flow of value 1 assigns \( \frac{1}{2} \) to both paths in the network, thus obtaining a latency of \( \frac{3}{2} \).

**Definition 2.** A graph \( G \) is vulnerable if there exist a flow value \( r \), latency function \( l \) and a subgraph \( H \) of \( G \) such that \( L(G,r,l) > L(H,r,l') \), where \( l' \) is the restriction of \( l \) on \( H \).
then there exist an inhibiting flow $\eta$.

**Lemma 1.** Given a cut $S$, we call the frontier of $S$ the set of nodes $F_S = \{ u \in S \mid \exists v \notin S. (u,v) \in E \}$.

A characterization of vulnerable (undirected) graphs is presented in [6]. In particular, in [6] it is proved that an undirected graph is vulnerable if and only if it contains the Wheatstone network.

## 4 Weakness vs Vulnerability

In this section, we investigate relationships between weak and vulnerable graphs. In particular, we will show that every weak graph is also vulnerable. We start by giving some preliminary notions. We omit some standard definitions (e.g., augmenting path) and refer the interested reader to [1]. For the sake of notation, we write $p \in \phi$ to denote $\phi(p) > 0$.

Given a cut $S$, we call the frontier of $S$ the set of nodes $F_S = \{ u \in S \mid \exists v \notin S. (u,v) \in E \}$.

Given a path $v_1v_2\ldots v_k$, we call $v_1$ and $v_k$ its extremes, while $v_2,\ldots,v_{k-1}$ are called internal.

An augmenting path $q$ touches a path $p$ if $p = p_1p_2p_3$ and $q = q_1p_2^Rq_2$, where $p_2^R$ denotes path $p_2$ traversed in reverse direction; $q$ touches a flow $\phi$ if $q$ touches a path $p \in \phi$.

We say that $q$ touches $\phi$ $n$ times if $q = q_0p_1^Rq_1p_2^R\ldots p_n^Rq_n$, where each $p_i$ is a segment of some $p \in \phi$ and each segment $q_i$ has internal vertices not touched by $\phi$.

We say that $q$ touches $\phi$ $n$ times before the frontier if $q = q_0p_1^Rq_1p_2^R\ldots p_n^Rq_n$ touches $\phi$ $n$ times and every $p_i$ has internal vertices not belonging to $F_{S_\phi}$.

**Proposition 1.** Let $\eta$ be a channel, $\phi$ an inhibiting flow and $S_\phi = \{ v \in V \mid \phi(v) > 0 \} \setminus \{ t \}$. Then:

1. $S_\phi$ is the maximum (w.r.t. $\subseteq$) saturated cut;
2. every path in $\eta$ passes through at least one node in $F_{S_\phi}$;
3. for all $v \in F_{S_\phi}$ there exists a path $v \rightsquigarrow t$ in $\eta$ after $\phi$ containing only nodes with positive energy (except for $v$).

**Lemma 1.** Let $\eta$ be a channel and $\phi$ an inhibiting flow of value $n < \max \eta$; then, there exist an inhibiting flow $\phi'$ of value $n$ and an augmenting path for $\eta$ after $\phi'$ that touches $\phi'$ before the frontier just once.

**Proof.** Let us consider an augmenting path $q$ for $\eta$ after $\phi$; such a path must exist because the value of $\phi$ is smaller than the maximum. Moreover, since
\( \phi \) is inhibiting, \( q \) touches \( \phi \) at least once, i.e. \( q = q_0p_1^Rq_1p_2^R \ldots p_k^Rq_k \) for some \( k > 0 \).

If \( p_1 \cap F_{S_o} \neq \emptyset \), then \( p_1 = p_1'vp_1'' \), where \( v \) is the last vertex of \( p_1 \) belonging to the frontier (i.e., \( v \in F_{S_o} \) and \( p_1' \cap F_{S_o} = \emptyset \)). By Proposition 1(3), there exists a path \( v \leadsto t \) in \( \eta \) after \( \phi \) (call it \( p'' \)) whose internal vertices are not touched by \( \phi \). Hence, the path \( q_0p_1'^Rp'' \) is an augmenting path that touches \( \phi \) before the frontier just once.

If \( p_1 \cap F_{S_o} = \emptyset \), let \( p = p'p_1p'' \) be the first path touched by \( q \). If \( p' \cap F_{S_o} \neq \emptyset \), then let \( v \) be the last vertex of \( p' = p_1'vp_2' \) belonging to \( F_{S_o} \); by Proposition 1(3), there exists a path \( v \leadsto t \) in \( \eta \) after \( \phi \) (call it \( p'' \)) whose internal vertices are not touched by \( \phi \) and, hence, \( q_0p_1'^Rp_2'^Rp'' = q_0(p_2'p_1')^R \) is an augmenting path that touches \( \phi \) before the frontier just once. Otherwise, consider the flow \( \phi' = (\phi - \{p\}) \cup \{q_0p''\} \) that still has value \( n \). Since \( p'p_1\cap F_{S_o} = \emptyset \), then \( F_{S_o} \subseteq F_{S_v} \); by Proposition 1(2), also \( \phi' \) is an inhibiting flow and \( p'q_1p_2^R \ldots p_k^Rq_k \) is an augmenting path for \( \eta \) after \( \phi' \) that touches \( \phi' \) before the frontier \( k - 1 \) times. We can repeat this reasoning for \( p_2, \ldots, p_k \) and eventually conclude, since \( q \) is a path from \( s \) to \( t \) and, hence, it contains at least one node of the frontier.

\( \square \)

**Lemma 2.** Let \((V, E, \eta)\) be a channel, \( \phi \) an inhibiting flow of value \( n < \max_n \) and \( q \) an augmenting path for \( \eta \) after \( \phi \) that touches \( \phi \) before the frontier just once; then, the graph \( G = (V, E) \) is vulnerable.

**Proof.** Let \( p \in \phi \) be the path that \( q \) touches before the frontier; thus, \( p = p_1p_2p_3, q = q_1p_2^Rq_2 \) and the internal vertices of \( p_2 \) do not belong to \( F_{S_o} \). Then consider the flow \( \phi' = (\phi \setminus \{p\}) \cup \{q_1p_3, p_1q_2\} \); its value is \( n + 1 \) and it is an inhibiting flow (indeed, \( F_{S_o} \subseteq F_{S_v} \)).

Consider the following latency function on \( G \):

\[
l(u, v) = \begin{cases} 
1 & \text{if } (u = s \lor v = t) \land (u, v) \notin \phi \\
0 & \text{otherwise}
\end{cases}
\]

The flow \( \varphi(p) = 1 \), if \( p \in \phi \), and \( \varphi(p) = 0 \), if \( p \notin \phi \), is a Nash flow and \( l_p(\varphi) = 2 \) for all \( p \in \phi \).

Let us consider the set \( O = \{(u, v) \mid u \in p_1q_2, v \notin p_1q_2\} \) of outgoing edges from \( p_1q_2 \) and the set \( I = \{(u, v) \mid u \notin q_1p_3, v \in q_1p_3\} \) of ingoing edges in \( q_1p_3 \), and consider the subgraph \( G' = (V, E - (O \cup I)) \). The flow

\[
\varphi'(p) = \begin{cases} 
\frac{n+1}{n+3} & \text{if } p \in \{q_1p_3, p_1q_2\} \\
\frac{n+2}{n+3} & \text{if } p \in \phi' \setminus \{q_1p_3, p_1q_2\} \\
0 & \text{otherwise}
\end{cases}
\]

is a Nash flow and \( l_p(\varphi') = \frac{2n+4}{n+3} < 2 \) for all \( p \in \phi' \). This proves that \( G \) is vulnerable.

\( \square \)

**Theorem 2.** Let \( G \) be a weak st-graph. Then \( G \) is vulnerable.
Proof. By Lemma 1, there exists an inhibiting flow of value smaller than the maximum touched before the frontier by some augmenting path just once; by Lemma 2 this suffices to conclude.

The converse is false. Let us consider the graph in Fig. 4. This graph is not weak. As a matter of fact, for all $m, n, p, q \in \mathbb{N}$, the channel $\eta$ in Fig. 5 has $\min_\eta = \max_\eta = \min\{m + n, p + q\}$. By contrast, this graph is vulnerable. To see this, consider a latency function as depicted in Fig. 5, that essentially yields the Wheatstone network of Fig. 2. Indeed, the $\infty$ cost of the outgoing diagonal edge in practice erases such an edge from the net.

The sufficient condition of Theorem 2 can be generalized:

**Corollary 1.** If $G$ contains a weak subgraph, then $G$ is vulnerable.

**Proof.** Let $G'$ be a weak subgraph of $G$. By Theorem 2, $G'$ is vulnerable; i.e., there exist $r \in \mathbb{R}_+$ and a latency function $l$ for the edges of $G'$ such that $G'$ has a subgraph $G''$ for which $L(G'', r, l'') < L(G', r, l)$, where $l''$ is the restriction of $l$ to $G''$. Let us consider the latency function $l'$ for $G$ such that

$$l'(e) = \begin{cases} l(e) & \text{if } e \in G' \\ \infty & \text{otherwise} \end{cases}$$

Clearly, $L(G, r, l') = L(G', r, l)$ and the restriction of $l'$ to $G''$ coincides with the restriction of $l$ to $G''$ (viz. $l''$). Thus, $G''$ is a subgraph of $G$ such that $L(G'', r, l'') < L(G, r, l')$. So, $G$ is vulnerable.

We conjecture that also the converse implication holds:

**If a graph is vulnerable, then it contains a weak subgraph.**

Thanks to Corollary 1, this conjecture would give a characterization of vulnerable graphs when the underlying model is directed, thus extending the result in [6]. It remains to be studied this characterization from the complexity point of view.

To conclude, it is worth remarking that the notion of weakness strongly depends on the fact that we assign charges to nodes, and not capacities to edges. Indeed, it is well-known [1] that in several settings the two models are interchangeable: the node charge can be easily converted to an edge capacity by expanding a node $v$ with charge $\eta(v)$ into an edge $(v^-, v^+)$ with capacity $\eta(v)$ (edges that are not “expanded nodes” have infinite – i.e., big enough –

\footnote{An infinite cost for an edge actually means “big enough”, i.e. bigger than the latency of the net without such an edge.}

---

Proceedings of ICTCS 2013
capacity). The two models are very closely related, in the sense that: every flow in the first model corresponds to a flow (of the same value) in the other; a inhibiting flow in the first model corresponds to a inhibiting flow in the other; the maximum flow in the first model corresponds to the maximum flow in the other (i.e., the usual maximum flow on an edge-capacitated network); etc.

Thus, also weakness can be adapted to the edge-capacitated model: a graph is \((\text{edge-})\text{weak}\) if there exists a capacity function for its edges such that it admits an inhibiting flow of value smaller than the maximum.

However, the translation does \textit{not} reflect weakness. Indeed, consider the graph of Fig. 4: it is \textit{not} weak, when charges are assigned to nodes (as already discussed before), but the graph underlying its translation (depicted in Fig. 7, where the short edges are the “expanded nodes”) is \((\text{edge-})\text{weak}\).

This should be surprising, but in fact it is not. Indeed, the notion of \((\text{edge-})\text{weakness}\) relies on a universal quantification over all the possible capacity functions, whereas the translation strongly relies on specific functions, viz. those assigning infinite capacity to the original edges (i.e., the ones not corresponding to the “expanded nodes”). For example, the graph of Fig. 7 can be shown not \((\text{edge-})\text{weak}\) only by assigning a 0 capacity to one of its diagonal edges.

5 Conclusion

We have studied two different models for networks, one with depletable node charge and the other with costs on the edges. We have compared two forms of inefficiencies in the design of such networks and came out with a preliminary relationship among them. If our conjecture is proved, the relationship will be completed. Another line of research is the comparison of these two inefficiencies with the edge-weakness mentioned at the end of Section 4; we think it coincides with vulnerability.

Acknowledgements. We are grateful to the reviewers that raised a few interesting points that stimulated the development of the paper.

References


\textit{Proceedings of ICTCS 2013}
On the number of prefix and border tables

Julien Clément$^1$ and Laura Giambruno$^1$

GREYC, CNRS-UMR 6072, Université de Caen, 14032 Caen, France
julien.clement@unicaen.fr, laura.giambruno@unicaen.fr

Abstract. For some text algorithms, the real measure for the complexity analysis is not the string itself but its structure stored in its prefix table (or border table, as border and prefix tables can be proved to be equivalent). We give a new upper bound on the number of prefix tables for strings of length $n$ (on any alphabet) which is of order $(1 + \varphi)^n$ (with $\varphi = \frac{1 + \sqrt{5}}{2}$ the golden mean) and present also a lower bound.

1 Introduction

The prefix table of a string $w$ reports for each position $i$ the length of the longest substring of $w$ that begins at $i$ and matches a prefix of $w$. This table stores the same information as the border table of the string, which memorises for each position the maximal length of prefixes of the string $w$ ending at that position. Indeed two strings have the same border table if and only if they have the same prefix table.

Both tables are useful in several algorithms on strings. They are used to design efficient string-matching algorithms and are essential for this type of applications (see for example [8] or [3]). It has been noted that for some text algorithms (like the Knuth-Morris-Pratt pattern matching algorithm), the string itself is not considered but rather its structure meaning that two strings with the same prefix or border table are treated in the same manner. For instance, strings $abbbbb$, $baaaaa$ and $abcdef$ are the same in this aspect.

The study of these tables has become topical. In fact several recent articles in literature (cf. [7, 4, 2, 5]) focus on the problem of validating prefix and border tables, that is the problem of checking if an integer array is either the prefix or the border table of at least one string. In a previous paper [10] the authors represented distinct border tables by canonic strings and gave results on generation and enumeration of these string for bounded and unbounded alphabets. Some of these results were reformulated in [5] using automata-theoretic methods.

In this paper, we define an injection from the set of prefix tables to another set of combinatorial structures called prefix lists which are easier to count. We then deduce a new upper bound and a new lower bound on the number $p_n$ of prefix tables (see Table 1 for first numerical values) for strings of length $n$ or, equivalently, on the number of border tables of length $n$.

Let $\varphi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$, the golden mean, we have:

**Proposition 1 (Upper bound).** The number of valid prefix tables $p_n$ can be asymptotically upper bounded by the quantity $\frac{1}{2} \left( 1 + \frac{\sqrt{5}}{2} \right)^n (1 + \varphi)^n + o(1)$. 


Table 1: First values: $p_n$ is the total number of prefix tables for strings of size $n$, $p_{n,k}$ is the number of prefix tables for strings of size $n$ with an alphabet of size $k$ which cannot be obtained using a smaller alphabet.

Note that different words on a binary alphabet have distinct prefix/border tables. This gives us a trivial lower bound in $2^{n-1}$ (since exchanging the two letters of the alphabet does not change tables). This is no longer true as soon as the alphabet has cardinality strictly greater than 2: for instance, words $abb$ and $abc$ admit the same prefix table $[3,0,0]$.

**Proposition 2 (Lower bound).** For any $\varepsilon > 0$ there exists a family of prefix tables $(\mathcal{L}_n)_{n \geq 0}$ such that $\text{Card}(\mathcal{L}_n) = \Theta((1 + \varphi - \varepsilon)^n)$.

Sketches of proof for these two propositions are deferred to Sections 3.3 and 3.4.

## 2 Preliminaries

### 2.1 Notations and definitions

Let $A$ be an ordered alphabet. A word $w$ of length $|w| = n$ is a finite sequence $w[0]w[1] \ldots w[n-1] = w[0 \ldots n-1]$ of letters of $A$. The language of all words is $A^*$, and $A^+$ is the set of nonempty words. The prefix (resp. suffix) of length $\ell$, $1 \leq \ell \leq n$, of $w$ is the word $u = w[0 \ldots \ell - 1]$ (resp. $u = w[n - \ell \ldots n - 1]$). A border $u$ of $w$ is a word that is both a prefix and a suffix of $w$ and distinct from $w$ itself. We define bord($w$) as the set of all proper borders of $w$.

**Definition 1 (Prefix table).** The prefix table $\text{Pref}_w$ of a word $w \in A^+$ of length $n$, is the table of size $n$ defined, for $0 \leq i < n$, by

$$\text{Pref}_w[i] = \text{lcp}(w, w[i \ldots n-1]),$$

where lcp denotes the maximal length of common prefixes of the two words.
Another well-known structure used to represent the correlation structure of a string is the border table of a word.

**Definition 2 (Border table).** The border table \( \text{Border}_w \) of a word \( w \in A^+ \) of length \( n \), is the table of size \( n \) defined, for \( 0 \leq i < n \), by

\[
\text{Border}_w[i] = \max\{|u| \mid u \text{ is a border of } w[i]\},
\]

**Example 1.** Let \( w \) be the word \( \text{abaababa} \). The following represents the prefix and border tables of \( w \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w[i] )</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>( \text{Pref}_w[i] )</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( \text{Border}_w[i] )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

These structures (border and prefix tables) are in facts equivalent; actually the following proposition states a fact discussed in [3] and recently deepened in [1].

**Proposition 3.** Two strings have the same border table if and only if they have the same prefix table.

**Proof (sketch).** Let \( w \) be a word in \( A^+ \) of length \( |w| = n > 0 \). We can relate the border table \( \text{Border}_w \) to the prefix table \( \text{Pref}_w \).

For a position \( j \) in \( w \) of length \( n \), let \( I(j) = \{i \mid 0 < i \leq j \text{ and } i + \text{Pref}_w[i] - 1 \geq j\} \). The elements in \( I(j) \) represent the positions \( i \leq j \) for which the longest common prefixes between \( w \) and \( w[i..n-1] \) overlap position \( j \) in \( w \). Then we have

\[
\text{Border}_w[j] = 0 \text{ if } I(j) = \emptyset, \text{ and } \text{Border}_w[j] = j - \min I(j) + 1 \text{ otherwise}. \quad (1)
\]

Conversely, given the border table \( \text{Border}_w \) of \( w \), we define the prefix table \( \text{Pref}_w \) in the following way. First we set \( \text{Pref}_w[0] = |w| \). Then let \( j > 0 \) be a position in \( w \) and let \( I'(j) = \{i \mid j \leq i < |w| \text{ and } w[j..i] \in \text{bord}(w[0..i])\} \). We have

\[
\text{Pref}_w[j] = 0 \text{ if } I'(j) = \emptyset, \text{ and } \text{Pref}_w[j] = \max I'(j) - j + 1 \text{ otherwise}. \quad (2)
\]

With (1) and (2), one proves that two words have the same border table if and only if they have the same prefix table. \( \square \)

Recent literature focuses on the problem of validating prefix and border tables and, in case of a valid table, providing the smallest lexicographically word associated to it (cf. [7, 2]).

### 2.2 Previous work

Previous work in [10] focused on counting distinct strings of length \( n \) with respect to their prefix:border tables: an upper bound is given in the form

\[
b_n = \sum_{k=1}^{k^*} \binom{n-2k-1+k}{k},
\]

\( \text{Proceedings of ICTCS 2013} \)
where \( \{m\} \) denotes the Stirling number of second kind (the number of partitions of \( m \) in \( j \) non empty parts), and \( k^* = \lceil \log_2(n + 1) \rceil \). The quantity \( k^* \) is the minimal number of distinct letters to obtain all possible prefix tables of size \( n \).

Numerically it is clear that \( b_n \) is far from being a tight approximation of the number \( p_n \) of prefix tables of size \( n \). One can indeed prove that \( b_n \gg p_n \) (the proof is deferred to a journal version of this paper).

In the next section we try to improve the bound in (3), yielding the result of Proposition 1.

3 Prefix Lists

The information in a valid prefix table is somewhat redundant since we do not need to use all values to build a corresponding word. We introduce prefix lists which contain minimal information to be able to reconstruct such a word. In this section we define an injection from the set \( P \) of prefix tables to another set \( L \) of combinatorial structures called prefix lists which are easier to count. Finally we give upper and lower bounds.

**Definition 3.** We define a prefix list \( L = [\ell_1, \ldots, \ell_k] \) as a finite sequence of positive integers together with a size defined for a list as \(|L| = \sum_{i=1}^{k} |\ell_i| \), where the size \(|i|\) is \( i \) if \( i > 0 \) and 1 if \( i = 0 \).

### 3.1 Algorithms

*From prefix tables to prefix lists.* We define constructively an injection \( \psi \) from \( P \) to \( L \) by defining an algorithm in a “right-to-left manner”.

```plaintext
Procedure PrefixToList\( (P = P[0\ldots n - 1]) \)

\[
\begin{align*}
L &\leftarrow [] \\
i &\leftarrow n - 1 \\
\text{while } i > 0 \text{ do} \\
 I &\leftarrow \{j \mid 0 \leq j \leq i \text{ and } j + P[j] - 1 \geq i\}
\quad \text{if } I = \emptyset \\
\quad \quad \quad (\ell, i) &\leftarrow (0, i - 1)
\quad \text{else} \\
\quad \quad \quad (\ell, i) &\leftarrow (i - \min(I) + 1, \min(I) - 1)
\quad L &\leftarrow [\ell] \cdot L
\end{align*}
\]

return \( L \)
```

Intuitively, it scans the prefix table from right to left, starts with the last position \( i = n - 1 \) and gets from the prefix table the length \( \ell \) of the leftmost longest common (proper) prefix which overlaps the current position \( i \), or sets \( \ell = 0 \) if there is no such prefix. This length is inserted at the beginning of the list and the position \( i \) is updated to the position immediately before the prefix (if it exists) or just one position before (if it is not the case). The algorithm stops when the first position \( i = 0 \) is attained.

For each position \( i \) in \( P \), the elements in \( I \) represent, as in the proof of Proposition 3, the positions less or equal to \( i \), such that the longest common prefix with \( w \) starting at these positions overlap position \( i \).
Example 2. Let \( w \) be the word \( \text{abaababa} \). The following represents the prefix table of \( w \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w[i] )</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>( \text{Pref}_w[i] )</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

For this table we get the associated list \( L = [0, 1, 2, 3] \). In fact, executing the algorithm \( \text{PrefixToList} \) to \( \text{Pref}_w \), we start with \( i = 7 \) and we get that the set of starting indexes of prefixes overlapping \( i \) is \( I = \{5, 7\} \). Thus \( \ell = i - \min(I) + 1 = 3 \), the length of the overlapping prefix until \( i \), is appended to \( L = [\] \). Now \( i \) is initialised to 4 the position before \( \min(I) = 5 \). Next we have \( I = \{3\} \) and so \( \ell = 2 \) is appended to \( L = [3] \) and \( i := 2 \). Again \( I = \{2\} \), \( \ell = 1 \) and \( i := 1 \). Now \( I = \emptyset \), thus \( \ell = 0 \), \( i := 0 \) and the algorithm stops.

Remark 1. At first view, it would be more intuitive to define prefix lists with an algorithm visiting the prefix table from left to right. For instance a greedy algorithm for this construction starts with the second position \( i = 1 \) and appends at the beginning of the list \( \ell \) the length of the longest common prefix starting there. Then \( i \) is updated to \( i + \ell - 1 \) and again the algorithm appends the element in the prefix table of position \( i \) and so on until position \( n - 1 \) is attained.

However this construction of prefix list from left to right fails to define an injection from prefix tables to prefix lists (which is our goal for finding an upper bound). In fact let \( P = [8, 0, 1, 3, 0, 3, 0, 1] \) be a valid prefix table, as in Example 2, and \( P' = [8, 0, 1, 3, 0, 1, 0, 1] \) be a valid prefix table associate to \( w' \), then one has

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w'[i] )</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>( \text{Pref}_{w'}[i] )</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Since to \( P \) and \( P' \) is associated the same list \( L = [0, 1, 3, 0, 1] \) then the correspondence between prefix tables and these lists cannot be injective.

Remark 2 (Border lists). Note that, due to the equivalence between prefix and border tables, we can define an algorithm with input a border table that creates the prefix list associated to the equivalent prefix table.

From lists to words. We now describe an “inverse” algorithm that associates to a prefix list \( L = \psi(P) \), corresponding to a valid prefix table \( P \), a word \( w \) whose prefix table is \( P \).

Informally the algorithm proceeds from left to right on the list input \([\ell_1, \ldots, \ell_m]\). It starts with a word reduced to one letter. Then iteratively for \( i \in [1 \ldots m] \), if \( \ell_i > 0 \) the algorithm copies \( \ell_i \) symbols, from the previous constructed word \( u \), at the end of \( u \), otherwise the algorithm introduces a new symbol in \( w \). Note that overlapping is allowed since we are building the word from left to right.

Example 3. Let \( w = \text{abaababa} \) as in Example 2, whose associated list is \( L = \psi(\text{Pref}_w) = [0, 1, 2, 3] \). Choosing arbitrarily the first letter to be \( a \), one can build \( w = a \cdot b \cdot a \cdot a \cdot b \cdot a b \cdot a b a \). A value 0 in the list implies we can choose a new letter (here \( b \) at the second position).
One key property is that the word \( w \) obtained by this algorithm performed on a list \( \psi(P) \) for a prefix table \( P \) is such that \( \text{Pref}_w = P \), that is prefix lists and prefix tables are equivalent and represent the same information. The proof (to be found in an extended version of this abstract) of this fact relies on properties of prefixes table stated in [2] (see also [4, 10]) asserting how a prefix/border table \( P \) may be extended to the right into another (longer) table.

### 3.2 Injective property

We get the following proposition

**Proposition 4.** The application \( \psi : \mathcal{P} \to \mathcal{L} \) is an injection. Moreover for a prefix table \( P \) of size \( n \), one has \( |\psi(P)| = n - 1 \) (considering for a list \( L = [\ell_1, \ldots, \ell_m] \) the size as \( |L| = \sum_{i=1}^{m} \ell_i + \text{Card}\{i \mid \ell_i = 0\} \)).

Indeed, let us consider two prefix tables \( P \neq P' \) and suppose that \( \psi(P) = \psi(P') = L \). The algorithm performed on \( L \) gives a word \( w \) such that \( \text{Pref}_w = P = P' \). Hence we must have \( \psi(P) \neq \psi(P') \).

**Remark 3.** Note that the application \( \psi \) is not surjective. For instance, to a list \( L = [0, 2, 2] \), we associate the word \( w = a \cdot b \cdot ab \cdot ab = \overline{ababab} \) by the previous algorithm. The prefix table of \( w \) is \( \text{Pref}_w = [6, 0, 4, 0, 2, 0] \) and, since \( \psi(\text{Pref}_w) = [0, 4] \), the function \( \psi \) is not surjective. In the following we say that a prefix list is *valid* if it is the image of a prefix table.

### 3.3 Upper bound

**Prefix lists.** We can see the set of prefix lists as a combinatorial class \( \mathcal{L} \) of lists of positive integers

\[
\mathcal{L} = \text{SEQ}(\{0, 1, 2, 3, \ldots\}),
\]

(4)

together with a *special size measure* which can be defined for a list \( L = [\ell_1, \ldots, \ell_k] \) as \( |L| = \sum_{i=1}^{k} |\ell_i| \), where the size \( |i| \) is \( i \) if \( i > 0 \) and \( 1 \) if \( i = 0 \). It just means that \( |L| = \sum_{i=1}^{k} \ell_i + \text{Card}\{i \mid \ell_i = 0\} \). The \( \text{SEQ} \) operator applied to a combinatorial class \( \mathcal{A} \) corresponds to all finite sequences of elements from \( \mathcal{A} \), i.e., \( \text{SEQ}(\mathcal{A}) = \cup_{n=0}^{\infty} \mathcal{A}^n \) (reminiscent of the Kleene star operation for regular languages). By convention \( \mathcal{A}^0 = \{\varepsilon\} \).

**Combinatorial specifications and generating functions.** In order to study a sequence \( (a_n)_{n \in \mathbb{N}} \), it is now usual [6] to consider its generating function \( A(z) \), that is the formal power series defined by \( A(z) = \sum_{n \geq 0} a_n z^n = \sum_{|\alpha|} z^{\alpha} \).

In our case, given the combinatorial specification of \( \mathcal{L} \), it is easy [6] to compute the generating function \( L(z) = \sum_{n \geq 0} \ell_n z^n \) where \( \ell_n \) denotes the numbers of lists of size \( n \). This is true when specification are unambiguous (in the same way as unambiguity is considered in regular expressions or formal grammars).

Indeed, the general idea is the following: here we first consider a set of atoms \( \mathbb{N} \). We need a size \( |\cdot| \) compatible with the cartesian product and disjoint union, i.e., here for \( i \in \mathbb{N} \) the size of atom \( i \) is \( |i| = i \) if \( i > 0 \) and \( |0| = 1 \). Let
us define an empty element $\varepsilon$ (the only one with size 0). Then we have the following dictionary for translating directly from combinatorial constructions to generating functions.

Empty element: $\varepsilon \mapsto 1$, Cartesian product: $A \times B \mapsto A(z) \times B(z)$,

Symbols: $\alpha \in \mathbb{N} \mapsto z^{[\alpha]}$, Disjoint Union: $A \cup B \mapsto A(z) + B(z)$,

Sequence product: $\text{SEQ}(A) \mapsto \frac{1}{1-A(z)}$

With this dictionary and the combinatorial description (4), we get

$$L(z) = \frac{1}{1-(z+\frac{z}{1-z})} = \frac{1-z}{1-3z+zz^2}.$$ 

Since this is a rational function, using decomposition in simple elements and setting $\varphi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$, we get for $\ell_n$ the coefficient of $z^n$ in $L(z)$

$$\ell_n = \frac{1}{2} \left(1 + \frac{\sqrt{5}}{5} \right) (1 + \varphi)^n + \frac{1}{2} \left(1 - \frac{\sqrt{5}}{5} \right) (1 + \varphi)^{-n} \quad (5)$$

$$= \frac{1}{2} \left(1 + \frac{\sqrt{5}}{5} \right) (1 + \varphi)^n + o(1).$$

**Result.** The main result on the upper bound (see Proposition 1) is a reformulation of the following corollary, which is a consequence of Proposition 4.

**Corollary 1.** The number $p_n$ of prefix tables of size $n$ is upper bounded by $\ell_{n-1}$ (defined in Equation (5)).

### 3.4 Lower bound

For the lower bound, we exhibit some sets of valid prefix lists such that we are able to count them. We wish these sets to be as large as possible. In this paper, as a first step, our goal is to evaluate the exponential order growth given in Proposition 2 rather than to give a precise estimate.

The idea for proving Proposition 2 is to exhibit a language which maps bijectively to a set of prefix lists, hence maps bijectively to a set of prefix tables.

Let us consider, for a fixed $k$, $L_k = ab^k (ab^{<k}(\varepsilon + cb^*))^*$. One can prove that each word in $L_k$ is in bijection with a unique prefix table. By using analytic combinatorics for regular expressions and since this regular expression is unambiguous, one can compute easily the generating function $L_k(z)$ for $L_k$

$$L_k(z) = z^{k+1} \frac{1}{1-\left(\frac{1-z}{1-z(1+\varepsilon)}\right)} = \frac{z^{k+1}(z-1)^2}{1-3z+z^2+z^{k+1}}.$$ 

By general principles [6] we have that $\ell_{n,k} := [z^n]L_k(z) \sim C_k \frac{1}{\rho_k}$, where $C_k$ is a constant and $\rho_k$ is the smallest real (simple) root of $1 - 3z + z^2 + z^{k+1}$. Considering $\rho_k = \frac{1}{1+\varphi-\varepsilon_k}$ as a perturbation of the solution of $1 - 3z + z^2 = 0$, we can then evaluate $\varepsilon_k$ by using the bootstrapping method (as in [9]). We get that $\varepsilon_k = \frac{1}{2}(1 + 3\sqrt{5}) (1 + o(1))$. Hence, for any $\varepsilon > 0$, one can fix $k$ such that $\ell_{n,k} = \Theta((\varphi + 1 - \varepsilon)^n)$ yielding the result of Proposition 2.
This result gives only rough information on the asymptotics of $\ell_{n,k}$. A more thorough study is in order to get better estimates. However this hints at the following conjecture.

Conjecture 1. The number $p_n$ of prefix tables of size $n$ is asymptotically equivalent to $c(1+\varphi)^n$ for a constant $c > 0$.

Conclusion

In this paper we have provided some bounds for the number of prefix (or border) tables. The problem of finding an asymptotic equivalent for the number of prefix tables is however still open, and would require a very fine understanding of the autocorrelation structure of words.

Acknowledgements

We would like to thank Maxime Crochemore, Cyril Nicaud and Giuseppina Rindone for helpful discussions.

References

Bridging syllogistics with combinatorics*

Eugenio G. Omodeo¹, Alberto Policriti², Alexandru Tomescu³

¹ Dipartimento di Matematica e Geoscienze, Università di Trieste,
Via Valerio 12/1, I-34127 – Trieste, Italy
email: eomodeo@units.it

² Dipartimento di Matematica e Informatica, Università di Udine,
Via delle Scienze 206, I-33100 – Udine, Italy
email: alberto.policriti@uniud.it

³ Department of Computer Science, University of Helsinki,
P.O. 68 (Gustaf Hällström katu 2b), FI-00014 – Helsinki, Finland
email: Alexandru.Tomescu@cs.helsinki.fi

Introduction

The core inferential mechanism in a recent proof-checker based on Set Theory, Ref ¹², implements an enhanced variant of multi-level syllogistic [⁵⁴]. This is a decision algorithm which determines whether a given formula involving only individual variables, which designate sets, and a restricted collection of set operators, is satisfiable. The said mechanism intervenes a few times, e.g., during Ref’s validation of the two tiny proofs shown in Fig. 1: in either proof, it is used once to check that the statement which starts the argument by contradiction is equivalent to the instantiated negation of the claim; then, less shallowly, to establish a conflict between that statement and the definition of \( \Theta \).

By and large, multi-level syllogistic has the ability to check a prenex \( \exists^*\forall \)-sentence in the relations \( \in, = \) for truth over sets. In practice, neither the Ref’s user nor the decision algorithm must handle quantifiers explicitly or need to eliminate the dyadic operators \( \cup, \setminus \) or the monadic operator \( \{ , \} \) (to mention a few). But the very possibility to do this reduction gives us a clue on the power of the decision method; to the authors, it made plain how to adapt the method to Aczel’s non-standard view on sets [⁷]; moreover, by bringing set-theoretic syllogistics closer to the stream of classical research on the decision problem for predicate calculus [³], it suggested ways to reinforce the known decidability results about those syllogistics.

In recent papers we have moved on to the much larger class, named BSR⁴, of all set-theoretic \( \exists^*\forall^* \)-sentences.

Concerning pure logic, namely first-order predicate calculus with equality, the \( \exists^*\forall^* \) satisfiability problem was solved long ago by Bernays and Schönfinkel. Frank P. Ramsey, by analyzing the full spectrum of interpretations modeling each sentence in this class (over an arbitrary, uninterpreted signature), got a foundational result in combinatorics [¹²]. Partly influenced by his historical success, we tackled the BSR truth problem in the context of Set Theory.

---

* Work partially supported by the INdAM/GNCS 2013 project “Strumenti basati sulla teoria degli insiemi per la verifica di algoritmi” and by FRA-UiTS PUMA.

⁴ This is an acronym for Bernays-Schönfinkel-Ramsey.
The following definition requires that injo send to \( \emptyset \):
- every set lying inside \( d_0 \),
- an arbitrary but fixed element \( a \) of the set-difference \( v_0 \setminus d_0 \).
Moreover, injo shall send each \( w \in v_0 \setminus d_0 \setminus \{ a \} \) to \( \{ \{ v_0 \} \cup (v_0 \setminus \{ w \}) \} \),
and each set lying outside \( v_0 \) to \( \{ v_0 \cup \{ v_0 \} \} \).

**Def** inj: \( \text{injo}(W) = \text{Def} \) if \( W \in d_0 \cup \{ \text{arb}(v_0 \setminus d_0) \} \) then \( \emptyset \) else \( \{ \{ v_0 \} \cup (v_0 \setminus \{ W \}) \} \).

**Theorem** aninjection\(_0\): [The restriction of injo to \( v_0 \setminus d_0 \) is 1–1]

\[
X \in v_0 \setminus d_0 \land Y \notin d_0 \land \text{injo}(X) = \text{injo}(Y) \rightarrow X = Y.
\]

**Proof:**
- Suppose \( \text{not}(x_0, y_0) \Rightarrow x_0 \in v_0 \setminus d_0 \setminus \{ y_0 \} \land y_0 \notin d_0 \land \text{injo}(x_0) = \text{injo}(y_0) \)
- Use def(injo), \( \Rightarrow \) false; Discharge \( \Rightarrow \) QED

**Theorem** aninjection\(_1\): [No membership between injo images of operands outside \( d_0 \)]

\[
\{ X, Y \} \cap d_0 = \emptyset \land X \in v_0 \rightarrow \text{injo}(Y) \notin \text{injo}(X).
\]

**Proof:**
- Suppose \( \text{not}(x_0, y_0) \Rightarrow x_0 \in v_0 \setminus d_0 \land y_0 \notin d_0 \land \text{injo}(x_0) \in \text{injo}(y_0) \)
- Use def(injo), \( \Rightarrow \) false; Discharge \( \Rightarrow \) QED

Fig. 1. Multi-level syllogistic invisibly at work in a Ref’s proof scenario.

Today, as we are about to report, that problem has been solved \([8, 9]\) for sets in von Neumann’s hierarchy of well-founded sets. (A partial solution, namely a \( \exists^* \forall \forall \)-satisfaction algorithm regarding the same realm, was presented in \([2]\).)

In Ramsey’s logical framework, it is generally easier to satisfy a BSR formula in a structure whose underlying domain is smaller. In presence of =, enlarging the domain may become troublesome. Indeed, the essence of Ramsey’s combinatorial analysis was the proof that when an \( \exists^* \forall \forall \) sentence with equality can be satisfied in a structure whose domain’s cardinality exceeds a specific computable threshold \( R \), then the sentence admits models of every integer cardinality larger than \( R \).

Within Set Theory, on the opposite, one can express the existence of infinite sets by way of a prenex \( \exists \forall \exists \forall \) sentence. More generally, for each \( n > 1 \), one can state the existence of \( n \) infinite sets by means of an \( \exists \cdots \exists \forall \forall \cdots \forall \) sentence, e.g. in the way shown by the template in Fig. 2.

On the basis of this remark and by paralleling the techniques involved in our decision method with Ramsey’s combinatorics, in \([6]\) we have begun to study the possibility of analyzing the spectrum of any \( \exists^* \forall \forall \) logical sentence by translating it into a set-theoretic BSR formula, so that the infinitude of the spectrum of the former can be revealed simply through the satisfaction of the latter. After concretizing this plan, one could exploit the results on syllogistics, i.e. on decidable fragments of Set Theory, not only as an aid to correct reasoning but also as a combinatorial means to collectively specify all possible ways of satisfying a given logical sentence.
We must establish whether a given formula
\( \forall y_1 \cdots \forall y_m \varphi(x_1, x_2, \ldots, x_n, y_1, \ldots, y_m) \) in the relators \( \in = \) can, or cannot, be made true by an assignment of sets to its existential variables \( x_i \). In the affirmative case our algorithm will also produce a (finite representation of a) model, i.e., a satisfying assignment. In this sense, it will not act as a simple-minded satisfiability tester, but as a satisfaction algorithm which constructs a model whenever possible.

Within Set Theory one can express the existence of infinite sets by way of a prenex \( \exists \exists \forall \forall \) sentence (as recalled above), but not by way of an \( \exists \forall \) sentence [11, 6]. In raising the skills of a decision method from the \( \exists \forall \)- to the \( \exists \forall^* \)-class, one encounters here a major challenge; also, each universal quantifier can add intricacy to the interplay among the infinite sets in a satisfying assignment.

Addressing the decision problem for the entire \( \exists \forall^* \) class in a single shot, offers a pleasant initial facilitation: thanks to extensionality (according to which, distinct sets do not own exactly the same elements), one can get rid of the equality symbol. In practice, one replaces the given sentence \( \exists x_1 \cdots \exists x_n \forall y_1 \cdots \forall y_m \varphi \) by a finite collection \( \Psi \) of \( \forall^* \)-formulae so that \( \forall y_1 \cdots \forall y_m \varphi \) can be satisfied through an assignment \( x_i \mapsto x_i \) of sets to its existential variables if and only if at least one formula \( \psi \) in \( \Psi \) can be satisfied injectively, i.e. by means of an assignment whose images are pairwise different sets. One can manage that each \( \psi \) in \( \Psi \) be devoid of the symbol =, usually at the price of introducing new existential variables.

Another essential preparation of the formulae to be tested for injective satisfiability consists in bounding the universal variables: specifically, on the grounds of a reduction carried out in [8, pp. 468–470], one can assume the following restricted format for formulae of the BSR class:

\[
\Phi = \bigwedge_{i=1}^{\aleph_1} (\forall y_1 \in z_1) \cdots (\forall y_m \in z_m) \phi_i(x_1, \ldots, x_n, y_1, \ldots, y_m),
\]

where \( z_h \in \{x_1, \ldots, x_n, y_1, \ldots, y_{h-1}\} \) for \( h \in \{1, \ldots, m\} \), and equality does not appear in any of the unquantified matrices \( \phi_i(x_1, \ldots, x_n, y_1, \ldots, y_m) \).

We must focus on models of a special, irredundant nature which can be captured by a finite (di)graph structure \( G \) on the one hand and can also suggest, on the other hand, how to compute a bound on the size of \( G \). Arcs will represent the inverse \( \in \) of membership restricted to the sets associated with the nodes.

Let \( x_i \xrightarrow{M} x_i \) be an injective model of \( \Phi \) and consider the transitive membership closure \( \text{TrCl}(\mathcal{F}) \) of the family \( \mathcal{F} \) of sets onto which the \( x_i \)'s are mapped.
by $\mathcal{M}$. Redundancy might derive from the presence of overly complex infinite sets in $\text{TrCl}(\mathcal{F})$. As proved in [8], the only unescapable kinds of infinitude can be described by means of formulae falling under the template of Fig. 2. These infinite sets are internally organized in regular structures: in a faithful graph representation of $\text{TrCl}(\mathcal{F})$, each one of these structures would form a peculiar ascending membership spiral. In $\mathcal{G}$ these situations will be encoded by finite cycles. $\text{TrCl}(\mathcal{F})$ will consist of nodes appearing in the said spirals, and of additional nodes forming the so-called core of $\mathcal{M}$, which includes the $x_i$’s.

In [9] we tackled the problem of setting a bound on the size of the core, and to compute it on the basis of how many existential/universal variables appear in $\Phi$. Thanks to this computable bound, the semi-decision algorithm proposed in [8] evolved into a decision algorithm.

To pinpoint additional restrictions on the nature of a model $\mathcal{M}$ worth of consideration, we can insist that $\text{TrCl}(\mathcal{F})$ owns no more elements per rank than the number $n$ of $x_i$’s. To these restrictions (and a few more), which appeared already in [8], we added an important one in [9]: the core has least possible cardinality. Altogether, the irredundancy assumptions, enable us to get a bound on the cardinality of $\mathcal{G}$. In particular, the bound on the size of the core is obtained very much in the spirit of the original Ramsey’s result.

Two steps are necessary: an equivalence relation of finite index on tuples of sets in the core (actually, on their membership graphs) and an application of the pigeonhole principle to a “striped” version of the core. The first step allows one to classify the elements of $\text{TrCl}(\mathcal{F})$ into finitely many types, in such a way that different elements of the same type can be interchangeably used to construct a model, as far as the satisfaction of the given BSR formula is concerned. Then, after having subdivided the core into “stripes”, one uses the pigeonhole principle to contract $\mathcal{M}$ into another satisfying assignment if any of its stripes repeats. Such a contraction, if doable, would lead to a smaller core, which is absurd.

2 Expressiveness of the BSR set-theoretic class and future research.

The BSR set-theoretic class turns out to be much more expressive than the corresponding class of formulae interpreted in merely logical terms. The observation, already made, that infinity can be captured by a BSR formula in the former framework but not in the latter, gives evidence of the higher expressiveness of the set-theoretic language. At a more elementary level, this can be seen from the three formulae displayed in Fig. 3: their status, which is indicated in the caption of that figure, depends either on extensionality alone or (for the third of them and richer variants of it, cf. [7]) on very little more.

In [6, Sec. 4] we gave two results ensuring the applicability of a technique embedding logical structures in the transitive closure of tuples of sets satisfying a given set-theoretic BSR formula. The first of those results refers to a non

---

5 A simple recursive characterization of the rank function from sets to ordinals is: $\text{rk}(X) = \sup\{ \text{rk}(y) + 1 : y \in X \}$.
Bridging syllogistics with combinatorics

\[
\begin{align*}
\exists x_1 \exists x_2 \exists x_3 \forall y \ (x_1 \in x_3 \land x_2 \notin x_3 \land (y \in x_1 \iff y \in x_2)) \\
\forall y_1 \cdots \forall y_n \left( \bigvee_{i=0}^{n-1} \bigwedge_{j=i+1}^n (y_k \in x_i \iff y_k \in x_j) \right) \\
\exists y_0 \cdots \exists y_n \left( \bigwedge_{j=0}^{n-1} \bigvee_{k=j+1}^n (y_k \in x_i \iff y_k \notin x_j) \land \\
\bigvee_{k=0}^n \bigwedge_{j=0}^{n-k} (y_k \notin y_i \land (i \neq k \rightarrow y_i \in x_i)) \right)
\end{align*}
\]

Fig. 3. A false $\exists^* \forall$ sentence, an injectively unsatisfiable $\forall^*$ scheme, and the contrary of an injectively unsatisfiable $\forall^*$ scheme.

well-founded universe of sets as target, while the second maps logical models into the standard universe of sets. But in neither case have we achieved, or can we take for granted, the universal result envisioned in the introduction and instrumental to proving the methodology suitable for the study of the entire spectrum of a given BSR (logical) formula.

Conclusions

Our satisfiability result for the BSR class exploits essentially the same combinatorial machinery as [12], after a recasting of the combinatorics in set-theoretic terms. Can we push the parallel further, to offer a very transparent set-theoretic insight on Ramsey’s celebrated theorem?

Now that the BSR-satisfiability problem has been solved over the von Neumann hierarchy, we expect that the analogous problem for the non-well-founded Aczel’s universe [1] can also be solved in the positive. Striking analogies between the two realms exist indeed [6], but our efforts have revealed a much greater intricacy of the satisfiability problem over Aczel’s sets.

As of today, the satisfaction algorithm for the set-theoretic BSR class proceeds by brute force, blindly attempting graphs one after another, no guidance being drawn for the systematic visit of the search space from the detailed structure of the input formula. Unless a goal-driven decision algorithm gets developed, the tester for BSR formulae cannot join or supersede multi-level syllogistic inside the proof-checker Ref.

References


Dynamically Closed Sets in Occurrence Nets

Luca Bernardinello¹, Carlo Ferigato², Stefan Haar³, and Lucia Pomello¹

¹ DISCo, Università degli studi di Milano - Bicocca
viale Sarca 336 U14, Milano, Italia
² JRC, Joint Research Centre of the European Commission
via E. Fermi, 1 21027 Ispra, Italia
³ LSV, CNRS & ENS de Cachan
avenue du Président Wilson, 61 94235 CACHAN Cedex, France

Abstract. We presents a selection of results given in [1]. The semantics of concurrent processes can be defined in terms of partially ordered sets. Occurrence nets, which belong to the family of Petri nets, model concurrent processes as partially ordered sets of occurrences of local states and local events. Here, we consider occurrence nets with forward conflicts, modelling families of processes. We study two closure operators on the elements of such occurrence nets and in particular, we show under which conditions they coincide and form complete, algebraic orthomodular lattices.

1 Introduction

We presents a selection of results given in [1].

Partially ordered sets (posets) are a traditional tool for modelling concurrent processes, in which the notions of causal dependence and independence, or concurrency, are clearly represented by the order relation and its complementary (non-order) relation.

We consider a special class of partially ordered sets, introduced within the theory of Petri nets: occurrence nets. In occurrence nets, the support set is split into two distinct sets: conditions and events representing, respectively, local states and (local) state changes.

The term “occurrence net” is applied to objects of two kinds: the first one represents a single history of a system (a run), while the second represents all of the system’s possible histories of execution. In the first case, the occurrence net cannot have branches while, in the second case, forward-oriented branches are allowed.

This work continues the one initiated in [2] on runs, possibly infinite in both directions, and adapts it to a class of occurrence nets allowing for forward branching representing the conflict between alternative histories of a system. In these nets, three relations between pairs of elements are possible: concurrency, causal dependence and conflict.

Under an assumption of local finiteness, a structure of closed sets obtained from the concurrency relation is defined and studied.

A different closure operator is then defined on the same set; this operator is defined by means of an iterative procedure, which starts from a set of concurrent conditions and applies the firing rule of Petri nets, both forward and backward, until no new events can be fired.
It is shown that the two notions of closed sets coincide if the underlying occurrence net is \textit{B-dense}. An occurrence net is B-dense if every maximal clique of the concurrency relation (a potential global state) intersects every \textit{trail} (a maximal sequential subnet).

It is shown that closed sets, ordered by set inclusion, form a \textit{complete algebraic orthomodular lattice}.

Besides stating some structural properties of closed sets, we show that, in B-dense occurrence nets, a trail either intersects a given closed set or its orthocomplement (but not both), thus suggesting to look at closed sets as at propositions of a non-classical logic. Furthermore, we show that this property actually characterizes B-dense occurrence nets.

\section{Preliminary definitions}

In this section we recall the definitions of the main objects of interest, namely orthomodular posets \cite{7} and occurrence nets \cite{9, 8, 3, 6}.

\subsection*{Partially ordered sets and orthomodular lattices}

A \textit{poset} is a set \( P \) with a partial order relation \( \leq \subseteq P \times P \). From \( \leq \) we derive – when they exist – the binary operators \( \land \) (\textit{meet}) and \( \lor \) (\textit{join}). For \( x, y \in P \), \( x \leq y \), \( [x, y] = \{ z \in P \mid x \leq z \leq y \} \) is the interval between \( x \) and \( y \).

An \textit{orthocomplemented poset} \( P = \langle P, \leq, 0, 1, (\cdot) \rangle \) is a poset \( \langle P, \leq \rangle \), bounded by a minimum \((0)\) and a maximum \((1)\) element and with a map \((\cdot) : P \to P\), such that: \( \forall x, y \in P, (x' \cdot y) = x \) and \( x \leq y \Rightarrow y' \leq x' \) and \( x \land x' = 0 \) and \( x \lor x' = 1 \). The map \((\cdot) : P \to P\) is called an \textit{orthocomplementation in} \( P \).

A lattice \( L \) is a poset in which for any pair of elements meet and join always exist. Furthermore, \( L \) is \textit{complete} when the meet and the join of any subset of \( L \) exist. An orthocomplemented lattice is also called an \textit{ortholattice}.

An \textit{orthomodular lattice} is an ortholattice \( L = \langle L, 0, 1, \leq, \land, \lor, (\cdot) \rangle \) in which the \textit{orthomodular law} holds: \( x \leq y \Rightarrow y = x \lor (y \land x') \).

\subsection*{Nets and occurrence nets}

A net is a triple \( N = (B, E, F) \) such that \( B \) and \( E \) are countable sets, \( B \cap E = \emptyset \) and \( F \subseteq (B \times E) \cup (E \times B) \). The preset and postset of \( x \in B \cup E \), denoted \( ^*x \) and \( x^* \), are defined by \( ^*x = \{ y \in B \cup E \mid (y, x) \in F \} \), and \( x^* = \{ y \in B \cup E \mid (x, y) \in F \} \), respectively; the neighbourhood of \( x \), denoted \( x^* \), is given by \( ^*x \cup x^* \).

The elements of \( B \) are called \textit{local states} or \textit{conditions}, the elements of \( E \) local changes of state or \textit{events}, and \( F \) is called \textit{flow relation}. We will use the standard graphical notation for nets.

Let \( N = (B, E, F) \) be a net, and \( x, y \in B \cup E \). Then \( x \) and \( y \) are in \textit{conflict}, denoted \( x \# y \), if there exist two distinct events \( e_x, e_y \in E \) such that \( e_x F^*x \), \( e_y F^*y \), and \( \text{cfs}(x, y) := ^*e_x \cap ^*e_y \neq \emptyset \).

Let \( \leq := F^* \) be a partial order, then two elements \( x \) and \( y \) are \textit{causally dependent}, denoted \( x \mathfrak{li} y \), if either \( x F^*y \) or \( y F^*x \). Conflicts are \textit{inherited}: \(( x \# y \text{ and } x \leq z \text{ and } y \leq w \) \Rightarrow w \# z. \( x \) and \( y \) are \textit{concurrent}, written
x co y, if neither x ≠ y nor x li y hold. A clique D of co ∪ id will be called a coset, where id denotes the identity relation on B ∪ E. A coset D such that D ⊆ B will be called a B-coset. An occurrence net is a net N = (B, E, F) such that for all b ∈ B, ∃b ≤ 1, F* is a partial order, the conflict relation is antireflexive, the minimal elements with respect to F* belong to B, and for all x ∈ B ∪ E, |{y ∈ B ∪ E : yF^+x}| < ∞.

In an occurrence net, the minimal nodes with respect to F* form a B-coset; equivalently, there is no event e ∈ E such that e = ∅. We obtain a poset (X, ≤) by defining X = (B ∪ E) and ≤ = F*. For any subset A of elements of an occurrence net N = (B, E, F), define \( \min(A) = \{x ∈ A : x \cap A = ∅\} \).

The definition of occurrence net implies that \( \forall x ∈ B ∪ E : |x| < ∞ \) and \( ∀x, y ∈ B ∪ E : ||x, y|| < ∞ \). In the following we assume also that all occurrence nets are condition bordered: every event e ∈ E in an occurrence net has at least one output condition, i.e. \( |e^*| ≥ 1 \) and, moreover, \( |e^*| < ∞ \)

Within an occurrence net, we have the following objects of interest ([3, 6]).

- A maximal clique \( λ ⊆ B ∪ E \) of li is a line.
- A maximal clique \( γ ⊆ B ∪ E \) of (co ∪ id) is a cut.
- A maximal clique \( τ ⊆ B ∪ E \) of (# ∪ li) is a trail.
- A maximal clique \( ρ ⊆ B ∪ E \) of (li ∪ co) is a run.

The interpretation for a line is as a, possibly infinite, history of a sequential process while cuts can be interpreted as system snapshots. Cuts composed exclusively by B elements are a special case; these cuts can be interpreted as maximal sets of system properties valid in a mutually independent way.

A trail, in which only the relations # and li occur, can be interpreted as the complete history of a sequential process, including all the possible alternatives. On the contrary, a run can be interpreted as an execution of the system since all of the choices are effectively solved. Runs induce conflict-free nets. A run \( ρ \) is called K-dense iff every line \( λ \) of \( ρ \) intersects each of \( ρ \)'s cuts [3]. An occurrence net N is called B-dense iff each of its runs is K-dense, or equivalently, iff for every trail \( τ \) and for every cut \( γ, τ ∩ γ ≠ ∅ \) [6].

## 3 Closure Operators on Occurrence Nets

Throughout this section, let N = (B, E, F) be a B-dense occurrence net.

For any A ⊆ B ∪ E, define \( A^⊥ = \{x ∈ B ∪ E : ∀y ∈ A : x co y\} \). Since the concurrency relation is symmetric and irreflexive, the operator (.)⊥⊥ defined by \( A⊥⊥ = (A⊥)⊥ \) is a closure operator on B ∪ E, and (.)⊥ is an orthocomplement. Moreover, define \( L(N) = \{A ⊆ B ∪ E : A = A⊥⊥\} \) as the set of closed sets of N, then \( L(N) = (L(N), ⊆, ∅, B ∪ E, (.)⊥) \) is a complete orthocomplemented lattice, where, in particular, \( ∅⊥⊥ = B ∪ E \) (see [4] and [5]).

Note that, for any A ⊆ B ∪ E, \( A⊥⊥ ∈ L(N) \).

Dynamically closed sets were introduced, for occurrence nets without conflicts, in [2]. Here, we extend the idea.

A ⊆ B ∪ E is dynamically closed if, for all e ∈ E:

i. \( \min(A) ⊆ B \) is a B-coset in N.
ii. $e \in A \Rightarrow e^* \subseteq A$;
iii. $e \subseteq A \Rightarrow e \in A$;
iv. $e^* \subseteq A \Rightarrow e \in A$.

Any dynamically closed set can be obtained by means of an iterative procedure, which justifies the view of dynamically closed sets as causally closed subprocesses. Starting from a B-coset, a set of pairwise concurrent conditions, the procedure adds all the elements reachable by applying the forward and backward firing rule.

In Figure 1 an application of the closure operator is represented starting from a B-coset $A$.

![Fig. 1. The closure of the B-coset $A$ on the left.](image)

Denote the set of dynamically closed sets of $N$ by $D(N)$. The intersection of an arbitrary family of dynamically closed sets is dynamically closed [1]. Together with the remark that the emptyset and $B \cup E$ are dynamically closed, this allows us to deduce that $D(N)$ is a complete lattice, denoted $\langle D(N), \subseteq, \emptyset, B \cup E \rangle$, where the meet operation coincides with intersection, and the join operation is defined as the dynamic closure of the set union. In general, the union of dynamically closed sets is not dynamically closed, but for directed families of subsets of $B \cup E$ [5].

Before discussing the characteristics of dynamically closed sets and closed sets, we state the following two main theorems, which show the relation between the two types of closed sets, and define the structure of their sets [1].

Any closed set is also dynamically closed, whereas in general the converse is not true. In the case of $B$-dense occurrence nets, the two closure coincide.

**Theorem 1.** [1] Let $N = (B, E, F)$ be a $B$-dense occurrence net, and $A \subseteq B \cup E$. Then $A \in D(N)$ if, and only if, $A \in L(N)$.

**Theorem 2.** [1] Let $N = (B, E, F)$ be a $B$-dense occurrence net. The lattice $\mathcal{L}(N)$ is a complete algebraic orthomodular lattice.

By Theorem 1, also $D(N)$ is a complete algebraic orthomodular lattice, if $N$ is $B$-dense.

Orthomodularity of $\mathcal{L}(N)$ is a consequence of the following Lemma, which highlights the orthomodular law.
Lemma 1. Let \( A_1 \in L(N) \), \( x \notin A_1 \) and \( H = (A_1 \cup \{x\})^{\perp\perp} \). Then \( H \cap A_1^{\perp} \neq \emptyset \).

Lemma 1 implies that any closed set can be made bigger only by adding at least an element concurrent to it, in other words: given \( A_1, A_2 \in L(N) \), if \( A_1 \subset A_2 \), then \( A_2 \cap A_1^{\perp} \neq \emptyset \). This corresponds to the orthomodularity of the ortholattice \( L(N) \).

Properties of dynamically closed sets

In this section we consider closed and dynamically closed sets of B-dense occurrence nets, and discuss their main properties.

Let \( A \in D(N) \) \((A \in L(N))\) then the following properties can be proved:

1. \( A \) is bordered by conditions.
2. \( A \) is convex, i.e.: if \( x, y \in A \) and \( x \leq y \), then \([x, y] \subseteq A\).
3. If \( A \) contains two conflicting elements, then it also contains their past history, up to the origin of the conflict, i.e.: if \( x, y \in A \), and \( x \neq y \), then \( \text{cfs}(x, y) \subseteq A \).
4. A closed set \( A \) and its orthocomplement \( A^{\perp} \) share the past, i.e.: \( \text{past}(A) = \text{past}(A^{\perp}) \), where the past of \( A \) is the set of elements which do not belong to \( A \) and precede at least one element in \( A \), i.e.: \( \text{past}(A) = \{x \in P \setminus A \mid x \leq a \text{ for some } a \in A\} \).

In general, a closed set and its orthocomplement do not share their future, whereas, if \( \neq \emptyset \), then a closed set and its orthocomplement do share their future ([2]).

Relations between closed sets and trails

Trails represent branching histories of sequential components of a system.

In general, it is possible that a trail crosses neither a closed set, nor its orthocomplement. However, if a net is B-dense, the following theorem holds, giving a characterization of B-density in terms of closed sets.

Theorem 3. \( N \) is B-dense if, and only if, for all \( A \in L(N) \), for any trail \( \tau \) of \( N \), \( \tau \cap A \neq \emptyset \) or \( \tau \cap A^{\perp} \neq \emptyset \).

The next theorem gives a characterization of the closure of cosets, related to the set of trails of the net. Let \( N = (B, E, F) \) be a B-dense occurrence net, and \( T \) the set of trails of \( N \). For each \( x \in B \cup E \), let \( T_x \) be the set of all trails passing through \( x \): \( T_x = \{\tau \in T \mid x \in \tau\} \); for \( H \subseteq B \cup E \), define \( T_H = \bigcup_{x \in H} T_x \).

Let \( A \) be a coset of \( N \), namely a set of pairwise concurrent elements. Then a point \( x \) belongs to the closure of \( A \) if, and only if, every trail which passes through \( x \) passes also through at least one point in \( A \).

Theorem 4. Let \( A \) be a coset of a B-dense occurrence net \( N \). Then

\[ \forall x \in B \cup E : x \in A^{\perp\perp} \iff T_x \subseteq T_A \]

This allows a more concrete interpretation of the closure operator, when applied to a B-coset. Suppose that an occurrence net \( N \) models a system of interacting sequential components. Then the net can be decomposed into a set of

Proceedings of ICTCS 2013
trails, each one modelling the possible alternative histories of one sequential component, where the conditions represent local states, and the events can represent either autonomous changes of state or interactions among several components. This decomposition of the net into trails is made of a subset of all the trails of \( N \) which is a minimal covering of \( N \).

Suppose that we “observe” a B-coset of \( N \), say \( \beta \), i.e. a set of independent properties representing some local states.

Then, the closure of \( \beta \), denoted \( \beta^{\perp\perp} \), can be characterized as follows: an element \( x \in B \cup E \) belongs to \( \beta^{\perp\perp} \) if, and only if, for every minimal covering, all the trails passing through \( x \) are also passing through one element of \( \beta \). In terms of interacting sequential components, an element \( x \in B \cup E \) belongs to \( \beta^{\perp\perp} \) if, and only if, for every interpretation of \( N \) in terms of interacting sequential components, \( x \) certainly belongs only to some sequential components observed in \( \beta \), i.e.: \( x \) certainly represents either a possible local state belonging to the components observed in \( \beta \), or a possible change of states involving only components observed in \( \beta \).

Acknowledgments

Work partially supported by MIUR - PRIN 2010-2011 grant H41J12000190001.

References

1. Luca Bernardinello, Carlo Ferigato, Stefan Haar, and Lucia Pomello. Closed sets in occurrence nets with conflicts. Accepted for publication on Fundamenta Informaticae, 2013.
An Introduction to Consistently Merging Trust-Networks with Bipolar Preferences

Stefano Bistarelli1,2, Simon N. Foley3, Francesco Santini1,4, and Francesco Vicino1

1 Department of Mathematics and Computer Science, University of Perugia, Italy
   [bista,francesco.santini,francesco.vicino]@dmi.unipg.it
2 IIT-CNR, Pisa, Italy
   stefano.bistarelli@iit.cnr.it
3 Department of Computer Science, University College Cork, Ireland
   s.foley@cs.ucc.ie
4 EPI Contraientes, INRIA Rocquencourt, France
   francesco.santini@inria.fr

Abstract. In this paper we study weighted trust networks, where each edge is associated with a score. We consider distrust relationships as well, allowing a user to positively or negatively rate other individuals in his web of acquaintances. Then, we propose a general method to compose two networks together, in order to merge the knowledge obtained by different interactions among the same (or distinct) individuals. A straightforward application is to exploit this critical merging whenever two trust-based communities need to be amalgamated.

1 Introduction and Motivations

In the last few years, the interaction among individuals has considerably leaned towards indirect relationships, through the use of Internet (or digital networks in general), and, more in detail, thanks to the new means of communication such as social networks, e-commerce sites and e-mail communications. Almost all social platforms of today, even if adopted for very different purposes, offer a mean to the user to somehow rate the information produced during previous interactions of different system-users. Therefore, in most of the cases, this kind of electronic communication occurs between entities/people who do not actually know each other, and for which it is not possible to establish a relationship of trust that has a solid foundation. In order to better capture this broad concept of trust, in this paper we refer to the definition inspired by McKnight & Chervany [6]: “Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible”. Therefore, trust plays a crucial role in computer mediated transactions and processes.

A Trust Network (TN), or Web of Trust, represents a social network that links some individuals according to the intended scope of the considered trust relationship (i.e., named trust scope), e.g. “to be a good car mechanic” if a user is looking for mechanic to repair her/his broken-down car. To be more precise, TNs actually are directed graphs rather than networks: nodes represent individual users, and, if we observe a relationship as “Individual X trusts Individual Y”, this results in an edge directed from node X to node Y.
Trust systems produce a score that reflects the relying party’s subjective view of an entity’s trustworthiness. This score can be derived from a combination of received referrals and personal experience. Moreover, transitivity is an explicit component in trust systems, and it allows each user to expand her/his limited and personal view by taking into account the referrals from acquaintances. However, sometimes the need for a composition between two (or more) TNs can arise. For instance, when two companies merge together into a single one, their two different TNs, representing their personal knowledge about (for example) their suppliers, has to be merged as well. In general, whenever two different (but identical in terms of trust scope) communities need to meet and join, this sort of knowledge fusion has to be accomplished in a proper way, also considering the conflicting opinions that are sparked from the different experiences in two separate communities.

Note that the framework presented in this paper is parametric to the chosen bipolar structure (see Sec. 2), and, by selecting crisp operators, it is possible to manage trust and distrust also in unweighted networks. In this case, for instance, \( \text{trust} \land \text{distrust} = \text{distrust} \).

## 2 Bipolar Preference Structures

We can use two different algebraic preference-structures [2] to represent both positive and negative (i.e., trust and distrust) weights in TNs: \( \langle P, +_P, \times_P, \perp_P, \top_P \rangle \) and \( \langle N, +_N, \times_N, \perp_N, \top_N \rangle \) respectively.

The structure we use to model negative preferences is exactly a c-semiring as originally described in [1]. In \( \langle N, +_N, \times_N, \perp_N, \top_N \rangle \), \( N \) is the set of negative preferences, \( +_N \) is commutative, associative, idempotent, \( \perp_N \) is its unit element, and \( \top_N \) is its absorbing element; \( \times_N \) is associative, commutative, distributes over \( +_N \), \( \top_N \) is its unit element and \( \perp_N \) is its absorbing element. Consider the relation \( \leq_N \) over \( N \) such that \( a \leq_N b \) iff \( a +_N b = b \). Then: \( \leq_N \) is a partial order; \( +_N \) and \( \times_N \) are monotone on \( \leq_N \); \( \perp_N \) is its minimum and \( \top_N \) its maximum; \( \langle A, \leq_N \rangle \) is a lattice and, \( \forall a, b \in A, a +_N b = \text{lub}(a, b) \). Informally, the relation \( \leq_N \) gives us a way to compare weights. An example of such structure is \( \langle \mathbb{R}^+, \text{min}, \text{sum}, +\infty, 0 \rangle \).

When dealing with positive preferences instead, we want two main properties to hold: combination should bring to better preferences, and indifference should be lower than all the other positive preferences. A positive preference structure is a tuple \( \langle P, +_P, \times_P, \perp_P, \top_P \rangle \) s.t. \( P \) is a set and \( \top_P, \perp_P \in P \); \( +_P \), the additive operator, is commutative, associative, idempotent, with \( \perp_P \) as its unit element (\( \forall a \in P, a +_P \perp_P = a \)) and \( \top_P \) as its absorbing element (\( \forall a \in P, a +_P \top_P = \top_P \)); \( \times_P \), the multiplicative operator, is associative, commutative and distributes over \( +_P \) (\( a \times_P (b +_P c) = (a \times_P b) +_P (a \times_P c) \)), with \( \perp_P \) as its unit element and \( \top_P \) as its absorbing element. The additive operator of this structure has the same properties as the corresponding one in c-semirings, and thus it induces a partial order over \( P \). On the other hand, \( \times_P \) has different properties w.r.t. \( \times_N \): the best element in the ordering \( \langle \top_P \rangle \) is now its absorbing element, while the worst element \( \langle \perp_P \rangle \) is its unit element. \( \perp_P \) models indifference. These are exactly the desired properties for the
combination and for indifference w.r.t. positive preferences. An example of a positive preference structure is \((\mathbb{R}^+, \max, \text{sum}, 0, +\infty)\).

To handle both positive and negative preferences, we merge these two structures in a single bipolar preference structure [2]. A bipolar preference structure is a tuple \((N, P, +, \times, \sqcup, \sqcap, \top, \bot)\), where \((N,+_N,\times_N,\sqcup_N,\sqcap_N,\top_N)\) is a c-semiring, and \((P,\oplus_P,\times_P,\sqcup_P,\sqcap_P,\top_P)\) a preference structure; \(\oplus: (N \sqcup P)^2 \rightarrow (N \sqcup P)\) is an operator s.t. \(a_N + a_P = a_P, \forall a_N \in N\) and \(a_P \in P\); it induces a partial ordering on \(N \sqcup P\): \(\forall a,b \in (N \sqcup P), a \leq b \iff a + b = b; \times_P: (N \sqcup P)^2 \rightarrow (N \sqcup P)\) is a commutative and monotone. An example of bipolar structure is \((N = [-1,0], P = [0,1], + = \max, \times = \hat{\times}, \sqcup = -1, \sqcap = 0, \top = 1)\) (where \(\hat{\times}\) is the arithmetic \(\times\)). In Sec. 3 we use \(\times\) as a compensation operator, to compose trust and distrust.

3 Managing Trust with Bipolar Preferences

A TN consists in a network that links individuals by establishing a relationship of trust among them. It is important to separate between trust in the ability to recommend a good car mechanic which represents referral trust [5], and trust in actually being a good car mechanic which represents functional trust [5]. A transitive trust path stops with the first functional trust arc encountered when there are no remaining outgoing referral trust arcs. It is, of course, possible for a principal to have both functional and referral trust in another principal, but that should be expressed as two separate trust arcs. In addition, in the following of this section we take care of representing distrust either. In this work we suppose trust as a transitive relationship, and we will exploit this property when composing two networks.

We consider a TN as represented by a graph \(G = (V, E)\). To represent when an individual \(x \in V\) trusts an individual \(y \in V\) we use a direct edge labeled with a weight \(w\). Formally, a trust relationship is represented as \(T(a, b) = w^+\), as visualized in Fig. 1; \(a, b \in V\), and \(w^+\) is a weight associated with the edge \(a \rightarrow b\). Since our goal is to represent also relationships of distrust as well, we add a weighted oriented edge if individual \(x \in V\) distrusts an individual \(y \in V\). Formally we represent it with \(T(a, b) = w^-\), visually represented in Fig. 2. To model these two dual metrics we adopt the bipolar structure presented in Sec. 2, i.e., \((N, P, +, \times, \sqcup, \sqcap, \top, \bot)\), where \((N, +_N, \times_N, \sqcup_N, \sqcap_N, \top_N)\).

Therefore, each \(w^+ \in P\), and they can be composed by using operator \(\times_P\), and the idempotent \(\times_P\) operator defines a partial order over the elements in \(P\). Finally, \(\top_P\) and \(\bot_P\) respectively represent the best and worst weight in \(P\), according to \(\oplus_P\). Specular considerations hold for the negative metrics (which is a c-semiring, see Sec. 2), e.g. \(w^- \in N\). The \(\times\) operator is used

![Fig. 1. Node a trusts node b with weight w.](image1)

![Fig. 2. Node a distrusts node b with weight w.](image2)
to solve possible conflicts that arise after the merge of two TNs. A possible instantiation of $\times$ in a crisp (i.e., unweighted) case is: $\text{Trust} \times \text{Distrust} = \text{Trust}$, $\text{Trust} \times \text{Distrust} = \text{Distrust}$, $\text{Trust} \times \text{Distrust} = \square$; we use the $\square$ value as a way to represent indifference, also in the weighted case.

A first result we want to obtain on weighted TNs is to apply transitive closure on them: after such step we are able to state if any node is reachable from another, i.e., if any node trust/distrusts any other node. In Prop. 1 and Prop. 2 we show how to respectively compose trust and distrust along a path, and, thus transitively extend a relationship.

**Proposition 1.** If the first edge along a path has an associated trust value of $w_1^+$, and the successive one has a trust value of $w_2^+$, then the derived edge has a trust value equal to $w_1^+ \times_P w_2^+$, as represented in the example in Fig. 3.

As an example, in this work we have decided to instantiate $\times_P$ by choosing the minimum weight of all the edges; therefore:

$$w_1^+ \times_P w_2^+ = \min(w_1^+, w_2^+) = \begin{cases} w_1^+ & \text{if } w_1^+ > w_2^+ \\ w_2^+ & \text{if } w_2^+ > w_1^+ \end{cases}$$

**Proposition 2.** If the first edge along a path has an associated trust value of $w_1^+$ and the successive one has a distrust value of $w_1^-$, then the derived edge has a distrust value equal to $w_1^+ \cdot w_1^-$, as represented in the example in Fig. 4.

The $\cdot$ operator is used to compose trust and distrust along the same path. It can be instantiated to different operators: in an unweighted network configuration, it classically extends the transitivity notion as we can find in literature, i.e., $(a \text{ trust } b) \cdot (a \text{ distrust } b) = true \cdot false = false$. In the weighted case, $w_r = w^+ \cdot w^-$ depends if we adopt an optimistic or a pessimistic view. The $\cdot$ operator can also be instantiated to the $\times$ in the chosen bipolar structure, but, in general, it can be different.

Clearly, after the transitivity-closure step, we may have introduced conflicts in a TN, since this step basically adds new edges to the graph, and these new edges can be in conflict with other pre-existent edges. We identify a conflict if $a \text{ trust } b$ and $a \text{ trust } b$ at the same time. Therefore, a successive step is needed to make a TN consistent again, that is conflict-free. Indeed, conflicts may arise only if we use trust and distrust edges.

This conflict removal leads also to the definition of “safe” in calling this procedure as “safe reconfiguration”. The safe reconfiguration procedure is composed of two consecutive steps: i) extending trust/distrust relationships between nodes by applying a transitive closure among edges, and ii) removing conflicts possibly introduced during step i)
Clearly, both steps exploit the operators defined in Sec. 2. Specifically, to accomplish conflict removal we take advantage of the compensation operator $\times$ that comes with our bipolar preference structure (see Sec. 2). If $w^+ \times w^- = w_p \in P$ we remove the distrust edge in our conflict, and we leave a trust edge with weight $w_p$. In the same way, if $w^+ \times w^- = w_p \in N$ we remove the trust edge from the conflict, and we leave the distrust edge with weight $w_p$. The third and last possible case corresponds to $w^+ \times w^- = \square$: in this case we remove both edges between the two nodes.

Note that $\times$ is, in general, different from the $\bullet$ operator we have introduced before in this section: $\times$ is used to compensate trust and distrust between two (or more) parallel paths, while $\bullet$ is used to compose trust and distrust along the same path.

A more complex example of conflict removal is reported in Fig. 5, where a possible conflict can arise due to the fact that we have two trust paths $(a \rightarrow c \rightarrow c$ and $a \rightarrow b \rightarrow c)$ and two distrust paths $(a \rightarrow f \rightarrow c$ and $a \rightarrow d \rightarrow c)$ between $a$ and $c$. We remove this conflict by first applying $+_P$ on trust paths, and $+_N$ on distrust paths; then, we use use our compensation operator $\times$ to aggregate these two values together. The whole computation is reported in Ex. 1. In the end, we check if the final result is in either in $P$ or $N$, such that we can decide if to add a trust or distrust edge between $a$ and $c$; if the result is equal to $\square$ we cannot take a sharp decision.

**Example 1.** $(\text{trusts b})_P (\text{trusts e})_P (\text{trusts c})_P (\text{trusts d})_N (\text{distrusts c})_N ((\text{trusts f})_N (\text{distrusts c})_N ((\text{trusts a})_P (\text{trusts b})_P (\text{trusts e})_P (\text{trusts c})_P (\text{trusts d})_N (\text{distrusts c})_N (w_2 P w_6) +_P (w_4 P w_3)) \times (w_5 \bullet w_8) +_N (w_4 \bullet w_7))$.

Therefore, all we need to model safe reconfiguration is a pipolar preference structure (together with its compensation operator) and an independent $\bullet$ operator; clearly, having this loose (but thoroughly formal) definition allows us to have a parametric computational framework, and to cast different instantiation of these operators which have only to respect the general reported properties.

## 4 Composing Two Trust Networks

As introduced in Sec. 1, our final goal is to be able to compose two different TNs with the same trust scope, in order to fuse information coming from distinct sources and to have a consistent result. Therefore, after

![Fig. 5. An example with multiple trust/distrust paths.](image-url)
having reconfigured each of the two networks through the procedure explained in Sec. 3, we can merge them: if $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$, $G_1 \cup G_2 = (N_1 \cup N_2, E_1 \cup E_2)$.

If this merge operation has led to inconsistencies (between trust and distrust edges between two adjacent nodes), we solve them by applying $\times$ as shown in Sec. 3. After this step, the new network has to be securely reconfigured as shown in Sec. 3. As a running example we consider the two networks a) and b) in Fig. 6, which we have been safely reconfigured in a previous step. To study this case, we consider the bipolar preference-structure $(N, P, +, \prec, -1, 0, 1)$, where $P = [0..1]$ and $N = [-1..0]$, $\times$ (i.e., the compensation operator) is equal to arithmetic sum, 0 is the worst trust level, and 1 is the best one. We consider the positive preference structure as $(P, +, \times, 0, 1)$, where $\times_P = \text{max}$. Moreover, $\bullet = \text{min}$. Figure 7 shows the network $G_2$ resulting from merging the two networks in Fig. 6.

After this step, we remove all the introduced conflicts: from Figure 7 we can see that the only conflict is between node $a$ and node $c$, since a trusts and distrusts $c$ at the same time. By obtaining $0 = -0.8 \times 0.8$ (thus, the indifference value), we can safely remove both edges and acquire the TN shown in Fig. 8.

Now we can start the safe reconfiguration proposed in Sec. 3. First we apply the transitive closure of trust/distrust edges. The transitive closure of a directed graph describes the reachability relationship between any two nodes. After the transitive closure is obtained, in an $O(1)$ operation one may determine if any node is reachable from any other node. One of the simplest techniques to find transitivity closure is Floyd-Warshall algorithm (worst case in $O(|N|^3)$).

Four sets of paths need to be checked during transitivity closure: i) $a \rightarrow d \rightarrow c$, ii) $b \rightarrow a \rightarrow d$, iii) $b \rightarrow a \rightarrow c$ (with $a \rightarrow c$ introduced in i)) and $b \rightarrow a \rightarrow d \rightarrow c$, and iv) $b \rightarrow d \rightarrow a$ (derived from $b \rightarrow a \rightarrow d \rightarrow a$). As an example, for path i) we add a new trust-edge between $a$ and $c$ with a weight of $0.4 \times 0.8 = 0.8$ ($\times_P \equiv \text{max}$), while for path iv) we add a new distrust edge.

![Fig. 6. Two TNs to be composed.](image)

![Fig. 7. The merging of Fig. 7.](image)

![Fig. 8. Figure 7 after a first inconsistency removal.](image)

![Fig. 9. Figure 8 after transitive closure.](image)
with a weight of $-0.4 \cdot 0.8 = -0.4$. In total, we obtain four new edges that are added to our intermediate TN (see Fig. 9).

As a final step, we need to again remove possible inconsistencies introduced after the transitive closure step. From Fig. 9 we notice we have two different conflicts, \(i\) between nodes \(a\) and \(b\), and \(ii\) between \(b\) and \(c\). For the sake of brevity, we only show the algebraic operation related to \(i\): in this case, we have \((b \text{ trusts } a) \times ((b \text{ trusts } d) \cdot (d \text{ distrusts } a))\), which leads us to \(0.6 \times (-0.4 \cdot 0.8) = 0.6 \times (-0.4) = 0.2\). Our final TN, obtained through merging the two networks in Fig. 6, is presented in Fig. 10.

5 Related Work

Both positive and negative ratings are quite common in today’s trust/reputation systems. For instance, such duality can be found in form of likes and dislikes in Internet Fora or Social Networks, in the rating of product reviews in e-commerce portals (as Amazon\(^5\)) or consumer review sites, as well as in the positive/negative ratings of sellers\(^6\), still in e-commerce systems. From this we can deduce that, in modern systems, distrust is widely recognised as a distinct concept from trust; modelling distrust by using a low trust-score is clearly less informative and less ties to human reasoning. Moreover, such “shifting” of distrust values to the positive axis would be sensitive to outliers and additionally distort the semantics of a zero score \(^7\). Therefore, having an explicit mean to also formally and mathematically evaluate even distrust is recommended. Indeed, the distrust concept applies also to a Web of Trust (instead of a centralised system), where parties are linked to trust/distrust relationships: in Epinions\(^7\), for example, users can explicitly trust or distrust other product reviewers. From their website we can quote the following sentence: “Just as the Web of Trust promotes the work of those members you trust, the Block List makes it less likely that you will encounter contributions you do not value in the future”.

While providing a computational framework to model propagation of trust scores has been thoroughly investigated in literature \([5,3]\), a few works deal with the propagation of trust and distrust at the same time.

In \([7]\) the authors model trust as a value in the interval \([0..1]\), while they represent distrust as a score in the interval \([-1..0]\). Trust/distrust evaluation

---

\(^5\) http://www.amazon.com
\(^6\) http://www.ebay.com
\(^7\) http://www.epinions.com
are represented in two separate matrices of values $T$ and $D$: $T_{i,j}$ represents how much $i$ trusts $j$, while $D_{i,j}$ how much $i$ distrusts $j$. The mathematical operations consists in subtracting one matrix from another, i.e., $R = T - D$; thus, $R$ stores the result scores.

In [4], the authors associate a weight of 1 to all trust edges, and of $-1$ to all distrust edges. In order to compute a final result between $i$ and $j$, all paths leading from $i$ to $j$ are considered, and their weights summed together. In the end, a positive result means trust, while a negative means distrust between the two entities.

### 6 Conclusion

In this paper we have summarised our preliminary ideas towards a general framework to compose TNs. The goals of the proposed framework are, i) to use both a metrics of trust and of distrust in the same framework, ii) to associate the strength of a trust/distrust relationship with a weight, thus obtaining a quantitative tool, and, finally, iii) to offer a system as more general as possible, offering a parametric framework open to different trust metrics.

### References

Language theoretic approach to synchronizing automata

(Extended Abstract)

Vladimir V. Gusev¹, Marina I. Maslennikova¹, Elena V. Pribavkina¹, Emanuele Rodaro²

¹ Institute of Mathematics and Computer Science
Ural Federal University, Ekaterinburg, Russia
² Centro de Matemática, Faculdade de Ciências
Universidade do Porto, 4169-007 Porto, Portugal
vl.gusev@gmail.com, maslennikova.marina@gmail.com, elena.pribavkina@usu.ru, emanuele.rodaro@fc.up.pt

Let $A = (Q, \Sigma, \delta)$ be a deterministic finite automaton (DFA for short), where $Q$ is the state set, $\Sigma$ stands for the input alphabet, and $\delta : Q \times \Sigma \rightarrow Q$ is the transition function defining an action of the letters in $\Sigma$ on $Q$. When $\delta$ is clear from the context, we will write $q.w$ instead of $\delta(q, w)$ for $q \in Q$ and $w \in \Sigma^*$. In the theory of formal languages the definition of a DFA usually includes the set $F \subseteq Q$ of terminal states and the initial state $q_0 \in Q$. We will use this definition when dealing with automata as devices for recognizing languages. The language $L \subseteq \Sigma^*$ is recognized (or accepted) by an automaton $A = (Q, \Sigma, \delta, F, q_0)$ if $L = \{w \in \Sigma^* | \delta(q_0, w) \in F\}$. We also use standard concepts of the theory of formal languages such as regular language, minimal automaton, etc. [6]

A DFA $A = (Q, \Sigma, \delta)$ is called synchronizing if there exists a word $w \in \Sigma^*$ which leaves the automaton in unique state no matter at which state in $Q$ it is applied: $q.w = q'.w$ for all $q, q' \in Q$. Any word $w$ with such property is said to be synchronizing (or reset) word for the DFA $A$. For the last 50 years synchronizing automata received a great deal of attention. For a brief introduction to the theory of synchronizing automata we refer the reader to the recent surveys [8,9].

Recently in a series of papers [2–5, 7] a language theoretic approach to the study of synchronizing automata was developed. In this abstract we summarize known facts from above papers and present some new results. We denote by $\text{Syn}(A)$ the language of synchronizing words for a given automaton $A$. It is well known that $\text{Syn}(A)$ is regular [9]. Furthermore, it is an ideal in $\Sigma^*$, i.e. $\text{Syn}(A) = \Sigma^* \text{Syn}(A) \Sigma^*$. On the other hand, every ideal language $L$ serves as a language of synchronizing words for some automaton. For instance, the minimal automaton of the language $L$ is synchronized by $L$ [5]. Thus, synchronizing automata can be considered as a special representation of ideal languages. Effectiveness of such a representation was addressed in [5]. The reset complexity $rc(L)$ of an ideal language $L$ is the minimal possible number of states in a synchronizing automaton $A$ such that $\text{Syn}(A) = L$. Every such automaton $A$ is called minimal synchronizing automaton (for brevity, MSA). Let $sc(L)$ be the number of states in the minimal automaton recognizing $L$. Then for every ideal language $L$ we have $rc(L) \leq sc(L)$ (since the minimal automaton

Proceedings of ICTCS 2013
is synchronized by \( L \). Moreover, there are languages \( L_n \) for every \( n \geq 3 \) such that \( rc(L_n) = n \) and \( sc(L_n) = 2^n - n \), see [5]. Thus, representation of an ideal language by means of a synchronizing automaton can be exponentially smaller than “traditional” representation via minimal automaton. However, no reasonable algorithm is known for computing MSA of a given language. One of the obstacles is that MSA is not uniquely defined. For instance, there is a language with at least two different MSA’s: one of them is strongly connected, another one has a sink state [5]. Therefore, some refinement of the notion of MSA seems to be necessary. Another important observation is the following: minimal synchronizing automata for the aforementioned languages \( L_n \) are strongly connected. Thus, one may ask, whether there always exists a strongly connected MSA for an ideal language. More generally, does there exist a strongly connected synchronizing automaton whose language of reset words coincides with a given ideal? It is rather easy to see that for the unary language \( L = a^* \) the answer to the latter question is negative. In case \( |\Sigma| > 1 \) it was recently shown in [7], that such a strongly connected automaton does exist for any ideal with at least two letters. In [2] it was shown, that a strongly connected MSA does not always exist. In particular, the following theorem is proved.

**Theorem 1.** Let \( \Sigma = \{a, b\} \). There is unique up to isomorphism strongly connected synchronizing automaton \( \mathcal{B} \) such that \( \text{Syn}(\mathcal{B}) = \Sigma^{\geq n} \).

The automaton \( \mathcal{B} \) is actually De Brujin automaton for the words of length \( n \). Thus, it has \( 2^n \) states. On the other hand, obviously we have \( rc(L) = sc(L) = n + 1 \). Thus the smallest strongly connected automaton with a given language \( L \) as the language of synchronizing words may be exponentially larger than an MSA of \( L \).

Another source of motivation for studying representations of ideal languages by means of synchronizing automata comes from the famous Černý conjecture [1]. Černý already in 1964 conjectured that every synchronizing automaton possesses a synchronizing word of length at most \( (n - 1)^2 \). Despite intensive efforts of researchers this conjecture is still widely open. We can restate the Černý conjecture in terms of reset complexity as follows: if \(|L|\) is the minimal length of words in an ideal language \( L \) then \( rc(L) \geq \sqrt{|L|} + 1 \). Actually even a lower bound \( rc(L) \geq \sqrt{|L|}/c \) for some constant \( c > 0 \) would be a major breakthrough for this conjecture. Thus, we hope that deeper understanding of reset complexity will bring us new ideas to resolve this longstanding conjecture. Additional motivation for studying strongly connected automata comes from the well-known fact, that the Černý conjecture holds true whenever it holds true for strongly connected automata. Note, that although in [7] the principal existence of a strongly connected synchronizing automaton with a given language of reset words is established, the construction itself is non-trivial, rather technical, and the upper bound on the number of states of associated strongly connected automaton is very big.

**Theorem 2.** Let \( I \) be an ideal language such that \( I^R \) (the ideal obtained applying the reversal operator) has state complexity \( n \). Then there is a strongly

---

Proceedings of ICTCS 2013
connected synchronizing automata $\mathcal{B}$ with $N$ states and $\text{Syn}(\mathcal{B}) = I$ such that:

$$N \leq m^k 2^n \left( \sum_{i=2}^{n} m_i \right)^2$$

where $k = |\Sigma|$ and $m = \left( \frac{n^2+n}{2} + 1 \right)$.

Thus, the questions whether there is a more effective construction, and whether it is possible to find a more precise upper bound, remain open. The first attempts to approach these questions were made in [3], where the partial case of principal ideal languages, i.e. the languages of the form $\Sigma^* w \Sigma^*$, is considered. Namely, the following theorem holds.

**Theorem 3.** For the language $\Sigma^* w \Sigma^*$ there is a strongly connected automaton $\mathcal{B}$ with $|w| + 1$ states, such that $\text{Syn}(\mathcal{B}) = \Sigma^* w \Sigma^*$. Such an automaton can be constructed in $O(|w|^2)$ time.

Recently we proved that the automaton $\mathcal{B}$ from the above theorem is in fact an MSA. More precisely, we have the following theorem.

**Theorem 4.** Let $I = \Sigma^* w \Sigma^*$ be a principal ideal language, then $\text{rc}(I) = |w| + 1$.

In case of a finitely generated ideal, i.e. a language of the form $L = \Sigma^* S \Sigma^*$, where $S$ is a finite set of words, the following result is proved in [2]. To state this result recall, that a set of words is anti-factorial, if no word in this set is a factor of another word in this set. Note, that we may assume without loss of generality, that $S$ is anti-factorial.

**Theorem 5.** Let $S$ be finite and anti-factorial set of words in $\Sigma^+$. There is a strongly connected synchronizing automaton $\mathcal{C}_S$ such that $\text{Syn}(\mathcal{C}_S) = \Sigma^* S \Sigma^*$. This automaton has at most $2^n$ states, where $n = \max \{|s| \mid s \in S\}$.

Principal left ideals, i.e. ideals of the form $\Sigma^* w$ for some word $w$, seem to play an important role in the Černý conjecture and the theory of synchronizing automata. Indeed, we characterize strongly connected automata via homomorphic images of automata belonging to a particular class of automata recognizing languages of the form $w^{-1} \Sigma^* w$ for some $w \in \Sigma^*$. Namely, consider the class $\mathcal{L}(\Sigma)$ of all trim automata $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, \{q_0\} \rangle$ such that $L(\mathcal{A}) = w^{-1} \Sigma^* w$ for some word $w \in \Sigma^*$. The following property holds.

**Property 1.** Let $\mathcal{A} \in \mathcal{L}(\Sigma)$ with $L(\mathcal{A}) = w^{-1} \Sigma^* w$. Then $\mathcal{A}$ is a strongly connected synchronizing automaton with $w \in \text{Syn}(\mathcal{A})$.

We recall that a homomorphism $\varphi : \mathcal{A} \to \mathcal{B}$ of automata is a map between the sets of states preserving the action of the two automata. Similarly, a congruence is an equivalence relation on the set of states which is compatible with the action of the letters. We have the following theorem.

**Theorem 6.** Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a strongly connected synchronizing automaton. For any synchronizing word $w$ of minimal length there is a DFA $\mathcal{B} \in \mathcal{L}(\Sigma)$ with $L(\mathcal{B}) = w^{-1} \Sigma^* w$ and

$$\Sigma^* w \Sigma^* \subseteq \text{Syn}(\mathcal{B}) \subseteq \text{Syn}(\mathcal{A})$$
such that \( \mathcal{A} \) is a homomorphic image of \( \mathcal{B} \).

**Corollary 1.** The class of strongly connected synchronizing automata are all and only all the homomorphic images of the class \( \mathcal{L}(\Sigma) \) formed by the trim automata \( \mathcal{A} = (Q, \Sigma, \delta, \{q_0\}, q_0) \) such that \( L(\mathcal{A}) = w^{-1}\Sigma^*w \) for some word \( w \in \Sigma^* \).

**Proof.** By Property 1 we have that any \( \mathcal{A} \in \mathcal{L}(\Sigma) \) is a strongly connected synchronizing automata, hence any homomorphic image \( \varphi(\mathcal{A}) \) is also a strongly connected synchronizing automaton. On the other hand, by Theorem 6 any strongly connected synchronizing automaton is a homomorphic image of a DFA from \( \mathcal{L}(\Sigma) \).

By \( \text{Cong}_k(\mathcal{B}) \) we mean the set of all congruences of automaton \( \mathcal{B} \) of index \( k \). Using Theorem 6 we can give another reformulation of the Černý conjecture using the automata from \( \mathcal{L}(\Sigma) \).

**Theorem 7.** Černý’s conjecture holds if and only if for any \( \mathcal{B} \in \mathcal{L}(\Sigma) \) and \( \rho \in \text{Cong}_k(\mathcal{B}) \) for all \( k < \sqrt{\|\text{Syn}(\mathcal{B})\| + 1} \) we have

\[
\|\text{Syn}(\mathcal{B}/\rho)\| < \|\text{Syn}(\mathcal{B})\|
\]

**Acknowledgement**

The work is partially supported by the Presidential Program for young researchers, grant MK-266.2012.1, and by the Russian Foundation for Basic research, grant 13-01-00852. The last author acknowledges the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT – Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011 and SFRH/BPD/65428/2009.

**References**


**Strong prefix codes of pictures**

Marcella Anselmo, Dora Giammarresi, Maria Madonia

1 Dipartimento di Informatica, Università di Salerno, Via Giovanni Paolo II, 132 I-84084 Fisciano (SA) Italy. E-mail: anselmo@dia.unisa.it
2 Dipartimento di Matematica, Università di Roma “Tor Vergata”, via della Ricerca Scientifica, 00133 Roma, Italy. E-mail: giammarr@mat.uniroma2.it
3 Dipartimento di Matematica e Informatica, Università di Catania, Viale Andrea Doria 6/a, 95125 Catania, Italy. E-mail: madonia@dmi.unict.it

**Extended Abstract**

The notion of codes in two dimensions is an interesting subject for researchers both from theoretical and applicative side due to the important role that images have nowadays in human communications. The aim is to generalize to 2D the well established theory of string codes (see [8] for a complete reference).

In the last two decades, two dimensional codes were studied in different contexts and it were defined polyomino codes, picture codes, and brick codes. A set $C$ of polyominoes is a code if every polyomino that is tilable with (copies of) elements of $C$, it is so in a unique way. Most of the results show that in the 2D context we loose important properties. A major result due to D. Beauquier and M. Nivat states that the problem whether a finite set of polyominoes is a code is undecidable, and the same result holds also for dominoes ([7]). Related particular cases were studied in [1]. In [13] codes of directed polyominoes equipped with catenation operations are considered, and some special decidable cases are detected. Codes of labeled polyominoes, called bricks, are studied in [14] and further undecidability results are proved.

As major observation, remark that all mentioned results consider 2D codes independently from a 2D language theory. The first attempt to connect 2D codes and 2D language theory was presented, in [9], by S. Bozapalidis et al. that considered picture languages. Recall that a picture is a rectangular array of symbols from a finite alphabet $\Sigma$. The size of a picture is a pair $(m,n)$ corresponding to its number of rows and columns, respectively. The set of all pictures over $\Sigma$ is denoted by $\Sigma^{**}$, while $\Sigma^{++}$ denotes the set of all non-empty pictures. A picture language (or 2D language) is a subset of $\Sigma^{**}$. Two partial concatenation operations can be defined between pictures. They are referred to as horizontal and vertical concatenation, and denoted by $\boxdot$ and $\boxplus$, respectively: pictures to be concatenated need to have same number of columns or rows, respectively. Using horizontal and vertical concatenation operations, doubly-ranked monoids are introduced and picture codes are studied in order to extend

---

* Partially supported by MIUR Project “Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali”, by 60% Projects of University of Catania, Roma “Tor Vergata”, Salerno.
syntactic properties to two dimensions [9]. Unfortunately many results are again negative and involve undecidability issues. Even the definition of prefix picture codes in [12] does not lead to any wide enough class.

Very recently, in [4], a new definition of picture codes was introduced in relation to the family REC of picture languages recognized by tiling systems (see [10, 11]). Instead of referring to horizontal and vertical concatenation, the operation of tiling star is considered: the tiling star of a set \(X\), as defined in [15], is the set \(X^{**}\) of all pictures that are tilable (in the polyominoes style) by elements of \(X\). The way to obtain a picture by composing pictures of \(X\) is called a tiling decomposition of the picture. Then \(X\) is a code if any picture in \(\Sigma^{**}\) is tilable in at most one way. Remark that (as well as in the one-dimensional case) when \(X\) is a code, \(X\) cannot contain an empty picture. Hence, in the following, we will consider only sets \(X \subseteq \Sigma^{+*}\); therefore also \(X^{**} \subseteq \Sigma^{++}\). Let us show some simple examples of codes over the alphabet \(\Sigma = \{a, b\}\).

**Example 1.** Let \(X = \{\begin{array}{c} a \ b \ a \\ b \ a \ a \end{array}, \begin{array}{c} a \ a \ a \\ a \ a \ a \end{array}\}\). It is easy to see that \(X\) is a code. Any picture \(p \in \Sigma^{**}\) can be decomposed starting at top-left corner and checking the sub-picture of size \((2, 2)\): it can be univocally decomposed in \(X\). Then, proceed similarly for the next contiguous sub-pictures of size \((2, 2)\).

**Example 2.** Let \(X = \{\begin{array}{c} a \ b \ a \\ a \ b \ a \end{array}, \begin{array}{c} b \ a \ a \\ a \ a \ a \end{array}\}\). Notice that \(X\) is not a code. Indeed picture \(\begin{array}{c} a \ b \ a \\ a \ b \ a \end{array}\) has the two following different tiling decompositions in \(X\): \(t_1 = \begin{array}{c} a \ b \ a \\ a \ b \ a \end{array}\) and \(t_2 = \begin{array}{c} a \ b \ a \\ a \ b \ a \end{array}\).

Remark that if \(X \in REC\) then \(X^{**}\) is also in \(REC\). By analogy to the string case, it holds that if \(X\) is a finite picture code then, starting from pictures in \(X\) we can easily construct an unambiguous tiling system for \(X^{**}\) (see [4] and [5] for the definition of unambiguity). Unfortunately, despite this nice connection to the string code theory, it is proved that it is still undecidable whether a given set of pictures is a code. This is actually coherent with the known result of undecidability for unambiguity inside \(REC\).

Looking for decidable subclasses of picture codes, in [4] the definition of prefix code is proposed. Pictures are then considered with a preferred scanning direction: from top-left corner to the bottom-right one. Then a picture \(p\) is a prefix of a picture \(q\), if \(p\) coincides with the “top-left portion” of \(q\). Observe that it is not convenient to define a set \(X\) to be prefix, by merely imposing that its pictures are not mutually prefixes: this would not automatically imply that \(X\) is a code. The basic idea in defining a prefix code is to prevent the possibility to start decoding a picture in two different ways (as it is for the prefix string codes). One major difference going from 1D to 2D case is that, while any initial part of a decomposition of a string is still a string, the initial part of a decomposition of a picture has not necessarily a rectangular shape: it is in general a (labeled) polyomino. Hence a notion related to tiling, and referred to as covering, is introduced. Informally a picture \(p\) is covered by (pictures in
a set) $X$, if $p$ can be tiled with pictures that possibly “exit” $p$ throughout the bottom and the right border. For example, in the figure below, the picture with thick borders is (properly) covered by the others. Refer to [4] for the formal definition.

Then the definition of prefix set given in [4] is equivalent to the following one.

**Definition 1.** A set $X$ is prefix if and only if every $x \in X$ cannot be properly covered by pictures in $X$.

In [4] it is proved that it is decidable whether a finite set of non-empty pictures is a prefix set and that, as in the 1D case, every prefix set of pictures is a code. Moreover a polynomial time decoding algorithm for finite prefix codes is presented. Prefix codes for pictures inherit several properties from the original family of prefix string codes and several non-trivial examples can be exhibited. Nevertheless it is worth to say that the definition is sometimes difficult to manage, since the presence of a specific picture in the prefix set depends on a tiling combination of (possibly) many other pictures in the same set.

In this paper we take back the definition of prefix set for strings and generalize the notion to 2D in a different way. This definition can be viewed as a more direct generalization from the notion of prefix sets for strings: we will refer to it as strong prefix. Despite the notion will correspond to a smaller family of codes, such family has many remarkable properties that generalize the theory of codes from one to two dimensions.

We first specialize the definition of “picture $p$ prefix of picture $q$” in the particular cases when $p$ and $q$ have the same number of rows (columns, resp.): in this case $p$ will correspond to a left (top, resp.) portion of $q$. Here below there is the formal definition.

**Definition 2.** Let $p,q \in \Sigma^+$. Picture $p$ is a horizontal prefix of $q$, denoted by $p \leq_h q$, if there exists $x \in \Sigma^*$ such that $q = px$. Picture $p$ is a vertical prefix of $q$, denoted by $p \leq_v q$, if there exists $y \in \Sigma^*$ such that $q = py$. If $x$ (y, resp.) is not empty the horizontal (vertical, resp.) prefix is proper.

Then, combining the previous two definitions of horizontal and vertical prefix for pictures, we generalize the notion of prefix set from strings to pictures, as follows.

**Definition 3.** Let $X \subseteq \Sigma^+$. $X$ is strong prefix if there is no picture in $X$ that is prefix of another picture in $X$ and, moreover, for any two different pictures $p$ and $q$ in $X$, there does not exist any picture $x \in \Sigma^+$ such that $x \leq_h p$ and $x \leq_v q$. 

---

*Proceedings of ICTCS 2013*
For example, the following two pictures $p$ and $q$ cannot both belong to a strong prefix set:

\[
\begin{array}{c}
\text{a b} \\
\text{a a}
\end{array}
\quad
\begin{array}{c}
\text{a b a a} \\
\text{a a}
\end{array}
\quad
\begin{array}{c}
\text{a b a a}
\end{array}
\]

$p$ and $q$ “overlapped”

Let us give some examples.

**Example 3.** Let $X = \{ \text{aba}, \text{abb}, \text{aa}, \text{ab}, \text{ba}, \text{bb} \}$. Language $X$ is strong prefix: no two pictures in $X$ overlap on their top-left corner.

**Example 4.** Let $X = \{ \text{aa}, \text{bb}, \text{bb}, \text{aba}, \text{aba}, \text{a a b b}, \text{b a b b}, \text{b b b}, \text{a a b} \}$. Language $X$ is strong prefix.

Remark that strong prefix sets are a proper subclass of prefix sets (as introduced in [4]): they still generalize the definition of prefix set from string to pictures and their definition is simpler to handle with. Let us state our main results on strong prefix sets (see [4]). Some results refer to maximal strong prefix sets, i.e. strong prefix sets not properly contained in another strong prefix set.

**Proposition 1.** If $X \subseteq \Sigma^+$ is strong prefix then $X$ is a code.

**Proposition 2.** It is decidable whether a finite strong prefix set $X \subseteq \Sigma^+$ is maximal strong prefix.

Let $X$ be a strong prefix set. By applying the definition, one can show that if $X$ is maximal prefix then it is maximal strong prefix. Moreover the following result holds.

**Proposition 3.** Let $X \subseteq \Sigma^+$ be a finite strong prefix set. Then it is possible to construct a finite set $Y \subseteq \Sigma^+$ such that $Y$ is maximal strong prefix and $X \subseteq Y$.

In 1D, given a finite prefix code, there exists a unique maximal finite code that contains it, and that is minimum both in cardinality and in the total length of its strings. We ask whether a similar situation holds in 2D. The setting looks like more involved, since pictures can “extend” both horizontally and vertically. Surprisingly, a result “similar” to the one-dimensional case holds.

Define the area of a picture of size $(m, n)$ as $m \times n$, and the size of a finite picture language $X$, denoted $size(X)$, as the sum of the areas of its pictures.

**Proposition 4.** Let $X \subseteq \Sigma^+$ be a finite strong prefix set. There exists a unique finite maximal strong prefix set $Y \subseteq \Sigma^+$ that contains $X$ and has minimum size. Moreover $Y$ is minimum in cardinality too.
In 1D the notion of maximality coincides with that of (right-) completeness for thin (prefix) codes. We compare the two notions for strong prefix two-dimensional codes.

A set $X \subseteq \Sigma^{++}$ is bottom-right-complete if every $p \in \Sigma^{**}$ can be covered by (pictures in) $X$.

**Proposition 5.** Let $X \subseteq \Sigma^{++}$ be a strong prefix code. If $X$ is bottom-right-complete then it is strong prefix maximal.

The vice versa does not hold, as shown by the following example.

**Example 5.** Let $\Sigma = \{a, b\}$ and let $Y = \{ \begin{array}{c} a \ b \ a \\ a \ a \ \\ b \ a \\ a \ b \ \\ b \ b \ \\ a \ a \ \\ b \ b \ \\ a \ b \ \\ b \ b \ \\ a \ b \ \\ b \ b \ \\ a \ a \ \\ b \ a \ \end{array} \}$. $Y$ is a maximal strong prefix set. Let us show that $Y$ is not bottom-right-complete. Consider the picture $p = \begin{array}{c} b \ b \ a \\ b \ b \ b \ x \ \\ a \ b \ y \ z \ \end{array}$ with $x, y, z \in \Sigma$: $p$ cannot be covered with pictures in $Y$. Indeed, from a careful analysis of possible compositions of pictures in $Y$, it follows that the symbol $b$ in position $(2, 3)$ of $p$ cannot be tiled by pictures in $Y$.

The following hierarchy summarizes the results on families of 2D finite codes obtained in this paper and in our previous work [4]:

Prefix maximal codes $\subseteq$ Strong prefix maximal codes

$\subseteq$ Strong prefix codes $\subseteq$ Prefix codes

$\subseteq$ Codes.

We let open the problem whether (finite) prefix maximal codes are equals to (finite) strong prefix maximal codes or not. As future research, we will try to remove the finiteness hypothesis and consider prefix sets belonging to particular sub-families in REC, such as deterministic ones ([3, 6]).

More details on these results can be found in [2].

**References**


Towards a linear contract logic

Massimo Bartoletti$^1$, Paolo Di Giamberardino$^1$, and Roberto Zunino$^2$

$^1$ Università degli Studi di Cagliari, Italy — bart, digiambe@unic.it
$^2$ Università degli Studi di Trento, Italy — roberto.zunino@unitn.it

Abstract. We introduce a linear logic for contracts. The logic (called PwLLW) extends intuitionistic linear affine logic ILLW with a contractual implication connective, along the lines of Propositional contract Logic (PwL). A proof system for PwLLW is presented, and it is shown sound and complete with respect to a phase structure model. By exploiting the finite model property, we show that PwLLW is decidable.

1 Introduction

Propositional Contract Logic (PCL) was introduced in [4] as an extension of intuitionistic propositional logic IPC with a new connective \( \rightarrow \), called contractual implication. Its aim was that of resolving circular dependencies among offers and requests in contractual clauses. The archetypical example is that of, say, a buyer offering \( B \) in exchange of the promise of obtaining \( A \), and a seller offering \( A \) in exchange of a promise of \( B \). In PCL, these contracts lead the two participants to an agreement, modelled by the theorem:

\[
(A \rightarrow B) \land (B \rightarrow A) \vdash A \land B
\]

Of course, an agreement would have been possible also if one of the participants (say, the buyer) were just offering \( B \) without asking anything in exchange. In such case, the buyer can wait for the other doing the first step, before doing \( A \):

\[
B \land (B \rightarrow A) \vdash A \land B
\]

The formal justification for the connective \( \rightarrow \) was then given in [3,1], where it is shown that, in the presence of circular offer-request constraints, either the participants reach an agreement, or some of them is not protected by her contract. Intuitively, the contract \( B \) does not protect the buyer, because it just dictates him to do \( B \), without asking anything in exchange; instead, \( A \rightarrow B \) protects him, by stating that \( B \) is an obligation only if \( A \) is guaranteed to happen, eventually.

A proof system for PCL was defined in [4] in terms of Gentzen-style rules, which extend those of IPC; decidability of PCL was then established by showing that the proof system enjoys cut elimination and the subformula property.

PCL is a suitable model for contracts when the atomic entities of the logic represent events, which can only occur once (as e.g., in event structures [11]). Were atoms used to model resources, which are subject to linearity constraints, PCL would no longer be adequate. For instance, there is no obvious way to model a situation where two occurrences of resource \( A \) have to be consumed to produce an occurrence of \( B \).
Towards a linear contract logic

Linear logic \[\text{[l]}\] has been described as a \textit{resource-aware} logic \[\text{[fe]}\]. This is evident by the absence of the structural rules of weakening and contraction, which otherwise would allow for free duplication/elimination of resources. It then seems a viable idea that of studying the circularity issues addressed by PCL in the setting of linear logic.

In this paper we start developing this idea, by extending intuitionistic linear affine logic ILLW (which stands for Intuitionistic Linear Logic with Weakening) with a new connective \(\rightarrow\), playing the same role as contractual implication in PCL. Our logic, called Propositional Contract Linear Logic with Weakening (PCLLW), allows for a sound encoding of PCL (Th. 1), and it enjoys some desirable structural properties, e.g., cut elimination (Th. 2). In §3 we combine results of Ciabattoni, Okada and Terui \[\text{[12,6]}\] to construct a sound and complete model of PCLLW, in the form of affine phase structures (Th. 3 and Th. 4). Along the same lines of \[\text{[12,6]}\], we show that PCLLW enjoys the finite model property (Th. 5), and we exploit this fact to prove it decidable (Th. 6).

An open question is whether, neglecting the nice properties it enjoys, PCLLW exactly captures the intuitively valid properties of contracts. We discuss some possible issues in §4.

2 Linear affine contract logic

The logic PCLLW extends ILLW with a connective \(\rightarrow\), which is the linear version of \(\rightarrow\) in PCL. Intuitively, we may interpret the linear contractual implication \(A \rightarrow B\) as “I will provide the resource \(B\) if the resource \(A\) at some point becomes available”. A sequent calculus for PCLLW is presented in Fig. 1:

- Rule \text{ZERO} states that provable formulae are contractually implied; intuitively, if one can produce a resource \(B\), then he can impose a condition over \(B\) (“I will provide \(B\) if \(A\) will be eventually available”).
- Rule \text{PREPost} provides \(\rightarrow\) with the same weakening properties of \(\rightarrow\). If from \(A\) one produces \(C\), and from \(D\) one produces \(B\), then if one provides \(D\) whenever \(C\) is available, one provides \(B\) whenever \(A\) is available.
- The \text{FIX} rule, which is the left rule for \(\rightarrow\), is the crucial one. Compared to the rule \(\rightarrow\text{l}\) of ILL, there are two differences: first, in the leftmost premise we allow for using the consequence \(B\) of a contractual implication \(A \rightarrow B\); second, we use the same context \(\Gamma\) in both premises. Intuitively, assume that one can obtain \(C\) from \(B\), and that \(A\) can be obtained from \(B\); then if we assume \(B\), under the condition that \(A\) will eventually be available (and it will be since from \(B\) we can produce \(A\)), \(C\) can be obtained.

We now discuss some relevant properties of PCLLW.
Fig. 2. Translation of PCL formulae into PCLLW formulae.

1. \((A \rightarrow B) \otimes (B \rightarrow A) \vdash A \otimes B\). This is the fundamental property of the system: it represents the handshake between two contracts, yielding both \(A\) and \(B\) at the same time. We stress that, in LL, from \(A \rightarrow B\) and \(B \rightarrow A\), \(A \otimes B\) cannot be deduced. For instance, if \(A\) corresponds to the resource “one slice of pizza” and \(B\) corresponds to “one euro”, then the formula \(A \rightarrow B\) (resp. \(B \rightarrow A\)), stands for the contract “I will give one euro for one slice of pizza” (resp. “I will provide one slice of pizza for one euro”). The entailment of \(A \otimes B\) then means that both the contracts have been respected.

2. \(A \rightarrow B \nvdash B\). Here the resource \(B\) cannot be provided in the absence of \(A\).

3. \(B \vdash A \rightarrow B\). This states that if the resource \(B\) has already been provided, then it can be provided under any condition.

4. \(A \rightarrow A \vdash A\) represents the natural property that if the resource \(A\) is provided under the condition that the resource \(A\) is provided, then \(A\) is provided. This is coherent with the fact that \(A \rightarrow A \vdash A\) holds in PCL.

5. \(((A \rightarrow B) \otimes ((B \otimes B) \rightarrow A)) \nvdash (A \otimes B)\). This property and the following one are strictly related to linearity. Here a contract offering one occurrence of \(B\) fails the handshake with a contract requesting two occurrences of \(B\).

6. \((A \rightarrow !B) \otimes ((B \otimes B) \rightarrow A) \vdash A \otimes B\). Here there is an agreement, since the resource \(B\) offered by the first contract is under a !, so is available \textit{ad libitum}.

7. \((A \rightarrow B) \otimes (B \rightarrow A) \vdash B \otimes (B \rightarrow A)\). Here a contract offers \(B\) under the condition that \(A\) becomes available, in the presence of a contract which produces \(A\) from \(B\). The entailment allows \(B\) to interact with \(B \rightarrow A\). However, we do not obtain \(A \otimes B\), because if \(B \rightarrow A\) is used, it consumes \(B\).

8. \(A \rightarrow B \vdash A \rightarrow B\). This states that linear contractual implication entails linear implication (coherently with the property \(A \rightarrow B \vdash A \rightarrow B\) of PCL).

9. \((A \rightarrow B) \otimes A \vdash A \otimes B\). This stresses a difference between \(\rightarrow\) and \(\rightarrow\): the latter does not consume its argument.

10. \((A \rightarrow B) \otimes B \rightarrow C \vdash A \rightarrow C\). This states transitivity of \(\rightarrow\).

**Encoding of PCL.** In Fig. 2 we show an encoding of PCL into PCLLW, extending the one in [7]. Theorem 1 shows the mapping correct and complete.

**Theorem 1.** \(\Gamma \vdash A\) is derivable in PCL iff \(\Gamma^* \vdash A^*\) in PCLLW.

**Proof.** For \(\Rightarrow\), first we use the equivalence between the sequent calculus and the natural deduction system for PCL (shown in [1]) to get from a sequent calculus proof of \(\Gamma \vdash A\) a natural deduction proof of \(A\) from \(\Gamma\); then using the translation of the rules in Fig. 3, together with the one given by Girard from natural deduction to intuitionistic linear sequent calculus [7], by induction on the height of the proof we get a proof of \(\Gamma^* \vdash A^*\). The proof of \(\Leftarrow\) is straightforward.
Cut elimination. As for PCL, the logic PCLLW enjoys cut elimination. The proof follows the one for ILL provided in [5]. The key cases concerning the connective $\rightarrow$ are shown in the Appendix. This result is not completely straightforward — e.g. it cannot be deduced using the technique of [6] — because rule $\text{PrePost}$ makes the sequent calculus of PCLLW non-simple according to [6].

**Theorem 2 (Cut elimination).** If $\Gamma \vdash A$ is provable in PCLLW, then a cut-free proof of $\Gamma \vdash A$ exists.

3 Phase structures and decidability

In this section we combine some results of Ciabattoni, Okada and Terui [6,12] on models of linear logic called phase structures, in order to define a model and prove decidability of a fragment of PCLLW, namely PCLLW without exponentials and PrePost rules. This restriction is due to the fact that the results of [6] do not extend to rules with the shape of PrePost and to the ones concerning exponentials.

**Phase structures.** Let $\mathcal{M} = (\mathcal{M}, \cdot, 1)$ be a commutative monoid. For any $P, Q \subseteq \mathcal{M}$, we define $P \rightarrow Q = \{ y \mid \forall x \in P, (x \cdot y \in Q) \}$.

**Definition 1 (Affine Phase structure).** A phase structure is a pair $(\mathcal{M}, D_{\mathcal{M}})$, where $D_{\mathcal{M}}$ is a subset of $\wp(\mathcal{M})$, called the set of facts, closed under arbitrary intersections, and such that for all $A, B \in D_{\mathcal{M}}$, the set $A \rightarrow B$ belongs to $D_{\mathcal{M}}$.

Given a commutative monoid $\mathcal{M}$, an ideal of $\mathcal{M}$ is a subset $A$ of $\mathcal{M}$ such that $A \cdot \mathcal{M} = A$ (where $P \cdot Q = \{ x \cdot y \mid x \in P, y \in Q \}$). An affine phase structure is a phase structure where each fact is an ideal.

**Valuations.** Given a connective $A \star B$ of PCLLW, we associate with it two operations on the facts of an affine phase structures, $A^* \ast_l B^*$, $A^* \ast_r B^*$ (where $A^*, B^*$ are facts), called respectively its left and right interpretation, defined on the basis of the left and right introduction rules of the connective. See [6] for the actual definitions.

**Definition 2.** Given an affine phase structure $(\mathcal{M}, D_{\mathcal{M}})$, a valuation on $(\mathcal{M}, D_{\mathcal{M}})$ is a map $f$ from the set of formulas of PCLLW to the elements of $D_{\mathcal{M}}$, such that for each formula $A \star B$, $(f(A) \ast_r f(B)) \subseteq f(A \ast B) \subseteq (f(A) \ast_l f(B))$.

A valuation $f$ is straightforwardly extended to sequences $\Gamma = A_1, \ldots, A_n$. A sequent $\Gamma \vdash A$ is valid iff $f(\Gamma) \subseteq f(A)$ (if $\Gamma$ is empty then we let $f(\Gamma) = 1$).

**Definition 3 (Model).** An affine phase model of PCLLW is a tuple $(\mathcal{M}, D_{\mathcal{M}}, f)$ such that $(\mathcal{M}, D_{\mathcal{M}})$ is an affine phase structure, and $f$ is a valuation on $(\mathcal{M}, D_{\mathcal{M}})$. 

*Proceedings of ICTCS 2013*
An affine phase model \((M, D_M, f)\) satisfies a formula \(A\) of PCLLW iff the sequent \(\Gamma \vdash A\) is valid under \(f\); similarly, a formula \(A\) of PCLLW is satisfiable if there exists an affine phase model which satisfies \(A\).

**Theorem 3 (Soundness).** For all affine phase models \((M, D_M, f)\), if \(\Gamma \vdash A\) is derivable in PCLLW, then \(\Gamma \vdash A\) is valid under \(f\).

**Proof.** Similar to the one in [6].

To prove completeness, we first have to define a syntactic model of PCLLW in affine phase structures. Let \(\mathcal{F}\) be the set of formulas of PCLLW. Let us consider the free monoid \(\mathcal{F}^*\) generated by \(\mathcal{F}\). By \([\Gamma \vdash C]\) we denote the set \(\{\Sigma \mid \Sigma, \Gamma \vdash C\}\). By \([C]\) we mean \([\top \vdash C]\). We define the syntactic affine phase structure of PCLLW taking as monoid the free monoid \(\mathcal{F}^*\), and as set of facts the set \(D_{\mathcal{F}^*}\) comprising \(\bigcap_{i \in A} [\Gamma_i \vdash C_i]\) for any index set \(A\). Then we define, by induction on the complexity of a formula of PCLLW, a specific valuation \(f_0\) (see [6] for details).

**Theorem 4 (Completeness).** Let \((\mathcal{F}^*, D_{\mathcal{F}^*}, f_0)\) be a (syntactic) affine phase model of PCLLW. If \(\Gamma \vdash C\) is valid under \(f_0\), then \(\Gamma \vdash C\) is derivable.

**Proof.** Similar to the one in [6].

**Decidability.** Given an affine phase model \((M, D_M, f)\), we consider the congruence relation \(\equiv\) on the elements of \(M\) defined by \(x \equiv y\) iff \((x \in X \leftrightarrow y \in X\) for every \(X \in D_M\)). For an affine phase model, we define its quotient model \((M/\equiv, D', f')\) where \(M/\equiv\) is the quotient of \(M\) w.r.t. \(\equiv\), \(D' = \{F/\equiv \mid F \in D_M\}\), and \(f'(A) = (f(A))/\equiv\).

As in [8], we can observe that there is a natural bijection between \(D_M\) and \(D'\) (because the facts of \(D_M\) are already closed by \(\equiv\)), so it is easy to conclude that \(M/\equiv\) satisfies the same formulas as \(M\). Following [12], we see that the \(M/\equiv\) is finite whenever the set of facts \(D_M\) is finite. We can then prove the finite model property of PCLLW by exhibiting a syntactical model where the set of facts is finite. Indeed, it suffices quotienting the syntactic model using \(\equiv\). Formally, for a formula \(A\) of PCLLW, we define PCLLW\((A)\) as PCLLW restricted to the subformulas of \(A\), and PCLLW\(^*\)(\(A\)) as the syntactical model induced by PCLLW\((A)\). The following lemma is adapted from [12].

**Lemma 1.** The set of facts of PCLLW\(^*\)(\(A\)) is finite. Therefore, PCLLW\(^*\)(\(A)/\equiv\) is a finite model.

**Theorem 5 (Finite Model Property).** For all formulas \(A\) of PCLLW, \(A\) is provable iff it is satisfied by every finite affine phase model.

**Theorem 6.** PCLLW is decidable.
Proof. Obviously the set of theorems of PCLLW is recursively enumerable. The set of non-theorems of PCLLW is recursively enumerable too, by the finite model property. Moreover, the property of being a finite affine phase model is decidable, since being a finite affine phase structure and being a valuation over a finite affine phase structure are both decidable properties (the domain is finite so all quantifications are finite). Then we can conclude that PCLLW is decidable.

4 Conclusions

As pointed out in §2, contractual implication $A \rightarrow B$ should be interpreted as “$B$ can be provided when $A$ is eventually available”. Standing by this interpretation, one should expect that the presence of some non-determinism involving the availability of the resources would also imply non-determinism on the consequences of the contracts. That is, we should have that (and indeed we have):

$$(A \& B) \otimes (A \rightarrow A') \otimes (B \rightarrow B') \vdash A' \& B'$$

because, having to choose one resource among $A, B$, only one of the two contracts should be used. Actually, a stronger property holds with the calculus in Fig. 1:

$$(A \& B) \otimes (A \rightarrow A') \otimes (B \rightarrow B') \vdash A' \otimes B'$$

Here both contracts are enabled at the same time, even though one has to choose between either $A$ or $B$. Refining PCLLW in order to eliminate this undesired feature is an current work in progress.

An alternative model for dealing with resources and debts is cancellative linear logic [9]. It corresponds to financial games in Petri nets, where moves allow for creating debts, and for annihilating debts with credits. While the relations with PCL (and with Lending Petri nets [2]) have to be investigated, some differences are apparent: e.g., in [9] one can always create new debts (for any resource), while in PCLLW a resource $B$ can be taken on credit only by consuming some $A \rightarrow B$, and only under the guarantee that $A$ will eventually be available.

References

Towards a linear contract logic

Fig. 3. Translation of PCL proofs into PCLLW proofs.
Proof of cut elimination (cases for $\rightarrow$)

- (Zero, Fix)

\[
\begin{align*}
\pi_1 & \quad \pi_2 & \quad \pi_3 \\
\Gamma \vdash B & \quad \Delta, B \vdash A & \quad \Delta, B \vdash C \\
\text{ZERO} & \quad \text{ZERO} & \quad \text{ZERO} \\
\Gamma \vdash A \rightarrow B & \quad \Delta, A \rightarrow B & \quad \Delta, A \rightarrow B \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta \vdash B & \quad \Delta, B \vdash C & \quad \Delta, B \vdash C \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Delta \vdash C & \quad \Delta, B \vdash C & \quad \Delta, B \vdash C \\
\end{align*}
\]

The cut above is reduced as follows:

\[
\begin{align*}
\pi_1 & \quad \pi_3 \\
\Gamma \vdash B & \quad \Delta, B \vdash C \\
\text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Delta \vdash C & \quad \Delta, B \vdash C & \quad \Delta, B \vdash C \\
\end{align*}
\]

- (Zero, PrePost)

\[
\begin{align*}
\pi_1 & \quad \pi_2 & \quad \pi_3 & \quad \pi_4 \\
\Gamma \vdash B & \quad \Delta, C \vdash A & \quad \Theta, B \vdash D & \quad \Theta, B \vdash D \\
\text{ZERO} & \quad \text{PREPOST} & \quad \text{PREPOST} & \quad \text{PREPOST} \\
\Gamma \vdash A \rightarrow B & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Theta \vdash B \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D \\
\end{align*}
\]

The cut above is reduced as follows:

\[
\begin{align*}
\pi_1 & \quad \pi_3 \\
\Gamma \vdash B & \quad \Theta, B \vdash D \\
\text{CUT} & \quad \text{CUT} \\
\Gamma, \Theta, \Theta \vdash D \rightarrow \Delta \rightarrow C \rightarrow D & \quad \Theta, B \vdash D & \quad \Theta, B \vdash D \\
\end{align*}
\]

- (PrePost, PrePost)

\[
\begin{align*}
\pi_1 & \quad \pi_2 & \quad \pi_3 & \quad \pi_4 \\
\Gamma, A \vdash E & \quad \Delta, C \vdash A & \quad \Theta, B \vdash D & \quad \Theta, B \vdash D \\
\text{PREPOST} & \quad \text{PREPOST} & \quad \text{PREPOST} & \quad \text{PREPOST} \\
\Gamma, A, E \rightarrow F \rightarrow A \rightarrow B & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D & \quad \Delta, \Theta, A \rightarrow B \rightarrow C \rightarrow D \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Theta, E \rightarrow F \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D & \quad \Delta, \Theta \vdash B \rightarrow C \rightarrow D \\
\end{align*}
\]

The cut above is reduced as follows:

\[
\begin{align*}
\pi_3 & \quad \pi_1 & \quad \pi_2 \\
\Delta, C \vdash A & \quad \Gamma, A \vdash E & \quad \Lambda, F \vdash B & \quad \Theta, B \vdash D \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Theta, E \rightarrow F \rightarrow C \rightarrow D & \quad \Lambda, \Theta, E \rightarrow F \rightarrow C \rightarrow D & \quad \Lambda, \Theta, \Theta \vdash D \rightarrow \Delta \rightarrow C \rightarrow D & \quad \Lambda, \Theta \vdash D \rightarrow \Delta \rightarrow C \rightarrow D \\
\end{align*}
\]

- (PrePost, Fix)

\[
\begin{align*}
\pi_1 & \quad \pi_2 & \quad \pi_3 & \quad \pi_4 \\
\Gamma, A \vdash C & \quad \Delta, B \vdash A & \quad \Delta, B \vdash E & \quad \Delta, B \vdash E \\
\text{PREPOST} & \quad \text{PREPOST} & \quad \text{PREPOST} & \quad \text{PREPOST} \\
\Gamma, A, C \rightarrow D \rightarrow A \rightarrow B & \quad \Delta, A \rightarrow B \rightarrow E & \quad \Delta, A \rightarrow B \rightarrow E & \quad \Delta, A \rightarrow B \rightarrow E \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Gamma, \Delta, \Delta \rightarrow D \rightarrow E & \quad \Delta, B \vdash E & \quad \Delta, B \vdash E & \quad \Delta, B \vdash E \\
\end{align*}
\]

The cut above is reduced as follows:

\[
\begin{align*}
\pi_3 & \quad \pi_1 & \quad \pi_2 \\
\Delta, B \vdash A & \quad \Gamma, A \vdash C & \quad \Lambda, D \vdash B & \quad \Delta, B \vdash E \\
\text{CUT} & \quad \text{CUT} & \quad \text{CUT} & \quad \text{CUT} \\
\Lambda, \Delta, \Delta \vdash D \rightarrow E & \quad \Lambda, \Delta \rightarrow E & \quad \Lambda, \Delta \rightarrow E & \quad \Lambda, \Delta \rightarrow E \\
\end{align*}
\]

Proceedings of ICTCS 2013
Minimum Weight Multicolor Dynamos

Sara Brunetti\textsuperscript{1}, Gennaro Cordasco\textsuperscript{2}, Luisa Gargano\textsuperscript{3}, Elena Lodi\textsuperscript{1}, and Walter Quattrociocchi\textsuperscript{4,5}

\textsuperscript{1} Department of Information Engineering and Mathematics, University of Siena, Italy
\textsuperscript{2} Department of Psychology, Second University of Naples, Italy
\textsuperscript{3} Department of Computer Science, University of Salerno, Italy
\textsuperscript{4} IMT Alti Studi Lucca, piazza S. Ponziano 6, 55100 Lucca, Italy
\textsuperscript{5} Laboratory for the modeling of biological and socio-technical systems, Northeastern University, Boston, MA 02115 USA

Abstract. In this paper we introduce a novel multicolored opinion dynamics model on networks. Given a graph $G$, initially, each node in $V$ has an assigned weight mimicking the individuals conviction of a new idea about a product. Weights’ values range in the set $\{0, \ldots, k-1\}$. Under an updating irreversible rule, by proceeding to rounds, each node changes its weight according to the weights of its neighbors. Our focus is on the initial assignments of weights leading each node to take the value $k - 1$ – e.g. unanimous maximum level acceptance – within a given number of rounds. As we aim to minimize the sum of the initial weights of the nodes, we determine lower bounds, and we provide constructive tight upper bounds for some class of regular topologies: rings, tori, and cliques.

Keywords: information spreading, opinion dynamics.

1 Introduction

Monopolies were initially introduced to study the effect of faulty nodes inside distributed computing systems. A monopoly in a graph is a subset $M$ of nodes such that each other node of the graph has a prescribed number (or a fraction) of neighbors belonging to $M$. The problem of finding monopolies in graphs has been widely studied (e.g. [1, 16, 17]) for connections with minimum dominating set problem).

Dynamic monopolies or shortly dynamo were introduced by Peleg [20]. The initial set $M_0 \subset V$ (usually, black-colored vertices) is given, and then a recoloring (majority) rule is applied to every vertex yielding to a new set $M_1$, and so on. The set $M_0$ is called a dynamo, if there is some $i$ such that $M_i = V$. Dynamos have been extensively studied with respect to the bounds of the size of the dynamos, the time needed to converge into a static configuration, and topologies over which the interaction takes place [2, 3, 12, 15, 19, 21]. Other works studied fault tolerance problems in majority-based systems in different networks topologies, [6, 8, 9, 13, 18].

In this paper we extend the notion of dynamos to weighted multicolor dynamos. Our aim is to use weight multicolor dynamos for modeling opinion spreading processes i.e. the diffusion of message or information as a process of social influence. Recently, information spreading has been intensively studied also in the context of viral marketing, which uses social networks to achieve marketing objectives through self-replicating viral processes, analogous to the
spread of viruses. The goal here is to create a marketing message that can initially convince a selected set of people and then spread to the whole network in a short period of time [10]. One problem in viral marketing is the target set selection targeting the definition of the minimal number of nodes able to activate (persuade) the whole network [11]. Such a challenge is absolutely not trivial as the target set selection problem has been proved to be NP-hard through a reduction to the node cover problem [14] and recently, even inapproximability results of opinion spreading problems have been presented in [7].

1.1 Definitions, Preliminaries and Results.

We consider the following multicolor opinion spreading model on networks (a different extension has been studied in [5]).

Let $G = (V, E)$ be an undirected connected graph. For each $v \in V$, we denote by $N(v) = \{u \in V \mid \{u, v\} \in E\}$ the neighborhood of $v$ and by $d(v) = |N(v)|$ its cardinality (i.e., the degree of $v$). We assume the nodes of $G$ to be weighted by the set $[k] = \{0, 1, \ldots, k-1\}$ of the first $k \geq 2$ integers. For each $v \in V$ we denote by $c_v \in [k]$ the weight assigned to a given node $v$.

**Definition 1.** A configuration $C$ on $G$ is a partition of $V$ into $k$ sets $\{V_0, V_1, \ldots, V_{k-1}\}$, where $V_j = \{v \in V \mid c_v = j\}$ is the set of nodes of weight $j$. The weight $w(C)$ of $C$ is the weighted sum of its nodes $w(C) = \sum_{j=0}^{k-1} j \times |V_j| = \sum_{v \in V} c_v$.

Consider the following node weighting process on $G$ using the set of weights $[k]$ and a threshold value $\lambda$ (for some $0 < \lambda \leq 1$):

In the initial configuration, each node has a weight in $[k]$. Then node weights are updated in synchronous rounds (i.e., round $i$ depends on round $i-1$ only). Let $c_v(i)$ denote the weight of node $v$ at the end of round $i \geq 0$; during round $i \geq 1$, each node updates its weight according to the weight of its neighbors at round $i-1$. Specifically, each node $v$ at first computes the number $n^+(v) = |\{u \in N(v) \mid c_u(i-1) > c_v(i-1)\}|$ of neighbors having a weight larger than its current one $c_v(i-1)$; then, it applies the following irreversible rule:

$$c_v(i) = \begin{cases} c_v(i-1) + 1 & \text{if } n^+(v) \geq \lceil \lambda d(v) \rceil \\ c_v(i-1) & \text{otherwise} \end{cases}$$

We denote the initial configuration by $C^0$ and the configuration at round $i$ by $C^i$.

We are interested into initial configurations that converge to the unanimous all-$(k-1)$s configuration – i.e., there exists a round $t^*$ such that for each $i \geq t^*$ and for each node $v$, it holds $c_v(i) = k - 1$. Such configurations are named $k$-weights dynamic monopoly (henceforth $k$-dynamo). A $(k, t)$-dynamo is a $k$-dynamo which reaches its final configuration within $t$ rounds, that is, $c_v(i) = k - 1$ for each node $v \in V$ and $i \geq t$. A $(k, t)$-dynamo is optimal if its weight is minimal among all the $(k, t)$-dynamos.
Given a graph $G$, a set of weights $[k]$, a threshold $\lambda$, and an integer $t > 0$, our goal is in determining an optimal $(k, t)$-dynamo. In the spirit of our model of opinion spreading, we want everyone to completely accept the new opinion within a given time bound while minimizing the initial convincing effort (sum of the initial node weights). We notice that we are interested in the case in which the spreading is essentially a one-way process: once an agent has adopted an opinion (or behavior, innovation, ...), she/he sticks with it.

We determine lower bounds on the sum of the initial weights of the nodes under the irreversible simple majority rules, where a node increases its weight if and only if the majority of its neighbors have a weight that is higher than its own one. Moreover, we provide constructive tight upper bounds for some class of regular topologies: rings, tori, and cliques. A complete version of this paper is given in [4].

## 2 Time Bounded Dynamos

In this section we provide a lower bound on the weight of a $(k, t)$-dynamo and study the minimum value of $t$ for which an optimal $(k, t)$-dynamo coincides with a $k$-dynamo. We omit the proofs for limit in space (see [4]).

**Definition 2.** Consider an undirected connected graph $G = (V, E)$. Let $k \geq 2$ and $t \geq 1$ be integers and $0 < \lambda \leq 1$. An initial configuration $C$ for $G$ is called $(k, t)$-simple-monotone if $V$ can be partitioned into $t + 1$ sets $X_{-s}, X_{-s+1}, \ldots, X_{k-1}$ (here $s = t - k + 1$) where $X_{k-1} \neq \emptyset$, and for each $v \in X_i$

  (i) $c_v(0) = \max(i, 0)$;

  (ii) $v$ has at least $\lceil \lambda d(v) \rceil$ neighbours in $\bigcup_{j=i+1}^{k-1} X_j$.

According to the definition, for each $v \in X_i$, if $i \leq 0$ then $c_v(0) = 0$, else $c_v(0) = i$ or $X_i \neq \emptyset$. We can show by induction on $i$ that for each $i = -s, -s+1, \ldots, k-1$ (here $s = t - k + 1$) and $j = 0, \ldots, t$ and for each $v \in X_i$: $c_v(j) = \min(j+i, k-1)$, if $j+i > 0$, and 0 otherwise. As a consequence:

**Lemma 1.** Any $(k, t)$-simple-monotone configuration for an undirected connected graph $G$ is a $(k, t)$-dynamo for $G$.

**Lemma 2.** Let $G = (V, E)$ be an undirected connected graph. There exists an optimal $(k, t)$-dynamo for $G$ which is a $(k, t)$-simple-monotone configuration for $G$.

By Lemma 2 we can restrict our attention to $(k, t)$-simple-monotone configurations for $G$. Therefore, the set $V$ can be partitioned into $t + 1$ subsets $X_{-s}, X_{-s+1}, \ldots, X_{k-1}$ where $s = t - k + 1$ and for $i = -s, -s+1, \ldots, k-1$, $X_i$ denotes the set of nodes whose weight at round $j$ is $\max(0, \min(j+i, k-1))$.

It is possible to prove the following lower bound for the weight of any $(k, t)$-dynamo $C$:

---

**Proceedings of ICTCS 2013**
Theorem 1. Consider an undirected connected graph $G = (V, E)$ and let $k \geq 2$ and $t \geq 1$ be integers. Any $(k, t)$-dynamo $C$, with $\lambda = 1/2$, has weight

$$w(C) \geq \begin{cases} \frac{|V|}{2^{\rho(t+s+1)+1}} \times (k-1 + \rho(\ell+1)) & \text{if } t \geq k-1 \\ \frac{|V|}{2^{\rho(t+1)+1}} \times (k-1 + \rho(\ell+1) - s(s+1)) & \text{otherwise,} \\ \end{cases}$$

where $\rho$ is the ratio between the maximum and the minimum degree of the nodes in $V$ and $s = t - k + 1$.

Let us denote the size of $X_i$ by $x_i$; in order to prove the theorem, one show that, for each $i = -s, -s+1, \ldots, k-2$, it holds $x_i \leq 2\rho x_{k-1}$. Moreover, the function to minimize can be written as: $w(C) = \sum_{j=1}^{k-1} jx_j = x_{k-1}\left( (k-1) + \sum_{j=1}^{k-2} jy_j \right)$ where $y_i = \frac{x_i}{x_{k-1}}$. Hence, $x_{k-1} = \frac{|V|}{1+\sum_{j=-s}^{k-2} y_j}$ and

$$w(C) = \frac{k}{|V|} \times \frac{1+\sum_{j=-s}^{k-2} jy_j}{1+\sum_{j=-s}^{k-2} y_j},$$

(1)

with $0 \leq y_j \leq 2$, for each $j = 0, 1, \ldots, k-2$. It is possible to show that the rightmost term is minimized when

$$y_i = \begin{cases} 2\rho & \text{if } -s \leq i \leq \ell \\ 0 & \text{if } \ell < i \leq k-2, \\ \end{cases}$$

(2)

where the value of $\ell$ is given in the Theorem. This means that if there exists a $(k, t)$-dynamo $C$ for a given graph $G$ such that $w(C)$ coincides with the lower bound, the set $V$ can be partitioned into $t+1$ subsets $X_{-s}, X_{-s+1}, \ldots, X_{k-1}$, where $x_i = 2\rho x_{k-1}$ for $-s \leq i \leq \ell$, and $X_i = \emptyset$ for $\ell < i < k-1$. The constructions we give in the next section are based on this result.

Remark 1. By analyzing the value of $\ell$ in the case $t \geq k-1$ we have that whenever $t > \frac{k(2\rho+1)-2\rho-4}{2\rho}$ then $\ell = 0$, hence only the weights 0 and $k-1$ will appear in the optimal configuration. When $\rho = 1$ (i.e., on regular graphs) one has $t > \frac{3}{2}k - 3$. Therefore our result generalizes the one in [12] with $k = 2$: indeed, when $t \geq k-1 = 1$ by the above consideration we get $t > \frac{3}{2}k - 3 = 0$ and $\ell = 0$, and hence, $w(C) \geq \frac{|V|}{2^{s+1}} \times (k-1) = \frac{|V|}{2^{t+1}}$.

3 Building $(k, t)$-dynamo

In this section we provide several optimal (or almost optimal) $(k, t)$-dynamo constructions for Rings and Tori ($\lambda = 1/2$) and Cliques (any $\lambda$).
3.1 Rings
A $n$-node ring $R_n$ consists of $n$ nodes and $n-1$ edges, where for $i=0, 1, \ldots, n-1$ each node $v_i$ is connected with $v_{(i-1) \mod n}$ and $v_{(i+1) \mod n}$.

A necessary and sufficient condition for $C(R_n, k)$ to be a $k$-dynamo ($\lambda \leq 1/2$) is that at least one node of $R_n$ is weighted by $k-1$.

A $(k, t)$-dynamo ($\lambda = 1/2$) for a ring $R_n$ is obtained by the following partition of $V$ which defines the initial configuration $C(R_n, k, t)$: for $i=0, 1, \ldots, n$,

$$\forall v_i \in R_n, \ v_i \in \begin{cases} X_{k-1} & \text{if } j = 0 \\ X_{\ell+1-j} & \text{if } 1 \leq j \leq \ell + s + 1 \\ X_{j-\ell-2s-2} & \text{if } \ell + s + 2 \leq j \leq 2\ell + 2s + 2 \end{cases}$$

where $s = t - k + 1$, $j = i \mod (2\ell + 2s + 3)$ and $\ell = \sqrt{t + 1 + s^2 + s} - (s + 1)$ if $t \geq k - 1$ and $\ell = \sqrt{t + 1} - (s + 1)$ otherwise.

**Theorem 2.** The configuration $C(R_n, k, t)$ is a $(k, t)$-dynamo for any value of $n$, $\lambda = 1/2$, $k \geq 2$ and $t \geq 1$, and its weight is

$$w(C(R_n, k, t)) \leq \begin{cases} \frac{n^2}{2t + 2s + 3} (k-1 + \ell(\ell+1)) & \text{if } t \geq k - 1 \\ \frac{n^2}{2t + 2s + 3} (k-1 + \ell(\ell+1) - s(s+1)) & \text{otherwise} \end{cases}$$

where $\ell = \sqrt{t + 1 + s^2 + s} - (s+1)$. If $n$ is a multiple of $2\ell + 2s + 3$, $C(R_n, k, t)$ is optimal.

**Proof.** By construction $C(R_n, k, t)$ is $(k, t)$-simple-monotone, and so by Lemma 1, $C(R_n, k, t)$ is a $(k, t)$-dynamo. In order to prove its optimality, there are two cases to consider: if $t \geq k - 1$, then starting from $v_0$ each set of $2\ell + 2s + 3$ nodes weights $k - 1 + 2 \sum_{i=1}^{t} i = k - 1 + \ell(\ell+1)$. Then the weight of $C(R_n, k, t)$ is smaller than the weight of $C(R_{\overline{n}}, k, t)$ where $\overline{n} = \left\lfloor \frac{n}{2t + 2s + 3} \right\rfloor \times (2\ell + 2s + 3)$. Hence, $w(C(R_n, k, t)) \leq w(C(R_{\overline{n}}, k, t)) = \frac{n^2}{2t + 2s + 3} (k-1 + \ell(\ell+1))$. Similarly for $t < k - 1$. If $n$ is a multiple of $2\ell + 2s + 3$, $w(C(R_n, k, t))$ matches the bound in Theorem 1. \hfill $\square$

3.2 Tori
A $n \times m$-node tori $T_{n,m}$ consists of $n \times m$ nodes and $2(n \times m)$ edges, where for $i=0, 1, \ldots, n-1$ and $j=0, 1, \ldots, m-1$, each node $v_{i,j}$ is connected with four nodes: $v_{i,(j-1) \mod m}$, $v_{i,(j+1) \mod m}$, $v_{(i-1) \mod n,j}$ and $v_{(i+1) \mod n,j}$.

A $(k, t)$-dynamo ($\lambda = 1/2$) for $T_{2\ell+2s+3,2\ell+2s+3}$ is obtained by weighting diagonals with the same order defined for dynamos on rings. Specifically, the configuration $C(T_{2\ell+2s+3,2\ell+2s+3}, k, t)$ is defined by the partition of $V$ described as follows, let $D_i = \{v_{a,b} : i = (b - a) \mod (2\ell + 2s + 3)\}$ denote the $i$-th diagonal of $T_{2\ell+2s+3,2\ell+2s+3}$, for $i=0, 1, \ldots, 2\ell + 2s + 2$,

$$\forall v \in D_i, \ v \in \begin{cases} X_{k-1} & \text{if } i = 0 \\ X_{\ell+1-i} & \text{if } 1 \leq i \leq \ell + s + 1 \\ X_{i-\ell-2s-2} & \text{if } \ell + s + 2 \leq i \leq 2\ell + 2s + 2 \end{cases}$$
where $s = t - k + 1$, $\ell = \lfloor \sqrt{t + 1 + 8s^2 + s} \rfloor$ if $t \geq k - 1$ and $\ell = \lfloor \sqrt{t + 1} \rfloor - (s + 1)$ otherwise.

**Theorem 3.** The configuration $C(T_{2\ell+2s+3, 2\ell+2s+3}, k, t)$ is an optimal $(k, t)$-dynamo for any $k \geq 2$, $t \geq 1$ and $\lambda = 1/2$.

**Proof.** Let $C = C(T_{2\ell+2s+3, 2\ell+2s+3}, k, t)$. By construction $C$ is $(k, t)$-simple-monotone, hence by Lemma 1, it is a $(k, t)$-dynamo. To show its optimality we distinguish two cases. If $t \geq k - 1$, each row (resp. each column) corresponds to $C(R_{2\ell+2s+3}, k, t)$ and its weight is $k - 1 + \ell(\ell + 1)$. Overall, $w(C) = (2\ell + 2s + 3) \times (k - 1 + \ell(\ell + 1))$ that matches the bound in Theorem 1. Similarly for $t < k - 1$. \qed

A $(k, t)$-dynamo for $T_{n,m}$, we denote by $C(T_{n,m}, k, t)$, is obtained by building a grid $\left[ \frac{n}{2\ell+2s+3} \right] \times \left[ \frac{m}{2\ell+2s+3} \right]$, where each cell is filled with a configuration $C(T_{2\ell+2s+3, 2\ell+2s+3}, k, t)$ defined above. Then, the exceeding part is removed and the last row and the last column are updated. In particular, for each column (resp. row), if the removed part contains a $k - 1$, then the element in the last row (resp. column) is given the value $k - 1$ (see Figure 1). Therefore we can prove that:

![Fig. 1.](image)

**Theorem 4.**

(i) $C(T_{n,m}, k, t)$ is an $(k, t)$-dynamo for any value of $n$, $m$, $\lambda = 1/2$, $k \geq 2$ and $t \geq 1$.

(ii) The weight of $C(T_{n,m}, k, t)$ is

$$w(C(T_{n,m}, k, t)) \leq \begin{cases} \left[ \frac{n}{2\ell+2s+3} \right] \left[ \frac{m}{2\ell+2s+3} \right] (2\ell+2s+3)(k-1+\ell(\ell+1)) & \text{if } t \geq k - 1 \\ \text{where } \ell = \lfloor \sqrt{t + 1 + 8s^2 + s} \rfloor - (s + 1) \end{cases}$$

$$\begin{cases} \left[ \frac{n}{2\ell+2s+3} \right] \left[ \frac{m}{2\ell+2s+3} \right] (2\ell+2s+3)(k-1+\ell(\ell+1)-s(s+1)) & \text{otherwise} \\ \text{where } \ell = \lfloor \sqrt{t + 1} \rfloor - (s + 1) \end{cases}$$

(iii) If both $n$ and $m$ are multiples of $2\ell + 2s + 3$, $C(T_{n,m}, k, t)$ is optimal.

3.3 Cliques

Let $K_n$ be the clique on $n$ nodes. A necessary condition for a $k$-dynamo $C(K_n, k)$ is that $\lfloor \lambda(n - 1) \rfloor$ nodes are weighted by $k - 1$. The condition is
also sufficient and if the remaining \(|\lambda(n-1)|\) nodes are weighted by 0, the \(k\)-dynamo is optimal and reaches its final configuration within \(t = k - 1\) rounds. So, when \(t \geq k - 1\) the optimal configuration is obtained by weighting \(|\lambda(n-1)|\) nodes by \(k - 1\) and the remaining nodes by 0. For \(t < k - 1\), an optimal \((k,t)\)-dynamo is obtained by assigning weight \(k - t - 1\) to all the non-\(k - 1\) weighted nodes. Clearly this configuration is optimal, if we assign a weight smaller than \(k - t - 1\) to a node \(v\), then \(v\) can not reach the weight \(k - 1\) within \(t\) rounds. Therefore:

**Theorem 5.** Let \(K_n\) be the clique on \(n\) nodes. An optimal \((k,t)\)-dynamo \(C(K_n,k,t)\) has weight

\[
w(C(K_n,k,t)) = (k - 1) \times |\lambda(n-1)| + \max(k - t - 1,0) \times |\lambda(n-1)|.
\]

4 Conclusion and Open Problems

We studied multicolor dynamos with respect to weight and time by providing lower bounds on the weight of any \((k,t)\)-dynamo as well as constructive tight upper bounds for rings, tori and cliques. Multicolor dynamos extend the setting of “classical” dynamos from 2 possible weights (denoting whether a node has accepted the opinion or not) to \(k\) levels so that our results generalize some of the results known for 2-dynamos. Constructions based on different graphs would be interesting to study. This challenge however is in general difficult due to the NP-hardness results \([14]\). In respect of the adopted protocol, different updating rules could also be considered. For instance, any node could update its weight if the ratio between the sum of the weights of its neighbours and its degree (number of neighbours) is larger than its weight. Reversible rules could also be investigated. Finally, instead of minimizing the sum of the weights of any \((k,t)\)-dynamo, one could consider a weighted sum so addressing the case where not all the nodes have the same priority.

Acknowledgments

We would like to thank Ugo Vaccaro for many stimulating discussions.

References

On the composition of regional structures and their logics

Luca Bernardinello\textsuperscript{1} and Carlo Ferigato\textsuperscript{2}

\textsuperscript{1} DISCo, Università degli studi di Milano-Bicocca
\textsuperscript{2} Joint Research Centre of the EU/IPSC, Ispra

Abstract. The algebraic structure associated to the set of regions of a Condition/Event transition system is known to be a specific class of partially ordered orthomodular sets. We show that the algebraic structure of the regions is as well a test space with some specific properties. Test spaces are logical structures used for the description of physical tests of quantum phenomena. By exploiting the properties of test spaces, an operation of composition of regional structures is defined. The exact characterization of the class of orthomodular sets in which the result falls is open.

This paper presents a selection of ideas and results from a paper presented at ART2013, a workshop on the theory and applications of regions of transition systems.

1 Introduction

The synthesis of Petri Nets \cite{llo} is a formal technique for the distributed design of systems once their behaviour is specified with traditional techniques, typically as automata. This technique was made possible by the introduction of the notion of region \cite{p}. Given a high level description in terms of automata of the system to be realized, its regional structure implicitly defines the concurrent components of the system itself and allows for a low level description of its behaviour and concurrent structure in terms of Petri Nets. By high level we intuitively mean “close to the design” while by low level we intuitively mean “close to the implementation”.

In the last twenty years, the research on this technique opened to several fields, while the general theory has been settled in \cite{z} together with the synthesis procedure for various classes of nets from the respective classes of automata. Among the research fields opened by the theory of regions, we can quote: the computational complexity of the synthesis process \cite{z1}, the composition of systems by exploiting their regional structure \cite{z2}, the logical and algebraic properties of regions \cite{z3}.

With this work, we aim at exploring further the two last research directions listed above, namely the composition of systems by exploiting their regional structure and the logical and algebraic properties of regions. We do this in two ways: firstly by defining a composition operation mediated by morphisms whose objects are orthomodular structures; secondly by showing that the structure of the elementary regions of a transition system has the same properties of a logical language used for the description of physical tests of
quantum phenomena: a test space. By exploiting the properties of test spaces, we subsequently show that the result of the composition operation gives rise to an orthomodular structure. The exact characterization of the class orthomodular structures resulting from the composition remains to be done.

In presenting our work, we build on the intuition of a region as a condition of a Condition/Event net system (CE net system for short). In this view, a CE net system can be seen as a collection of logical propositions about a system concurrently valid or non-valid. The system dynamics — concurrent changes from validity to non-validity and viceversa — is structurally defined by the environment of any proposition. In CE net systems, snapshots of the set of valid conditions in a given state can be reconstructed from the local dynamics in terms of sequential case graphs, a class of automata with a well defined regional structure.

2 Preliminary definitions

Condition/event (CE for short) net systems are a basic class of Petri nets, where places are boolean variables (or conditions), and a transition is enabled if its preconditions are all true and its postconditions are all false.

The behaviour of a CE net system can be described by its sequential case graph, whose nodes are reachable global states, and whose arcs correspond to the firing of single transitions.

Sequential case graphs of CE net systems are a subclass of transition systems, called Condition/Event transition systems (CETS from now on). They were characterized in [7] with the help of the notion of region. The relevant definitions are here recalled.

**Definition 1.** A transition system is a structure $A = (S, E, T)$, where $S$ is a set of states, $E$ is a set of events, $T \subseteq S \times E \times S$ is a set of transitions. A transition system is finite if $S$ and $E$ are finite. We will only consider finite transition systems such that the underlying graph is simply connected; there are no loops and no multiple arcs; each event labels at least one transition.

A region is a set of states such that all occurrences of a given event have the same crossing relation (entering, leaving or non crossing) with respect to the region itself, and this property holds for all events [7].

**Definition 2.** Let $A = (S, E, T)$ be a transition system. A set of states $r \subseteq S$ is said to be a region iff $\forall e \in E, (s_1, e, s'_1), (s_2, e, s'_2) \in T$ we have $(s_1 \in r \text{ and } s'_1 \notin r) \Rightarrow (s_2 \in r \text{ and } s'_2 \notin r)$ and $(s_1 \notin r \text{ and } s'_1 \in r) \Rightarrow (s_2 \notin r \text{ and } s'_2 \in r)$.

**Example 1.** Consider the transition system in the left part of Figure 1. The sets $\{1, 2, 3\}$ and $\{3, 5, 7\}$ are regions. On the other hand $r = \{1\}$ is not a region, since an occurrence of $x$ leaves $r$, while at least another occurrence of the same event in the system neither leaves nor enters $r$.

The set of all regions of $A$ will be denoted by $R_A$. From the definition it follows that: $\emptyset, S \in R_A$ and $\forall r \in R_A : S \setminus r \in R_A$. For each $s \in S$, $R_s$ will denote
the set of regions containing $s$. Let $r \in R$. Then the pre-set of $r$, denoted by $\star r$, and the post-set of $r$, denoted by $r^\star$, are defined by: $\star r = \{ e \in E | \exists (s, e, s') \in T : s \not\in r \text{ and } s' \in r \}$, $r^\star = \{ e \in E | \exists (s, e, s') \in T : s \in r \text{ and } s' \not\in r \}$. Let $e \in E$. Then the pre-set and the post-set of $e$, denoted by, respectively, $e$ and $e^\star$, are defined by: $e = \{ r | r \in R \text{ and } e \in r^\star \}$, $e^\star = \{ r | r \in R \text{ and } e \in e^\star \}$.

Regions in transition systems and conditions in ˜E net systems are strictly related. Take a ˜E net system $N$ and its sequential case graph $A_aN_b$, the nodes of $A_aN_b$ correspond to the reachable states of $N$, namely sets of conditions (a global state is the set of true conditions in that state). Let $s$ be a condition of $N$; we can compute its extensions, defined as the set of global states in which $b$ is true. This set of global states is a region of $A_aN_b$ as shown in [7].

In the other direction, the set of all non trivial regions of a transition system serves as the set of conditions of a ˜E net system in solving the synthesis problem: given a transition system $A$, find a ˜E net system $N$ such that the transitions of $N$ are the events of $A$, and the sequential case graph of $N$ is isomorphic to $A$.

The class of transition systems which admit a solution (called Condition Event Transition Systems) is characterized with the help of the set of regions, as recalled below.

**Definition 3.** A finite transition system $A = (S, E, T)$ is a Condition/Event transition system (CETS) iff it satisfies the following axioms:

- **A1.** $\forall s, s' \in S : R_s = R_{s'} \Rightarrow s = s'$
- **A2.** $\forall s \in S \forall e \in E : e \subseteq R_s \Rightarrow \exists s' \in S (s, e, s') \in T$
- **A3.** $\forall s \in S \forall e \in E : e^\star \subseteq R_s \Rightarrow \exists s' \in S (s', e, s) \in T$

**Example 2.** The transition system given in the left part of Figure 1 is not a CETS. In fact, axiom A3 is not satisfied in state 3 for event $x$. On the contrary, the transition system given in the right part of Figure 1 satisfies axioms A1., A2. and A3. of the previous definition and therefore is a CETS.

Seen as subsets of the powerset $\wp(S)$ of $S$, the regions can be partially ordered by set inclusion. The trivial regions $\emptyset$ and $S$ are the maximum and minimum
element of this partial order. Moreover, since the complement of a region is a region, set complements can be seen as complements in this partially ordered structure of regions. In the following, a particular role will be played by the set of minimal (with respect to the order relation) regions. In the partial order of the regions, minimal regions are the elements immediately above $\emptyset$.

In [4], an analysis of the relations between CE Nets systems, CE transition systems and the associated partially ordered structure of regions is made in categorical terms. In the same work it is shown that regions of a transition system form an orthomodular set.

**Definition 4.** An Orthomodular set (OS) $P = (P, 0, 1, \leq, \wedge, \vee, (.')$ is a partially ordered set with a minimum and a maximum element, denoted 0 and 1, and with a map: $(.)' : P \to P$ such that $\forall x \in P' x' \in P$ and $(x')' = x$; $x \leq y \Rightarrow y' \leq x'$; $x \leq y \Rightarrow y = x \vee (y \wedge x')$; $x \leq y' \Rightarrow x \vee y \in P$; $x \wedge x' = 0$ (where the operators $\wedge$ and $\vee$ (inf and sup) are defined — when they exist — with respect to the order relation).

**Definition 5.** Let $P_1$ and $P_2$ be OS. A map $\phi : P_1 \to P_2$ is a morphism of OS if:

i) $p \leq q \Rightarrow \phi(p) \leq \phi(q)$;

ii) $\phi(p') = (\phi(p))'$;

iii) $\phi(0) = 0$;

iv) $p \perp q \Rightarrow \phi(p \vee q) = \phi(p) \vee \phi(q)$.

2.1 Test spaces

The following definitions and discussion about test spaces and their properties are taken mainly from [8].

A test space is a couple $\mathcal{X} = (X, \mathcal{A})$ such that $X$ is a finite set and $\mathcal{A}$ is a collection of subsets of $X$ satisfying the following conditions:

i) $\bigcup_{E \in \mathcal{A}} E = X$;

ii) $\forall E, F' \in \mathcal{A} \quad E \subseteq F \Rightarrow E = F$.

The elements $x$ of $X$ are called outcomes while the elements $E$ of $\mathcal{A}$ are called operations or tests.

Given a test space $\mathcal{X} = (X, \mathcal{A})$, we define $\tilde{\mathcal{A}} = \{\alpha \subseteq X \mid \exists E \in \mathcal{A} : \alpha \subseteq E\}$; we call events the elements of $\tilde{\mathcal{A}}$.

The logic of a test space is an orthocomplemented poset whose elements are equivalence classes of events with respect to an equivalence relation defined starting from a preorder, intuitively interpreted as a kind of implication ([8]).

We can construct a test space from the set of regions of a transition system. Let $A = (S, E, T)$ be a CE transition system, $R_A$ the set of its non trivial regions, and $\text{MR}(A)$ the set of its minimal regions. A partition of $S$ is said to be regional (m-regional) if it is made of regions (minimal regions). The set of all m-regional partitions of $S$ will be denoted by $\text{MRP}$.

The test space associated to $A$ takes as outcomes the minimal regions of $A$, and as tests the regional partitions formed by minimal regions. An event is then a set of minimal regions belonging to a partition, hence pairwise disjoint.
The test space constructed in this way is algebraic and coherent (see [8]). The following theorem will be used in the following.

**Theorem 1.** [8, theorem 6.5] The logic \( \Pi(X) \) of a coherent and algebraic test space \( X = (X, A) \) is an orthomodular poset (an OS in our notation).

### 3 Composing regional structures on an interface

In this section, we introduce an operation of composition of OS mediated by morphisms. The basis for this composition operation is the theory of regions and we apply here the same reasoning (with the appropriate variations) already used for defining a related operation of composition for nets and transition systems [5]. The operation does not result directly in a OS but in a test space (algebraic and coherent) from which an OS can be built.

Given three OS \( P_0, P_1 \) and \( P_2 \), and two injective morphisms \( \phi_1 : P_0 \to P_1 \) and \( \phi_2 : P_0 \to P_2 \), we consider \( P_0 \) as the *interface* between \( P_1 \) and \( P_2 \) representing, intuitively, a common regional structure shared by the two systems being composed.

Our final goal is at defining a composition operation for \( P_1 \) and \( P_2 \) whose result is a new OS \( P_3 \). The main idea for constructing the support set for \( P_3 \) is in the identification of the couples of elements \( x, y \) in, respectively, \( P_1 \) and \( P_2 \) when these elements are mapped to the same element by the morphisms \( \phi_1 \) and \( \phi_2 \) (that is when it exists some \( z \) in \( P_0 \) such that \( \phi_1(z) = x \) and \( \phi_2(z) = y \)). We extract from \( P_1 \) and \( P_2 \) a test space and prove that this test space is algebraic and coherent in case \( P_0 \) is a Boolean algebra.

**Definition 6.** The OS \( P_3 \) is defined by its set of atoms and the orthogonality relation. We construct the set of atoms — denoted by \( \text{at}(\cdot) \) — of \( P_3 \) in three steps:

\[ C1) \quad V_1 = \text{at}(P_1) \cup \text{at}(P_2); \]
\[ C2) \quad \text{for all } p \in V_1 \text{ the atoms in the result set whose counter-image is mapped to non-atoms by one of the morphisms are removed from } V_1: \]
\[ V_2 = V_1 \setminus \{ p \mid \phi_1(\phi_2^{-1}(p)) \notin \text{at}(P_1) \text{ or } \phi_2(\phi_1^{-1}(p)) \notin \text{at}(P_2) \} \]
\[ C3) \quad \text{all } p_1 \text{ and } p_2 \text{ in } V_2 \text{ that are images of the same element } p \text{ in } P_0 \text{ are identified (let } Z = \{ p_1, p_2 \mid \exists z \in P_0 \phi_1(z) = p_1 \text{ and } \phi_2(z) = p_2 \} \text{ and let } K = \{ (p_1, p_2) \mid \exists z \in P_0 \phi_1(z) = p_1 \text{ and } \phi_2(z) = p_2 \} \}; \]
\[ V_3 = (V_2 \setminus Z) \cup K \]

We set \( V_3 \) as the set of atoms of \( P_3 \). We define now the orthogonality relation \( \perp \) between couples of atoms in \( V_3 = \text{at}(P_3) \): \( p \perp q \) in \( \text{at}(P_3) \) if:

\[ O1) \quad p \perp q \text{ either in } P_1 \text{ or } P_2 \text{ or} \]
\[ O2) \quad p = (p_1, p_2) \text{ and } q \perp p_1 \text{ in } P_1 \text{ or } p_2 \perp q \text{ in } P_2 \text{ or} \]
\[ O3) \quad \text{there exist two elements } v \text{ and } w \text{ in } P_1 \text{ and, respectively, } P_2 \text{ such that } p \leq v \text{ in } P_1, q \leq w \text{ in } P_2 \text{ and } v \text{ and } w \text{ are the images of orthogonal elements in } P_0. \]
Said in a different way, two atoms $p$ and $q$ in $V_3$ are orthogonal if they are orthogonal elements in one of the components $P_1$ and $P_2$ or if there exist $z$ and $u$ ($z \perp u$) in $P_0$ such that $\phi_1(z) = v$, $\phi_2(u) = w$ and $v$ and $w$ are greater than $p$ and $q$ in, respectively, $P_1$ and $P_2$.

We define $A$ as the collection of the maximal cliques of $\perp$ in $V_3$ and $X_3$ as the couple $(V_3, A)$. We represent the fact that $(V_3, A)$ is the result of the composition of two OS $P_1$ and $P_2$ via the appropriate injective morphisms from an interface OS $P_0$ as:

$$X_3 = P_1(P_0)P_2.$$  

**Example 3.** Consider the orthomodular posets $P_0$, $P_1$, and $P_2$ shown below.

[Diagram]

We want to compose $P_1$ and $P_2$ with respect to the isomorphic embeddings of the interface into $P_1$ and $P_2$ given by the morphisms $\Phi_1 : P_0 \to P_1$ and $\Phi_2 : P_0 \to P_2$:

$$\begin{align*}
\Phi_1 : & \ 0 \to 0 & \Phi_2 : & \ 0 \to 0 \\
1 & \to 1 & 1 & \to 1 \\
x & \to c & x & \to f \\
x' & \to c' & x' & \to f'
\end{align*}$$

After the application of the first composition clause C1) above, the set of atoms $V_1$ is: $\{a, a', b, b', c, c', d, e, f, g, h\}$. We then apply clause C2) and remove $c'$ from the set of atoms. So, we obtain: $V_2 = \{a, a', b, b', c, d, e, f, g, h\}$. We then identify $c$ and $f$ by applying C3) and obtain $V_3 = \{a, a', b, b', d, e, c \perp f, g, h\}$.

Concerning the orthogonality relation, after the application of clauses O1) and O2) — O3) does not apply — we obtain: $\{a \perp a', b \perp b', d \perp e, d \perp c \perp f, e \perp (c, f), (c, f) \perp g, (c, f) \perp h, g \perp h\}$.

Before stating that, when the interface $P_0$ is a Boolean algebra, the result of the composition operation is an algebraic test space, we make the following remark: in an orthomodular poset $P$, an atomic decomposition of an element $p$ is a set $\{a_1, \ldots a_K\}$ of atoms of $P$ if $p = a_1 \lor \ldots \lor a_K$. In the general case, the atomic decomposition is not unique.
Theorem 2. Let $P_0$ be a Boolean algebra, $P_1$ and $P_2$ be two OSp, and $\phi_1 : P_0 \to P_1$ and $\phi_2 : P_0 \to P_2$ be injective morphisms. By applying the operation in definition 6, we obtain a test space $X$. $X$ is an algebraic and coherent test space.

We can now use theorem 1 and say that the the logic of $X$ resulting from the composition operation in definition 6 is an orthomodular poset.

Example 4. The logic of the test space obtained by the application of the composition operation in example 3 is represented below. The names of the equivalence classes are the respective representatives taken from $P_1$ and $P_2$.

![Diagram of a test space representing the logic of $X$.](image)

The next development we plan for this line of research concerns generalizing Theorem 2 to the case where $P_0$ is an orthomodular set, and proving that, if $P_0$, $P_1$ and $P_2$ are prime and coherent (see [4]), then also $P_3$ is prime and coherent. Our interest in such result derives from the fact that the orthomodular set of regions of a transition system is prime and coherent.

References

A SAT Encoding for Solving Games with Energy Objectives

Raffaella Gentilini

Dip. di Matematica e Informatica, Università di Perugia, Via Vanvitelli 1, Perugia (IT)

Abstract. Recently, a reduction from the problem of solving parity games to the satisfiability problem in propositional logic (SAT) have been proposed in [5], motivated by the success of SAT solvers in symbolic verification. With analogous motivations, we show how to exploit the notion of energy progress measure to devise a reduction from the problem of energy games to the satisfiability problem for formulas of propositional logic in conjunctive normal form.

1 Introduction

Energy games (EG) are two-players games played on weighted graphs, where the integer weight associated to each edge represents the corresponding energy gain/loss. The arenas of energy games are endowed of two types of vertices: in player 0 (resp. player 1) vertices, player 0 (resp. player 1) chooses the successor vertex from the set of outgoing edges and the game results in an infinite path through the graph. Given an initial credit of energy $c$, the objective of player 0 is to maintain the sum of the weights (the energy level) positive. The decision problem for EG asks, given a weighted game graph with initial vertex $v_0$, if there exists an initial credit for which player 0 wins from $v_0$.

Energy games have been introduced in [3, 2] to model the synthesis problem within the design of reactive systems that work in resource-constrained environments. Beside their applicability to the modeling of quantitative problems for computer aided design, EG have tight connections with important problems in game theory and logic. For instance, they are log-space equivalent to mean-payoff games (MPG) [2], another kind of quantitative two-player game very well studied both in economics and in computer science. The latter are characterized by a theoretically engaging complexity status, being one of the few inhabitants of the complexity class $\text{NP}\cap \text{coNP}$ (for which the inclusion in P is still an open problem). Moreover, parity games [4, 6]—notoriously known as poly-time equivalent to the model-checking problem for the modal mu-calculus—are in turn poly-time reducible to MPG and EG. It is a long-standing open question to know whether the model-checking problem for the modal mu-calculus is in P.

The algorithm with the currently best (pseudopolynomial) complexity for solving EG (and MPG via log-space reduction) is based on the so-called notion of energy progress measure [7]. Progress measures for weighted graphs are functions that impose local conditions to ensure global properties of the graph. A notion of parity progress measure [6] was previously exploited in [6] for the algorithmic analysis of parity games and reconsidered in [5] to devise a SAT
encoding of the corresponding games, motivated by the considerable success that using SAT solvers has had in symbolic verification. As a matter of fact, clever heuristics implemented in nowadays SAT solvers can result in algorithms that are very efficient in practice. Furthermore, there are fragments of SAT that can be solved in polynomial time. Hence, the reduction in [5] opens up a new possibility for showing inclusion of parity games in P.

Motivated by analogous reasons, in this paper we show how to exploit the notion of energy progress measure to devise a reduction from the problem of energy games to the satisfiability problem for formulas of propositional logic in conjunctive normal form. Tight upper bounds on the sizes of our reductions are also reported.

The paper is organized as follows. We recall the notions of energy games and energy progress measure in Section 2. Section 3 and Section 4 develop the reductions from energy games to difference logic and pure SAT, respectively, reporting tight bounds on the sizes of the corresponding reductions.

2 Preliminaries

In this section we provide the preliminaries to the rest of the paper. In particular, Subsection 2.1 introduces the notion of games with energy objectives, while Subsection 2.2 introduces difference logic.

2.1 Energy Games

**Game Graphs** A game graph is a tuple $\Gamma = (V,E,v_0,w,\langle V_0,V_1 \rangle)$ where $G^I = (V,E,v_0,w)$ is a weighted graph with weight function $w : E \to \mathbb{Z}$ and $\langle V_0,V_1 \rangle$ is a partition of $V$ into the set $V_0$ of player-0 vertices and the set $V_1$ of player-1 vertices. An infinite game on $\Gamma$ is played for infinitely many rounds by two players moving a pebble along the edges of the weighted graph $G^I$. In the first round, the pebble is on some vertex $v \in V$. In each round, if the pebble is on a vertex $v \in V_i$ ($i = 0,1$), then player $i$ chooses an edge $(v,v') \in E$ and the next round starts with the pebble on $v'$. A play in the game graph $\Gamma$ is an infinite sequence $p = v_0v_1\ldots v_n\ldots$ such that $(v_i,v_{i+1}) \in E$ for all $i \geq 0$. A strategy for player $i$ ($i = 0,1$) is a function $\sigma : V^* \cdot V_i \to V_i$, such that for all finite paths $v_0v_1\ldots v_n$ with $v_n \in V_i$, we have $(v_n,\sigma(v_0v_1\ldots v_n)) \in E$. We denote by $\Sigma_i$ ($i = 0,1$) the set of strategies for player $i$. A strategy $\sigma$ for player $i$ is memoryless if $\sigma(p) = \sigma(p')$ for all sequences $p = v_0v_1\ldots v_n$ and $p' = v'_0v'_1\ldots v'_n$ such that $v_n = v'_n$. We denote by $\Sigma_i^M$ the set of memoryless strategies of player $i$. A play $v_0v_1\ldots v_n\ldots$ is consistent with a strategy $\sigma$ for player $i$ if $v_{j+1} = \sigma(v_0v_1\ldots v_j)$ for all positions $j \geq 0$ such that $v_j \in V_i$. Given an initial vertex $v \in V$, the outcome of two strategies $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$ in $v$ is the (unique) play outcome$(v,\sigma_0,\sigma_1)$ that starts in $v$ and is consistent with both $\sigma_0$ and $\sigma_1$. Given a memoryless strategy $\pi_i$ for player $i$ in the game $\Gamma$, we denote by $G^I(\pi_i) = (V,E,\pi_i,w)$ the weighted graph obtained by removing from $G^I$ all edges $(v,v')$ such that $v \in V_i$ and $v' \neq \pi_i(v)$.
Energy Games \([3, 2]\) An energy game (EG) is an infinite game on the game graph \(\Gamma\), where the goal of player 0 is to construct an infinite play \(v_0v_1 \ldots v_n \ldots\) such that for some initial credit \(c \in \mathbb{N}\):

\[
c + \sum_{i=0}^{j-1} w(v_i, v_{i+1}) \geq 0 \quad \text{for all } j \geq 0 \quad (1)
\]

The quantity \(c + \sum_{i=0}^{j-1} w(v_i, v_{i+1})\) is called the energy level of the play prefix \(v_0v_1 \ldots v_j\). Given a credit \(c\), a play \(p = v_0v_1 \ldots\) is winning for player 0 if it satisfies (1), otherwise it is winning for player 1. A vertex \(v \in V\) is winning for player \(i\) if there exists an initial credit \(c\) and a winning strategy for player \(i\) from \(v\) for credit \(c\). In the sequel, we denote by \(W_i\) the set of winning states for player \(i\). Energy games are memoryless determined [2], i.e. for all \(v \in V\), either \(v\) is winning for player 0, or \(v\) is winning for player 1, and memoryless strategies are sufficient.

**Theorem 1** ([2]). Let \(\Gamma = (V, E, v_0, w, \langle V_0, V_1 \rangle)\) be an EG, for all \(v \in V\), the following four statements are equivalent:

1. \(\exists \sigma_0 \in \Sigma_0 \cdot \forall \sigma_1 \in \Sigma_1 \cdot \text{outcome}_\Gamma(v, \sigma_0, \sigma_1)\) is winning for player 0;
2. \(\forall \sigma_1 \in \Sigma_1 \cdot \exists \sigma_0 \in \Sigma_0 \cdot \text{outcome}_\Gamma(v, \sigma_0, \sigma_1)\) is winning for player 0;
3. \(\exists \pi_0 \in \Sigma_0^M \cdot \forall \pi_1 \in \Sigma_1^M \cdot \text{outcome}_\Gamma(v, \pi_0, \pi_1)\) is winning for player 0;
4. \(\forall \pi_1 \in \Sigma_1^M \cdot \exists \pi_0 \in \Sigma_0^M \cdot \text{outcome}_\Gamma(v, \pi_0, \pi_1)\) is winning for player 0;

Using the memoryless determinacy of energy games, the authors of [7] derived the next characterization lemma for EG winning strategies.

**Lemma 1** ([7]). Let \(\Gamma = (V, E, w, \langle V_0, V_1 \rangle)\) be an EG. For all vertices \(v \in V\), for all memoryless strategies \(\pi_0 \in \Sigma_0^M\) for player 0, the strategy \(\pi_0\) is winning from \(v\) if and only if all cycles reachable from \(v\) in the weighted graph \(G^\Gamma(\pi_0)\) are nonnegative.

Given the energy game \(\Gamma = (V, E, v_0, w, \langle V_0, V_1 \rangle)\), the EG decision problem asks whether \(v_0\) is winning for player 0. Such a problem is polynomially equivalent to the corresponding decision problem for so-called mean-payoff games [2, 1].

The algorithm with the currently best (pseudopolynomial) complexity for solving energy games is based on the so-called notion of small energy progress measure [7]. Intuitively, the latter is a condition locally defined on the vertices of the given game graph, tailored to witness the global absence of negative cycles within the subgame induced by a proper strategy for player 0 (cfr. the characterization lemma 1). Formally, the notion of small progress measure is recalled in Definition 1 (below) and relies on the following notation. Given \(\Gamma = (V, E, v_0, w, \langle V_0, V_1 \rangle)\), denote by \(C_\Gamma\) the following set:

\[
C_\Gamma = \{n \in \mathbb{N} \mid n \leq \mathcal{M}_{G^\Gamma} \} \cup \{\top\}.
\]

where:

\[
\mathcal{M}_{G^\Gamma} = \sum_{v \in V} \max(\{0\} \cup \{-w(v, v') \mid (v, v') \in E\})
\]
Moreover, denote by $\leq$ the total order on $C_\Gamma$ defined by $x \leq y$ if and only if either $y = \top$ or $x \leq y \leq M_\Gamma$. Finally, let $\ominus : C_\Gamma \times \mathbb{Z} \rightarrow C_\Gamma$ be the operator such that for all $a \in C_\Gamma$ and $b \in \mathbb{Z}$:

$$a \ominus b = \begin{cases} \max(0, a - b) & \text{if } a \neq \top \text{ and } a - b \leq M_\Gamma \\ \top & \text{otherwise} \end{cases}$$

**Definition 1 ([7]).** Let $\Gamma = (V, E, v_0, w, (V_0, V_1))$ be an EG. A function $f : V \rightarrow C_\Gamma$ is a small energy progress measure for $\Gamma$ if and only if the following conditions hold:

- if $v \in V_0$, then $f(v) \geq f(v') \ominus w(v, v')$ for some $(v, v') \in E$;
- if $v \in V_1$, then $f(v) \geq f(v') \ominus w(v, v')$ for all $(v, v') \in E$.

Given a small energy progress measure $f$ for the game graph $\Gamma = (V, E, v_0, w, (V_0, V_1))$, we denote by $V_f$ the set of states $V_f = \{v \mid f(v) \neq \top\}$. A memoryless strategy $\pi_0^f : V_0 \rightarrow V$ for player 0 is called compatible with $f$ whenever for all $v \in V_0$, if $\pi_0^f(v) = v'$ then $f(v) \geq f(v') \ominus w(v, v')$. The following property holds [7]: if $\pi_0^f$ is a strategy for player 0 compatible with the energy progress measure $f$, then $\pi_0^f$ is a winning strategy for player 0 from all vertices in $V_f$. Formally:

**Theorem 2 ([7]).** Let $\Gamma = (V, E, v_0, w, (V_0, V_1))$ be an EG. For all small energy progress measures $f$ for $\Gamma$, if $\pi_0^f$ is a strategy for player 0 compatible with $f$, then $\pi_0^f$ is a winning strategy for player 0 from all vertices $v \in V_f$, i.e. $V_f \subseteq W_0$. Moreover, $\Gamma$ admits a small energy progress measure $f$ such that $V_f = W_0$.

### 2.2 Difference Logic

Let $B = \{b_1, \ldots, b_n\}$ be a set of boolean variables and $X = \{x_1, \ldots, x_n\}$ be a set of integer variables. The set of atomic formulas of difference logic consists of the boolean variables in $B$ and integer constraints of the form $x_i - x_j \geq c$, $c \in \mathbb{Z}$.

The set $F$ of difference logic formulas is the smallest set containing the atomic formulas which is closed under negation and conjunction (the boolean connectives $\lor, \rightarrow, \leftrightarrow$ are defined in the usual way in terms of the operators of negation and conjunction $\land, \neg$). A $(B, X)$ valuation consists of two functions (overloaded with the name $\alpha$), $\alpha : B \rightarrow \{0, 1\}, \alpha : X \rightarrow \mathbb{Z}$. The valuation is extended to all difference logic formulas by letting $\alpha(x_i - x_j \geq c) = 1$ if and only if $\alpha(x_i) - \alpha(x_j) \geq c$ and applying the obvious rules for boolean connectives. A difference logic formula $\phi$ is satisfied by a valuation $\alpha$ if and only if $\alpha(\phi) = 1$. A formula $\phi$ is satisfiable if it admits a satisfying valuation. The satisfiability problem for difference logic is NP-complete [8].

### 3 Encoding EG Winning Strategies in Difference Logic

In this section we show how to derive a difference logic formula $\phi_\Gamma$ from a given energy game $\Gamma = (V, E, v_0, w, (V_0, V_1))$ such that $\phi_\Gamma$ is satisfiable if and only if player 0 has a winning strategy on $\Gamma$.
In particular, the difference logic formula $\phi_F$ uses the set of $|E|$ integer constants $\{w_{(v,z)} \mid (v, z) \in E\}$ and ranges over the following set of boolean and integer variables:

- for each $v \in V$, there is a boolean variable $n_v$ and an integer variable $c_v$
- for each edge $(v, z) \in E$, there is a boolean variable $m_{(v,z)}$

Given the above variables, $\phi_F \equiv n_{v_0} \land \phi_0 \land \phi_1 \land \phi_\sigma \land \phi_e$ is the conjunction of five subformulas, where $\phi_0, \phi_1, \phi_\sigma, \phi_e$ are defined as follows:

- $\phi_0 \equiv \bigwedge_{v \in V_0} (n_v \rightarrow \bigvee_{(v,z) \in E} m_{(v,z)})$
- $\phi_1 \equiv \bigwedge_{v \in V_0} (n_v \rightarrow \bigwedge_{(v,z) \in E} m_{(v,z)})$
- $\phi_\sigma \equiv \bigvee_{v \in V, v \neq v_0} (\bigvee_{(v,z) \in E} m_{(v,z)} \rightarrow n_z)$
- $\phi_e \equiv \bigvee_{(v,z) \in E} (m_{(v,z)} \rightarrow \psi_{(v,z)})$
- $\psi_{(v,z)} \equiv c_v + w_{(v,z)} \geq c_z$

**Theorem 3.** Player 0 has a winning strategy in the energy game $\Gamma = (V, E, v_0, w, (V_0, V_1))$ if and only if the difference logic formula $\phi_F$ is satisfiable.

**Proof.** ($\Rightarrow$) Let $G_\Gamma(\pi)$ be the graph induced by a winning strategy $\pi$ for player 0 on the energy game $\Gamma = (V, E, v_0, w, (V_0, V_1))$. Consider the assignment $\alpha$ to the variables of $\phi_F$ defined as follows: for each boolean variable $n_v$ (resp. $m_{(v,z)}$) let $\alpha(n_v) = 1$ (resp. $\alpha(m_{(v,z)}) = 1$) if and only if $v$ is a node (resp. $(v,z)$ is an edge) of $G_\Gamma(\pi)$. By definition of $G_\Gamma(\pi)$, the assignment $\alpha$ satisfies $n_{v_0} \land \phi_0 \land \phi_1$. By Theorem 2, $G_\Gamma(\pi)$ admits a small progress measure function $f : W \rightarrow \mathcal{M}_{G_\Gamma(\pi)}$, where $W$ is the set of vertices of $G_\Gamma(\pi)$. For each integer variable $c_v$ in $\phi_F$, define $\alpha(c_v) = f(v)$ if $v \in W$. Since $\pi$ is a winning strategy on $\Gamma$ for player 0, the assignment $\alpha$ satisfies also the last conjunct $\phi_\sigma$ in $\phi_F$. Therefore, $\alpha \models \phi_F$.

($\Leftarrow$) Suppose that $\alpha$ is a satisfying variable assignment of $\phi_F$. Define the following game $\Gamma' = (V', E', v_0, w', (V'_0, V'_1))$: $v \in V'$ (resp. $(v,z) \in E'$) if and only if $\alpha(n_v) = 1$ (resp. $\alpha(m_{(v,z)} = 1$) and for each $(v,z) \in E'$ let $w'(v,z) = w_{(v,z)}$. Since $\alpha$ satisfies $n_{v_0} \land \phi_0 \land \phi_1 \land \phi_\sigma$, we derive that $\Gamma'$ is a non empty subgame of $\Gamma$. Hence, since $\alpha$ satisfies also $\phi_e$, by Theorem 2 we deduce that $V' \subseteq W_0$ and $\Gamma'$ induces a winning strategy for player 0 on $\Gamma$. □

**Theorem 4.** Given an energy game $\Gamma = (V, E, v_0, w, (V_0, V_1))$, the size of the difference logic formula $\phi_F$ is $\mathcal{O}(|E|)$, even if $\phi_F$ is required to be in CNF.

**Proof.** Each subformula $\phi_0 \land \phi_1, \phi_\sigma, \phi_e$ has size $\mathcal{O}(|E|)$, while the remaining conjunct $n_{v_0}$ in $\phi_F$ has size 1. $\phi_F$ can be rewritten in CNF with a constant blow up by reformulating the conjuncts $\phi_0, \phi_1, \phi_\sigma$ and $\phi_e$ using the boolean equivalences:

- $\chi \rightarrow (\phi \land \psi) \equiv (\chi \rightarrow \phi) \land (\chi \rightarrow \psi)$
- $(\phi \lor \psi) \rightarrow \chi) \equiv (\phi \rightarrow \chi) \land (\psi \rightarrow \chi)$

□
4 Solving Energy Games by a Reduction to SAT

In this section, we present an encoding for the difference logic formula $\phi_F$ associated to a given energy game $F$ into propositional logic, i.e. the subset of difference logic with boolean variables only. Clearly, all that remains to be done is to translate the integer variables and the constraints on them of the form $c_v + w_{(v,z)} \geq c_z$ inside the conjunct $\phi_F$.

Let $\Gamma = \langle V, E, v_0, w, \{V_0, V_1\} \rangle$ be the energy game underlying $\phi_F$. By Theorem 2 the domain of the integer variables in $\phi_F$ can be bounded by $M_{G_F} \leq V \cdot W$, where $W$ is the maximum absolute weight in $\Gamma$. Let $k = \lceil \log(M_{G_F} + W) \rceil$ be the number of bits necessary to code $M_{G_F}, W$.

For each edge $(v, z) \in E$, let $\vec{w}(v, z) = w_1 \ldots w_k$ be the boolean encoding of $|w_{(v,z)}|$ using $k$ boolean variables, let $e^v_1, e^z_1, \ldots e^z_k, s^v_{(v,z)} \ldots s^v_{(v,z)}, r^v_0 \ldots r^v_{k}$, be further boolean variables and consider the following propositional formulas:

- If $w_{(v,z)} \geq 0$:
  - $\text{CURRY}(v, z, k) \equiv \neg r^v_k$
  - for $i = k \ldots 1$:
    - $\text{SUM}(v, z, i) \equiv s^v_i \iff (\neg e^v_i \land \neg w_i \land \neg r^v_i) \lor (e^v_i \land w_i \land r^v_i)$
    - $\lor (e^v_i \land w_i \land \neg r^v_i) \lor (e^v_i \land w_i \land r^v_i)$
    - $\text{CURRY}(v, z, i - 1) \equiv r^v_{i-1} \iff (\neg e^v_i \land w_i \land r^v_i) \lor (\neg e^v_i \land \neg w_i \land r^v_i)$
    - $\lor (\neg e^v_i \land w_i \land \neg r^v_i) \lor (\neg e^v_i \land w_i \land r^v_i)$

- If $w_{(v,z)} < 0$:
  - $\text{CURRY}(v, z, k) \equiv \neg r^v_k$
  - for $i = k \ldots 1$:
    - $\text{SUM}(v, z, i) \equiv s^v_i \iff (\neg e^z_i \land \neg w_i \land \neg r^v_i) \lor (\neg e^z_i \land w_i \land \neg r^v_i)$
    - $\lor (\neg e^z_i \land \neg w_i \land r^v_i) \lor (\neg e^z_i \land w_i \land r^v_i)$
    - $\text{CURRY}(v, z, i - 1) \equiv r^v_{i-1} \iff (\neg e^z_i \land w_i \land r^v_i) \lor (\neg e^z_i \land \neg w_i \land r^v_i)$
    - $\lor (\neg e^z_i \land w_i \land \neg r^v_i) \lor (\neg e^z_i \land w_i \land r^v_i)$
\begin{itemize}
  \item \text{CURRY}(v, z, 0) \equiv \neg t_0^{(v, z)}
  \item \text{GEQ}(v, z, 1) \equiv e_1^v \Rightarrow s_1^{(v, z)}
  \item for \( i = k \ldots 1 \):
    \[ \text{GEQ}(v, z, i) \equiv (e_i^v) \Rightarrow s_i^{(v, z)} \land ((e_i^v \lor \neg s_i^{(v, z)}) \Rightarrow \text{GEQ}(v, z, i - 1) \]
\end{itemize}

Let \( \phi'_f \) be the propositional logic formula obtained by replacing each integer constraint in \( \phi_f \) of the form \( c_v + w_{(v, z)} \geq c_z \) by the propositional formula \( \text{GEQ}(v, z, k) \)

\textbf{Theorem 5.} Player 0 has a winning strategy in the energy game \( \Gamma = (V, E, v_0, w, (V_0, V_1)) \) if and only if the propositional logic formula \( \phi'_f \) is satisfiable.

\textit{Proof.} (\( \Rightarrow \)) Let \( G_f(\pi) \) be the graph induced by a winning strategy \( \pi \) for player 0 on the energy game \( \Gamma = (V, E, v_0, w, (V_0, V_1)) \). Consider the assignment \( \alpha \) to the variables of \( \phi_f \) defined as follows: for each boolean variable \( n_v \) (resp. \( m_{(v, z)} \)) let \( \alpha(n_v) = 1 \) (resp. \( \alpha(m_{(v, z)}) = 1 \)) if and only if \( v \) is a node (resp. \( (v, z) \) is an edge) of \( G_f(\pi) \). By Theorem 3, the assignment \( \alpha \) satisfies \( n_v \land \phi_0 \land \phi_1 \). By Theorem 2, \( G_f(\pi) \) admits a small progress measure function \( f : W \rightarrow M_{G_f(\pi)} \), where \( W \) is the set of vertices of \( G_f(\pi) \). For each \( (v, z) \in E \) such that \( w(v, z) \geq 0 \) (resp. \( w(v, z) < 0 \) :

- let \( \alpha(e_i^v) \ldots \alpha(e_k^v) \) be the boolean code of \( f(v) \)
- let \( \alpha(e_i^z) \ldots \alpha(e_k^z) \) be the boolean code of \( f(z) \)
- let \( \alpha(s_1^{(v, z)}) \ldots \alpha(s_k^{(v, z)}) \), \( \alpha(r_0^{(v, z)}) \ldots \alpha(r_k^{(v, z)}) \) be the boolean code of the sum \( f(v) + w(v, z) \) (resp. \( f(z) + (-w(v, z)) \)) and the corresponding curry bits.

Since \( \pi \) is a winning strategy on \( \Gamma \) for player 0, the assignment \( \alpha \) satisfies the propositional formula \( \text{GEQ}(v, z, k) \). Therefore, \( \alpha \models \phi'_f \).

(\( \Leftarrow \)) Suppose that \( \alpha \) is a satisfying variable assignment of \( \phi_f \). Define the following game \( \Gamma' = (V', E', v_0, w', (V_0', V_1')) \): \( v \in V' \) (resp. \( (v, z) \in E' \)) if and only if \( \alpha(n_v) = 1 \) (resp. \( \alpha(m_{(v, z)}) = 1 \)) and for each \( (v, z) \in E' \) let \( w'(v, z) = w(v, z) \). Since \( \alpha \) satisfies \( v_0 \land \phi_0 \land \phi_1 \land \phi_\sigma \), we derive that \( \Gamma' \) is a non empty subgame of \( \Gamma \). Hence, since \( \alpha \) satisfies also \( \phi_\sigma \), by Theorem 2 we deduce that \( V' \subseteq V_0 \) and \( \Gamma' \) induces a winning strategy for player 0 on \( \Gamma \).

\textbf{Theorem 6.} Given an energy game \( \Gamma = (V, E, v_0, w, (V_0, V_1)) \), the size of the propositional logic formula \( \phi'_f \) is \( \mathcal{O}(|E| \cdot \lceil \log(V \cdot W) \rceil) \), even if \( \phi'_f \) is required to be in CNF.

\section*{References}

The Language of Initially Connected Deterministic Finite Automata*
(Extended Abstract)

Rogério Reis, Emanuele Rodaro

Centro de Matemática, Universidade do Porto
R. Campo Alegre 687, 4169-007 Porto, Portugal
e-mail: rvr@dcc.fc.up.pt, emanuele.rodaro@fc.up.pt

1 Locally finite \(\infty\)-pushdown automata

A word \(w\) on an alphabet \(\Sigma\), and alphabets here will be considered not necessarily finite, is a finite string \(w = w_1w_2\ldots w_n\) with \(w_i \in \Sigma\). Let \(|w| = n\). The empty word is denoted by \(\varepsilon\) and the set containing the all words on \(\Sigma\) is denoted by \(\Sigma^*\), and the set of all non-empty words denoted by \(\Sigma^+\). The support of \(w\) is the set \(\text{Supp}(w) = [1, n]\). By \(w[i, j]\) we denote the factor \(w_i\ldots w_j\), with the convention that \(w[i] = w_i\). Mostly we deal with the infinite alphabet \(\mathbb{N}_0 = \{0, 1, \ldots\}\) of the natural numbers.

An \(\infty\)-pushdown automaton of depth \(m \geq 1\) is an \(8\)-tuple

\[
\mathcal{A} = (Q, \Sigma, \Gamma, \delta, m, q_0, \bot, F)
\]

where \(Q\) is the finite set of states, \(\Sigma\) is the input alphabet, \(\Gamma\) is called the stack alphabet and \(\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\})) \times \Gamma^m \times Q \times \Gamma^*\) while \(q_0 \in Q, F \subseteq Q, \bot\) are respectively the initial state, the set of final states and the initial symbol of the stack. An instruction \((q, x, \alpha, p, \beta) \in \delta\) is interpreted in the usual way as for pushdown automata: if the machine is in state \(q\), it is reading the symbol \(x\) on the tape and \(\alpha\) is the the word of length \(m\) representing the last \(m\) elements in the stack, then the machine goes to the state \(p\) and substitutes \(\alpha\) with \(\beta\). A configuration of \(\mathcal{A}\) is a tuple \((q, u, \gamma)\) where \(q\) is the state of machine \(u \in \Sigma^*\) is the part of the input that have to be read and \(\gamma \in \Gamma^*\) is the content of the stack. As usual, the transition relation \(\delta\) induces a relation \(\vdash_{\mathcal{A}}\) on the set of configurations defined by

\[
(q, ax, \alpha\gamma) \vdash_{\mathcal{A}} (p, x, \alpha'\gamma')
\]

for \(\alpha \in \Gamma^m\), \(a \in (\Sigma \cup \{\varepsilon\})\), \(\alpha', \gamma, \gamma' \in \Gamma^*\) and \((q, a, \alpha, p, \alpha') \in \delta\). We also write

\[
(q, y, \zeta) \vdash_{\mathcal{A}} (q', y', \zeta')
\]

whenever

\[
(q, y, \zeta) = (q_0, y_0, \zeta_0) \vdash_{\mathcal{A}} (q_1, y_1, \zeta_1) \vdash_{\mathcal{A}} \ldots \vdash_{\mathcal{A}} (q_n, y_n, \zeta_n) = (q', y', \zeta')
\]

* Work partially supported by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT – Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011 and CANTET-PTDC/EIA-CCO/101904/2008.

** Partialy supported by FCT project SFRH/BPD/65428/2009.
and by $\vdash^*_{\mathcal{A}}$ we denote the transitive closure of $\vdash_{\mathcal{A}}$. The language recognized by $\mathcal{A}$ is the set of words $w \in \Sigma^*$ such that $(q_0, w, \bot^m) \vdash^*_{\mathcal{A}} (q, \epsilon, \bot^m)$ with $q \in F$ and it is denoted by $L(\mathcal{A})$.

Assume now that $\Sigma$ is endowed with a locally finite partial order $\leq$ with minimum 0 which is also $\lor$-semilattice (lf$\mathcal{s}$-order), equivalently,

- $[x, x'] = \{ y \in \Sigma : x \leq y \leq x' \}$ is finite (locally finiteness);
- for each pair of elements $x, x'$, $x \lor x' = \sup \{x, x'\}$ is defined.

It is easy to see that for any finite set $\Sigma' \subseteq \Sigma$ the element $\sup \{\Sigma'\}$ is defined and clearly satisfies $\Sigma' \subseteq [0, \sup \{\Sigma'\}]$. For a word $w \in \Sigma^+$, let $\Sigma'$ be the set of letters appearing in $w$, in the sequel we put $\sup \{w\} = \sup \{\Sigma'\}$.

**Definition 1 (locally finite $\infty$-pushdown automaton)** A locally finite $\infty$-pushdown automaton is a tuple $\mathcal{A} = (Q, (\Sigma, \leq), \Gamma, \delta, m, q_0, \bot, F)$ such that

- $(\Sigma, \leq)$ is a lf$\mathcal{s}$-order;
- There is a computable function which associate to each $x \in \Sigma$ a finite subset $\Gamma_x \subseteq \Gamma$ such that if $\mathcal{A}_x = (Q, [0, x], \Gamma_x, \delta_x, m, q_0, \bot, F)$ denotes the pushdown automaton with
  $$\delta_x = \delta \cap (Q \times ([0, x] \cup \{\epsilon\}) \times \Gamma_x \times Q \times \Gamma_x^*)$$
  then the following condition holds:

$$\forall w, w' \in [0, x]^* : \quad (q_0, w, \bot^m) \vdash^*_{\mathcal{A}} (p, w', \zeta) \Leftrightarrow (q_0, w, \bot^m) \vdash^*_{\mathcal{A}_x} (p, w', \zeta)$$  \hspace{1cm} (1)

Using the definition it is not difficult to check that the following proposition holds.

**Proposition 1.** Let $\mathcal{A} = (Q, (\Sigma, \leq), \Gamma, \delta, m, q_0, \bot, F)$ be a locally finite directed pushdown automata, then the membership problem is decidable.

Therefore we are dealing with machines that are not as powerful as Turing machines. The class of languages recognized by locally finite $\infty$-pushdown automata is called *locally finite-infinite context-free language* and denoted by **LF-ICF**. The introduction of these machines is justified in the interest of the expressivity of locally finite-$\infty$-pushdown automata. It is straightforward to see that this class is closed for the standard operations of union, concatenation and Kleene’s closure and that it strictly contains **CFL**. For instance, in next section we present a way to encode initially connected deterministic finite automata (ICDFA) as particular strings of integers. These encoding allow us to define the language $L_{\Sigma}$ of all the strings of integers representing ICDFA’s on the same alphabet $\Sigma$, and to prove that $L_{\Sigma} \in \textbf{LF-ICF}$.

There are many way to encode a ICDFA, for instance in [2] it is proposed a bijection between initially connected deterministic finite automata and some diagrams, instead in [1] the authors present for the first time a way to encode a ICDFA by means of some string of integers satisfying particular combinatorial conditions. What we do here is very similar in some aspect to the code described in [1], however, our approach has the advantage of being more feasible to be described by locally finite $\infty$-pushdown automata.
2 The language of initially connected deterministic finite automata

We recall some standard notations and basic definitions needed henceforth. A rooted plane tree (also called ordered tree) consists of a set of vertices each of which has an associated total ordered (possibly empty) list of vertices called its children. Let \( A \) be a finite alphabet. Let \( k \geq 1 \), a \( k \)-regular \( A \)-labeled plane forest is a \( 3 \)-tuple \((V, \sigma, \lambda)\) where \( V \) is the set of vertices, the map \( \lambda : V \to A \) is called the labeling function, while the map \( \sigma : V \to V^k \cup \{ \emptyset \} \) is the function such that if \( \sigma(v) \) is non-empty, then it is the \( k \)-tuple of ordered children having \( v \) as parent, or if \( \sigma(v) = \{ \emptyset \} \), then \( v \) has no child and it is called a leaf. The relation of being a child can be transitively extended and we say that \( v' \) is a descendant of \( v \) if there is a sequence \( v_0, \ldots, v_m \) such that \( v = v_0, v' = v_m \) and \( v_{i+1} \) is a child of \( v_i \) for \( i = 0, \ldots, m - 1 \). A \( k \)-regular \( A \)-labeled rooted plane tree is a \( 4 \)-tuple \((V, \sigma, \lambda, v_0)\), where \((V, \sigma, \lambda)\) is a \( k \)-regular \( A \)-labelled plane forest where \( v_0 \in V \) is the root and for any \( v \neq v_0 \), \( v \) is a descendent of \( v_0 \). In the sequel, when \( \sigma(v) \neq \{ \emptyset \} \), for \( i \in [1, k] \) we denote the \( i \)-th component of \( \sigma(v) \) by \( \sigma(v)_i \).

The main object studied in this section are the initially connected deterministic finite automata (ICIDA for short). An ICIDA is a tuple \( \mathcal{A} = (Q, \Sigma, \delta, q_0) \) where \( Q \) is the set of states, \( \Sigma \) is a finite alphabet, \( \delta : Q \times \Sigma \rightarrow Q \) is the action of the alphabet on \( Q \). This action can be naturally extended to \( \Sigma^* \) in the obvious way. The special state \( q_0 \) is used to fulfill the condition of initially connectedness, i.e. for any \( q \in Q \) there is a word \( w \in \Sigma^* \) such that \( \delta(q_0, w) = q \). When the automaton is clear from the context, we omit the action \( \delta \) and we write \( q \cdot w \) instead of \( \delta(q, w) \). In the context of ICIDA’s an isomorphism \( \mathcal{A} \rightarrow \mathcal{A}' \) between the two initially connected automata \( \mathcal{A} = (Q, \Sigma, \delta, q_0) \), \( \mathcal{A}' = (Q', \Sigma, \delta', q_0') \) is a bijection \( \varphi : Q \rightarrow Q' \) such that \( \delta' (\varphi(q), a) = \varphi (\delta(q,a)) \) for all \( q \in Q, a \in \Sigma \), and satisfying also the condition \( \varphi(q_0) = q_0' \). If there is an isomorphism \( \mathcal{A} \rightarrow \mathcal{A}' \) we also use the notation \( \mathcal{A} \simeq \mathcal{A}' \). Let us consider a total order \( < \) on the elements of \( \Sigma = \{ a_1, \ldots, a_k \} \) by \( a_1 < a_2 < \ldots < a_k \). This order induces a total order on the set \( Q \) using a depth-first search function defined recursively by Algorithm 1. Starting with \( L = (q_0) \), \( \text{Search}(q_0, L) \) returns a list \( L \) of the \( n \) states \((q_0, \ldots, q_{n-1})\) with \( q_0 = q_0, n = |Q| \). If we denote by \( \varphi_{\mathcal{A}} : Q \rightarrow [0, n - 1] \) the one-to-one function such that \( \varphi_{\mathcal{A}}(q) \) gives the position of a state \( q \) in the list \( L \), then this function orders the states of

\[ \text{Algorithm 1 The pseudo-code for a depth-first search in } \mathcal{A} \text{ starting at } q. \]

```
1: procedure Search(q, L)
2:    for i = 1 → k do
3:      if q · aᵢ ∉ L then
4:        L.push(q · aᵢ)
5:        Search(q · aᵢ, L)
6:    end if
7: end for
8: end procedure
```
\[ A \text{ canonically.} \] For canonically we mean that for any isomorphism \( \psi : A \to A' \) between two ICDFA’s, then \( \varphi_A = \varphi_{A'} \circ \psi \). Thus we can associate to \( A \) its canonical representative given by the ICDFA

\[ A^* = \langle [0, n - 1], [1, k], \delta^*, 0 \rangle \]

where \( \delta^*(i, j) = \varphi_A(\delta(\varphi_A^{-1}(i), a_j)) \). This is a canonical representative in the sense that if \( A \simeq A' \), then \( A^* = A'^* \).

![Diagram](image)

**Fig. 1.** An example of ICDFA \( B \) and the corresponding 2-labelled plane tree \( T(B^*) \).

We can associate to \( A^* \) a \( k \)-regular \([0, n - 1]\)-labeled plane tree \( T(A^*) = (V, \sigma, \lambda, 0) \), defined in the following way. The set of vertices

\[ V = [0, n - 1] \cup \{ q_{j,i} : \text{ for } j, i \text{ with } \delta^*(j, i) \leq j \} \]

and the map \( \sigma \) defined on \( j \in [0, n - 1], i \in [1, k] \) by

\[ \sigma(j) = \begin{cases} q_{j,i} & \text{if } \delta(j, i) \leq j \\ \delta(j, i) & \text{otherwise} \end{cases} \]

while \( \sigma(q_{j,i}) = \{ \emptyset \} \) for any \( q_{j,i} \in V \setminus [0, n - 1] \). The labeling map is defined by \( \lambda(j) = j \) for \( j \in [0, n - 1] \), and \( \lambda(q_{j,i}) = \delta^*(j, i) \) for \( q_{j,i} \in V \setminus [0, n - 1] \). Consider the procedure described in the pseudo Algorithm 2, if we run \( S(0, \lambda) \), a list \( \lambda \) is returned, that seen as a string on \([0, n - 1]^*\), is the word obtained by reading the labels of the vertices of \( T(A^*) \) while making a depth-first traversal of the tree \([4]\). We call such string the depth-first canonical string of \( A \) and denoted by \( S_D(A) \). For instance, in Fig. 1, it is depicted an example of ICDFA \( B \). Applying Algorithm 1 with respect to the order \( a < b \) on the alphabet, we obtain \( \varphi_B(q_i) = i \) for \( i = [0, 3] \), and on the right of Fig. 1 we exhibit the associated 2-labelled plane tree \( T(B^*) \). The depth-first canonical string of \( B \) is given by \( S_D(B) = 011231000 \). Using this encoding we define the language of ICDFA’s on the alphabet \( \Sigma \) as the set \( L_\Sigma \subseteq \mathbb{N}^* \) defined by

\[ L_\Sigma = \{ S_D(A) : A \text{ is a ICDFA on } \Sigma \} . \]
The first and natural task is to give a combinatorial characterization to such words. To achieve this, we give first some few definitions. Given a word $w \in [0, n - 1]^*$ with $w = w_1 \ldots w_m$, an index $j \in \text{Supp}(w)$ is called a flag if $w_j \notin \{w_1, \ldots, w_{j-1}\}$, in other words, a flag is the index $j$ of the first occurrence $w_j$ in $w$. For instance, in the example in Fig. 1 the associated depth-first canonical string of $\mathcal{B}$ is $S_D(\mathcal{B}) = 011231000$ and the flags are pinpointed by the underlined elements in $S_D(\mathcal{B}) = \underline{0} \underline{1} 1 1 2 3 1 0 0 0$. A block of $w$ is a factor of $w$ defined recursively as follows

- $w[i, i + k]$ where $i$ is a flag and $w[i + j]$ is not a flag for all $j \in [1, k]$ is a block,
- there is a factorization $w[i, j] = w[i]w[i_1, i_2 - 1] \ldots w[i_k, j]$ into $k + 1$ factors, where $i$ is a flag and for each $s \in [1, k]$, $w[i_s, i_{s+1} - 1]$ is either a block or $i_s = i_{s+1} - 1$ and $i_s$ is not a flag.
- Nothing else is a block.

Thus a block can be always factorized as $w[i, j] = w[i]w[i_1, i_2 - 1] \ldots w[i_k, j]$ where $i$ is a flag and for each $j \in [1, k]$, $w[i_j, i_{j+1} - 1]$ is either a block or $i_j = i_{j+1} - 1$ is not a flag, in such case $i_j$ is called the external child of the flag $i$. We also call to $w[i, j]$ the block associated (or corresponding) to the flag $i$. For instance all the blocks of $S_D(\mathcal{B}) = 011231000$ presented with their factorizations are: $0(11(2310)0)0, 11(2310)0, 2(310)0, 310$. Using the notion of block we can characterized the strings in $\mathcal{L}_\Sigma$

**Theorem 2.** A word $w \in \mathcal{L}_\Sigma$ if and only if the following conditions occur:

1. $w$ is a block,
2. put $n - 1 = \text{sup}(w)$, for any $i \in [0, n - 1]$ there is a $j \in \text{Supp}(w)$ with $w[j] = i$,
3. for any flag $j \in \text{Supp}(w)$ we have $w[i] < w[j]$ for all $i < j$ (where $<$ denotes the usual strict order on $\mathbb{N}_0$).

Using this characterization it is possible to prove the following main theorem.

**Theorem 3.** Let $\Sigma = \{a_1, \ldots, a_k\}$, then $\mathcal{L}_{\Sigma} \in \text{LF-ICF}$.

The proof of this fact is essentially based on proving that $\mathcal{L}_{\Sigma}$ is equal to the language recognized by the locally finite $\infty$-pushdown automaton defined by:

$$\mathcal{A} = (Q, (\mathbb{N}_0, \leq), \Gamma, \delta, 2, q_0, \bot, \{q_0\})$$

*Proceedings of ICTCS 2013*
where $Q = \{q_0, q_1\}$, $\Gamma = \{0, \ldots, k\} \times \mathbb{N}_0$ and $\delta : Q \times \mathbb{N}_0 \times \Gamma^2 \to Q \times \Gamma^*$ is the function defined by $\delta(q_0, 0, \bot^2) = (q_0, (0,0) \bot)$ and where

$$
\delta(q_i, a, (j, m)z) = \begin{cases}
(q_0, (j + 1, m)(j, m)z), & \text{if } i = 0, j < k, a < m; \\
(q_0, (0, m + 1)(j, m)z), & \text{if } i = 0, j < k, a = m + 1; \\
(q_1, (j, m)z), & \text{if } i = 0, j = k, a = e; \\
(q_1, (t, m)), & \text{if } i = 1, j \neq 0, a = e, z = (t, m'); \\
(q_0, (t + 1, m)(t, m)), & \text{if } i = 1, j = 0, a = e, z = (t, m'); \\
(q_0, \bot), & \text{if } i = 1, j = 0, a = e, z = \bot;
\end{cases}
$$

It would be interesting to explore if there are other classes of DFA’s, like for instance strongly connected or synchronizing DFA’s, that can be represented using this formalism. Beside the compactness and the fact that we frame a class of important objects like ICDFA’s in a quite natural extension of context-free languages, one reason why we are interested in this kind of formalism is in the possibility of giving grammars to recursively generated important combinatorial objects. This is important for instance in order to apply Boltzmann samplers [3] for the random generations of important classes of DFA’s.

Acknowledgement

The work is partially supported by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT – Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011, CANTE-PTDC/EIA-CCO/101904/2008. The second author also acknowledges the support of the FCT project SFRH/BPD/65428/2009.

References

On the Multiple Common Substring Problem

Maxime Crochemore\textsuperscript{1,2}, Costas S. Iliopoulos\textsuperscript{3}, Alessio Langiu\textsuperscript{1}, and Filippo Mignosi\textsuperscript{3}

\textsuperscript{1} Department of Informatics, King’s College London, London, UK
\texttt{Maxime.Crochemore|Costas.Iliopoulos|Alessio.Langiu@kcl.ac.uk}
\textsuperscript{2} Université Paris-Est, Marne-La-Vallée, France
\textsuperscript{3} DISIM, Università dell’Aquila, L’Aquila, Italy
\texttt{Filippo.Mignosi@di.univaq.it}

Abstract. The Colored Sub-Tree Problem (CST) is the following: given a tree with colored leaves, find for each internal node \( v \) the number of colors that are present in the leaves of the subtree rooted in \( v \). We show that for this problem we can use a well known union-find data structure maintaining an amortized overall linear time and space in the RAM model. Indeed the number of operations needed to settle above problem is proportional to the tree size. We further show a second solution that uses less space but where the worst case time has to be multiplied by the inverse of the Ackerman function. Given a set \( D \) of \( d \) documents, the Multiple Common Substring (MCS) problem asks, for any integer \( 2 \leq k \leq d \), the longest substring which appears in \( k \) documents. MCS is a well studied problem having a wide range of applications in bioinformatics: from microarrays to DNA sequences alignments and analysis. This problem has been solved by Hui (2000) who uses a famous constant-time solution to the Lowest Common Ancestor (LCA) problem in trees coupled with the use of suffix trees. A data structure for the LCA problem, although linear in space and construction time, introduces a multiplicative constant in both space and time that reduces the range of applications in many biological applications.

In this article we present a very simple method for solving the MCS problem using suffix trees and above CST solution. In turn, we show how this simple algorithm can be modified in order to work with other space efficient data structures such as the Enhanced Suffix Arrays (ESA).

Introduction

One of the classic \textit{union-find} data structure for disjoint sets has myriads of practical applications (see for instance\cite{10}). The running time, due to a multiplicative inverse of the Ackermann function, is not linear. Due to its simplicity, and to the small constants involved, it turns out to be, in practice, faster of other linear alternatives. From a theoretical point of view, Tarjan et al. \cite{9} showed how to get rid of the multiplicative Ackermann function when the the tree of the union is known in advance. In \cite{15,14,16} it is shown that if the \textit{finds} are performed in post order (at most one find for each element) then the overall amortized time is truly linear.

More recently appeared other algorithms that perform union-find in linear time (see for instance \cite{7,12} ) that show that under some special conditions one can perform the classical union-find in linear time, even adding a new operation \textit{delete} (see \cite{6}). Other more recent papers deal with \textit{union-find-delete} data structure for disjoint sets (see \cite{2,3}).
In this paper we will make use of the Tarjan et al. linear time algorithm and, in order to reduce space, we will also use the delete operation.

The Colored Sub-Tree Problem (CST) is the following: given a tree with colored leaves, find for each internal node $v$ the number of colors that are present in the leaves of the subtree rooted in $v$. The Multiple Common Substring Problem (MCS) of a set of strings is the following: given $d$ strings (or documents) of total length $n$ find one longest substring that appears in $k$ strings for every $k$ between 2 and $d$.

Both above problems were settled in linear time by L. C. K. Hui in [13] and are reported, together to the Hui’s solutions, by D. Gusfield in [11, Section 7.6, 9.4]. These solutions uses a famous constant-time solution to the Lowest Common Ancestor (LCA) problem in trees coupled with the use of suffix trees. A data structure for the LCA, although it is linear in space and time, introduces a multiplicative constant in both space and time that reduces the range of applications in many biological applications. Indeed the research of efficient (mainly in space) data structure that can be used in the place of suffix trees has become an independent research field. Up to less than a decade ago the most commonly used data structures are Suffix Trees, Suffix Arrays, DAWGs and CDAWGs. Usually any problem that can be settled by the aid of one of such data structure can also be settled by using any of the other ones. Despite this fact the passage from one data structure to another is not automatic nor always easy. Each of these structures has some advantage and some disadvantage. Some relation among the data structures and their size is reported in [4]. The size of an implementation of the above data structures is often evaluated by the average number of bytes necessary to store one letter of the original text. It is commonly admitted that these ratios are 4 for suffix arrays, 9 to 11 for suffix trees, and 5 for CDAWGs (cf. [4] for further information).

In the meantime other space-efficient related data structures started to appear in research papers. For instance Enhanced Suffix Array showed to have the potential to replace in many applications suffix trees with a ratio of 5 bytes per letter with smart implementations. This potential is claimed in [1] but still few applications, even if important, were described in this paper. Succinctness coupled with Enhanced Suffix Array (that we call here Succinct Enhanced Suffix Array) seems to have reduced the space required for many applications down to $n H_k \log \sigma + O(n)$ bits, where $H_k$ is the $k$th order entropy of the text and $\sigma$ is the alphabet size assumed to be a polylog of $n$ (see [18,8,5]).

Even if the potential of settling known applications is present, the translation procedure is not automatic at all.

Indeed, for each problem and each data structure one has often to find a dedicated algorithm to solve it. It is clear that if the original algorithm for solving one problem is simple, above task is likely to be feasible.

The Multiple Common Substring Problem (MCS) is one of the problems for which no solution that makes use of one of the recent above space efficient data structure is known up to the present, except a partial solution in [17] that settle it only for one special case ($k = n$) using suffix arrays and enhanced suffix
arrays. Indeed the Hui’s solution, even if not extremely complicated, presents anyhow some technicalities that made hard any translation.

One of the main contribution of this paper is thus to find a new simple solution to the MCS using Suffix Trees. Due to its simplicity, we further show how to extend it in order to use Enhanced Suffix Arrays or even Succinct Enhanced Suffix Arrays.

1 Linear Algorithm

Definition 1. Given a tree with colored leaves, the Colored Sub-Tree Problem (CST) asks, for each internal node \( v \), the number of different colors present in the sub tree rooted in \( v \).

Definition 2. Given a set \( D \) of \( d \) documents, the Multiple Common Substring (MCS) problem finds, for any integer \( 2 \leq k \leq d \), the longest substring which appears in \( k \) documents.

Those two problems are strictly related. Therefore, if we build the generalized suffix tree \( ST_D \) of a set \( D \) of documents and we color any leaf by assigning a different color to any document, the MCS problem can be solved with a simple traverse of \( ST_D \) after that \( ST_D \) has been preprocessed to solve the CST problem.

We present a simple linear time and space algorithm which uses the Tarjan et al. linear time union-find algorithm [9] in order to compute, for any internal node in the suffix tree, the number of different color in the rooted sub tree. Let us assume that \( m \) is the number of different colors in the tree. The nodes in the tree are augmented with a color counter field. Node counters are initialized to 0 for the internal nodes and 1 for leaves. An auxiliary global array \( L \) (\( L \) stands for “last seen”) of size \( m \) is used along the algorithm execution in order to keep track of the previous occurrence of a color \( 1 \leq k \leq m \) during the execution of the algorithm.

The pseudo code is reported in Algorithm 1. The tree \( T \) is given as input. The algorithm initializes the union-find structure following [9]. We recall from [9] that \( \text{union}(x, y) \) create a new set that is the union of the sets containing \( x \) and \( y \). The representative of the new set is the representative of the old set containing \( x \). This operation destroys the old sets containing \( x \) and \( y \). When the tree of the union is known in advance, as it is in our case, the only allowed union operations are of the form \( \text{union}(p(u), u) \), where \( p(u) \) is the father, in the tree, of \( u \). In this situation both \( p(u) \) and \( u \) are always the representatives of the sets they belong to. A variant of the Algorithm 1 which does not require to ask for the father \( p(u) \) of a given node \( u \) is reported in Algorithm 2.

In every internal node there is a counter that will eventually contain the number of all different color in the rooted subtree and it is initialized to 0. The \( L \) array and the \( u \) variable are initialized to \( \text{nill} \). The algorithm proceeds by performing a depth-first search on the tree \( T \). The main idea is to compute the node color counters with a bottom-up strategy, while maintaining the
Algorithm 1 Preprocessing the tree $T$ for the CST problem. (Assume that any node $v$ in $T$ is equipped with a pointer to its father node $p(v)$)

for any node $v$ in $T$ do
    makeset($v$)
end for

$L[i] \leftarrow \text{nill}$, for any $1 \leq i \leq m$.

for any node $v$ in the DFS visit of $T$ do
    if $v$ is a leaf then
        if $L[v_{\text{color}}] \neq \text{nill}$ then
            $r \leftarrow \text{find}(L[v_{\text{color}}])$
            $r_{\text{counter}} \leftarrow (r_{\text{counter}} - 1)$
        end if
        $L[v_{\text{color}}] \leftarrow v$
    end if
    if $v$ is a leaf or any child of $v$ has been visited then
        union($p(v), v$)
        $p(v)_{\text{counter}} \leftarrow (p(v)_{\text{counter}} + v_{\text{counter}})$
    end if
end for

Algorithm 2 Preprocessing the tree $T$ for the CST problem.

for any node $v$ in $T$ do
    makeset($v$)
end for

$L[i] \leftarrow \text{nill}$, for any $1 \leq i \leq m$.

$u \leftarrow \text{nill}$

for any node $v$ in the DFS visit of $T$ do
    if $u \neq \text{nill}$ then
        union($v, u$)
        $v_{\text{counter}} \leftarrow (v_{\text{counter}} + u_{\text{counter}})$
        $u \leftarrow \text{nill}$
    end if
    if $v$ is a leaf then
        if $L[v_{\text{color}}] \neq \text{nill}$ then
            $r \leftarrow \text{find}(L[v_{\text{color}}])$
            $r_{\text{counter}} \leftarrow (r_{\text{counter}} - 1)$
        end if
        $L[v_{\text{color}}] \leftarrow v$
    end if
    if $v$ is a leaf or any child of $v$ has been visited then
        $u \leftarrow v$
    end if
end for
sets of colors via the union-find structure. Those sets are virtually maintained disjoint (as explained below) by using the previous occurrence array $L$. After that a node is visited and all of its children have been visited, we propagate its color counter upward in the tree by making an union operation between such node $v$ (stored as $u$ in the Algorithm 2) and its father $p(v)$ ($v$ in Algorithm 2). The father node $p(v)$ is now the root of the newly created set, as required in [9]. When a leaf $v$ is visited, the last occurrence of the same color $L[v_{\text{color}}]$ is virtually deleted from the old set in order to maintain disjoint the colors associated those sets. We actually decrement the color counter of the representative of the set which contains $L[v_{\text{color}}]$, that is $\text{find}(L[v_{\text{color}}])$. At this point a real delete operation could be executed in the union-find data structure for space saving purpose. The $\text{makeset}(x)$, $\text{union}(x, y)$ and $\text{find}(x)$ are the usual union-find operations.

**Theorem 1.** Given a tree $T$ with colored leaves, the Algorithm 1 correctly computes in linear time the color counters for each internal node $v$ of $T$.

**Proof. Sketch.** The proof is by induction on the number of leaves in the subtree rooted in $v$ and any step of the algorithm. One proves by induction that, for every internal node $v$ of the tree, the color counter of $v$ maintains the number of different colors present in the already visited portion of the subtree rooted in $v$ minus the number of times that a subsequent $\text{find}$ operation has returned $v$ as the set representative. When the number of leaves in the subtree of $v$ is equal to 1, clearly $v$ is a leaf and its color counter is correct by definition, that is, it is initialized to 1 and it is never changed along the algorithm. Let us focus on the nodes involved in the DFS traverse of $T$, since the rest of the tree and their color counters will be never changed. When the number of leaves in the subtree of $v$ is equal to $x$ and the visited part contains $y$ of such leaves, let assume that $1 \leq p \leq m$ is the number of different colors in the visited $y$ leaves. In the next step the algorithm visits another leaf $u$ of $v$. Now, $u$ can be colored by a color already seen in the $y$ leaves, by using the $L$ array, the algorithm decrements the color counter in $v$ that is maintained as the representative of the union-find set containing all the $y$ leaves already visited. Therefore, when the color of $u$ will be propagated up to $v$ it will be added correctly to the color counter of $v$. Otherwise, if $u$ has a color not present in any of the $y$ leaves, it will be normally propagated up to $v$.

## 2 Reducing Space

In this section we describe how to reduce space in using the solution for Colored Sub-Tree Problem for solving the Multiple Common Substring Problem when the tree is the generalized suffix tree of the set of strings and the different color of suffixes are the documents to which they belong.

Even if the two problems are deeply related we observe that the two outputs are totally different in size. Recall that the problem MCS is: given $d$ strings (or documents) of total length $n$ find one longest substring that appears in $k$ strings for every $k$ between 2 and $d$. 

*Proceedings of ICTCS 2013*
While the output of the first problem maintains the tree structure, the output of the second can be an array $P$ that, for each $k$, $k = 2 \ldots d$ contains the length of one longest substring that appears in at least $k$ strings and one pointer to it (i.e. the number of the string that contains it and the position inside that string where it starts). The outputs is then proportional to $d$ (more precisely it has the size of $d$ pointers), while the output of the first problem is proportional to $n$.

First of all we notice that the array $P$ can be obtained during the deep first search of the algorithm of previous section. Indeed if a node has been examined together its rooted subtree during the visit then its color counter will not change anymore. Therefore if the color counter is equal to $k$ and the string depth of the node is greater than the length contained in $P(k)$, then one replace the contents of $P(k)$ with the color counter and a pointer to the string that from the root ends in that node. In this way in $P(k)$ there is always the length and a pointer to the longest string that is common to exactly $k$ substring up to that step of the visit.

As a second observation we notice that the union-find is essentially performed on the leaves of the tree. The internal nodes are necessary in order to have a linear time algorithm as described in [9]. If we renounce to a linear algorithm and we accept an $O(n \alpha(n))$ algorithm, where $\alpha$ is the inverse of the Ackermann function (that is smaller than or equal to 4 for any conceivable input), then we can use one of the data structure allowing the union-find-delete described in [2,3]. We can then modify the algorithm of previous section making the union only between the leaves and performing a true (non virtual) delete of each leaf after we do a $find$ operation on it. In this way the total space required for the union-find-delete data structure is always proportional to the number of documents $d$ if we are careful to use the $makeset$ operation on-line each time we encounter a new leaf in the tree during the visit.

As third and last observation we notice that it is possible to use the enhanced suffix array (ESA) and the succinct enhanced suffix array (SESA) data structures to simulate a depth-first search of a suffix tree. The SESA data structure in particular is space thrifty by using only $n H_k \log \sigma + O(n)$ bits.

Putting the previous three observation altogether, we obtain that we can reduce the space requirements to settle the MCS problem over a collection of $d$ documents of total size $n$ by using $n H_k \log \sigma + O(n)$ bits plus $O(d \times q)$ bits where $q$ is the bit-size of a pointer. Details will be given in the full version of this paper.

References

Abstract. We extend the simply-typed lambda-calculus with a mechanism for dynamic and incremental rebinding of code. Fragments of open code which can be dynamically rebound are values. Differently from standard static binding, which is done on a positional basis, rebinding is done on a nominal basis, that is, free variables in open code are associated with names which do not obey $\alpha$-equivalence. Moreover, rebinding is incremental, that is, just a subset of names can be rebound, making possible code specialization. Finally, rebindings, which are associations between names and terms, are first-class values, and can be manipulated by operators such as overriding and renaming. We define a type system in which the type for a rebinding, in addition to specify an association between names and types (similarly to record types), is also annotated. The annotation says whether or not the domain of the rebinding having this type may contain more names than the ones that are specified in the type. We show soundness of the type system.

1 Introduction

In previous work [1], we have proposed a simple calculus which smoothly integrates binding by position and binding by name. In the former, which is the classical binding of lambda calculus, and parameter passing in most languages, the choice of identifiers does not matter, that is, $\alpha$-equivalence holds. However, many features in programming languages, such as exception handling, method look-up in object-oriented languages, synchronization in process calculi, are based on matching of names, as are usually called identifiers which cannot be $\alpha$-renamed (if not globally in a program) [2, 8]. In order to provide a “minimal” unifying foundation for the two mechanisms, in [1] we have extended the simply typed lambda-calculus with unbound terms, of shape $(x_1 \mapsto X_1, \ldots, x_m \mapsto X_m | t)$ which are values representing “open code”. That is, $t$ may contain free occurrences of variables $x_1, \ldots, x_m$ to be dynamically bound through the global nominal interface $X_1, \ldots, X_m$. To be used, open code should be combined with a rebinding $X_1 \mapsto t_1, \ldots, X_m \mapsto t_m$.

In this paper, we propose a variant of the calculus in [1] with new features which allows more flexible manipulation of code. Notably:

- Rebinding application is incremental, that is, an unbound term can be partially rebound, leading to still open code. For instance, the term $(x \mapsto X, y \mapsto Y | x + y)$ can be combined with the rebinding $(X \mapsto 0, Z \mapsto 1)$, getting $(y \mapsto Y | 0 + y)$. This allows code specialization, similarly to what partial application achieves for positional binding.

* Partly funded by the project MIUR CINA - Compositionality, Interaction, Negotiation, Autonomicity for the future ICT society.
Rebindings are first-class values as well, and can be manipulated by operators such as overriding and renaming.

We define a type system for the calculus which guarantees soundness by distinguishing types for rebindings on the basis of the allowed subtyping. In particular, rebindings have record types, that may be open or closed. The former are subject to both width and depth subtyping, whereas for the latter we have only depth subtyping.

In the rest of this paper, we first provide the formal definition of an untyped version of the calculus (Section 2), followed by some examples showing the expressive power of the calculus. We then define a typed version of the calculus, for which we state a soundness result. In the Conclusion we discuss some future work.

## 2 Untyped calculus

The syntax and reduction rules of the untyped calculus are given in Figure 1, where we leave unspecified constructs of primitive types such as integers, which we will use in the examples. We assume infinite sets of variables $x$ and names $X$. We use various kinds of finite maps: unbinding maps $u$ from variables to names, rebinding maps $r$ from names to terms, renamings $\sigma$ from names to names, and substitutions $s$ from variables to terms. Finite maps are represented as sequences, e.g., $x_1 \mapsto X_1, \ldots, x_m \mapsto X_m$, assuming all $x_i$ are distinct, and we use the following notations: $\text{dom}$ and $\text{rng}$ for the domain and range, respectively, $u_1 \circ u_2$ for map composition, assuming $\text{rng}(u_2) \subseteq \text{dom}(u_1)$, $u_1, u_2$ for the union of two maps with disjoint domains, and $u_1[u_2]$ for the map coinciding with $u_2$ wherever the latter is defined, with $u_1$ elsewhere.

```
t ::= \ldots | x | \lambda x . t | t_1 . t_2 | (u \mid t_1) | (r) | t_1 \triangleright t_2 | \text{term}

u ::= x_1 \mapsto X_1, \ldots, x_m \mapsto X_m \quad \text{unbinding map}

r ::= X_1 \mapsto t_1, \ldots, X_m \mapsto t_m \quad \text{rebinding map}

\sigma ::= X_1 \mapsto Y_1, \ldots, X_m \mapsto Y_m \quad \text{renaming}

v ::= \ldots | \lambda x . t | (u \mid t) | (r) \quad \text{value}

E ::= \ldots | E \mid v | E \mid E \mid v \quad \text{evaluation context}

s ::= x_1 \mapsto t_1, \ldots, x_m \mapsto t_m \quad \text{substitution}
```

### Reduction Rules

- **(Ctx)** $t \longrightarrow t'$
- **(App)** $(\lambda x . t) \; v \longrightarrow t\{x \mapsto v\}$
- **(Run)** $(\emptyset \mid t) \longrightarrow t$
- **(Override)** $\{r_1\} \ll \{r_2\} \longrightarrow \{r_1[r_2]\}$
- **(Rebind)** $(u_1, u_2 \mid t)\triangleright (r) \longrightarrow (u_2 \mid t\{x \mapsto r(u_1(x)) \mid x \in \text{dom}(u_1)\})$
- $\text{rng}(u_2) \cap \text{dom}(r) = \emptyset$
- **(RenameUsn)** $u \mid t \sigma \longrightarrow (\sigma \circ u \mid t)$
- **(RenameReb)** $(r) \sigma \longrightarrow (r \circ \sigma)$

Fig. 1: Untyped calculus: syntax and reduction rules
Besides lambda-abstractions and values of primitive types, there are two new kinds of values in the calculus: \emph{unbound terms} \(\langle u \mid t \rangle\) and \emph{rebindings} \(\langle r \rangle\). Both represent code which is not directly used but, rather, boxed, as the \(\langle \ \rangle\) brackets suggest, and possibly combined in various ways, to produce code to be actually used via the \emph{run} operator.

An unbound term, e.g., \(\langle x \mapsto X \mid x + 1 \rangle\) represents open code, which can be rebound through a nominal interface.\(^3\) A rebinding, e.g., \(\langle X \mapsto 0, Z \mapsto 1 \rangle\), represents code which can be used to complete open code.

Operators for combining code are \emph{rebinding application, run, overriding,} and \emph{rename}. When a rebinding is applied to an unbound term, rule \(\text{(REBIND)}\), the variables associated with names which are provided by the rebinding are replaced by the corresponding terms. For instance,

\[
\langle x \mapsto X, y \mapsto Y \mid x + y \rangle \mapsto (X \mapsto 0, Z \mapsto 1)
\]

reduces to \(\langle y \mapsto Y \mid 0 + y \rangle\). An unbound term with no names to be rebound can become a conventional term by applying the run operator, rule \(\text{(RUN)}\). For instance, \(\langle \emptyset \mid 0 + 1 \rangle\) reduces to \(0 + 1\), which can then be evaluated. The overriding operator allows one to merge two rebinding\(^s\) giving preference to the right one in case of conflict. For instance, \(\langle X \mapsto 1, Y \mapsto 0 \rangle \triangleleft \langle Y \mapsto 1, Z \mapsto 1 \rangle\) reduces to \(\langle X \mapsto 1, Y \mapsto 1, Z \mapsto 1 \rangle\). The renaming operator allows one to change the nominal interface of boxed code, and can be applied both to unbound terms, rule \(\text{(RENAMEUNB)}\), and rebinding\(^s\), rule \(\text{(RENAMEREB)}\). In the former case, it is possible to merge names, e.g., \(\langle x \mapsto X, y \mapsto Y \mid x + y \rangle\) (\(X \mapsto Y, Y \mapsto Y\)) reduces to \(\langle x \mapsto Y, y \mapsto Y \mid x + y \rangle\), in the latter, to duplicate and remove terms, e.g., \(\langle X \mapsto 0, Z \mapsto 1 \rangle \langle X \mapsto X, Y \mapsto X \rangle\) reduces to \(\langle X \mapsto 0, Y \mapsto 0 \rangle\).

Application of a substitution to a term, \(t\{s\}\), and \emph{free variables of a term, \(\text{FV}(t)\)}, are defined in the usual way. Note that a variable occurrence in the domain of an unbinding map behaves like a \(\lambda\)-binder. Hence, the variables in \emph{dom}(\(u\)) are not free in \(\langle u \mid t \rangle\), and not subject to substitution.

\textbf{Expressive power.} In the rest of this section we discuss the role of our calculus as unifying foundation for dynamic scoping, rebinding, and meta-programming features.

As well synthesized in the classical reference \cite{6} and in \cite{10}, in \emph{lexical scoping}, a variable in an expression refers to the innermost lexically enclosing construct declaring that variable, whereas in \emph{dynamic scoping} a variable refers to the latest active binding existing for that variable at execution time. Calculi and languages supporting dynamic scoping typically distinguish in some way variable occurrences to be dynamically bound. For instance, \cite{6} has two distinct sets of variables \(x_a\) and \(x_d\), and \(\alpha\)-renaming is only allowed on the former.\(^4\)

In our calculus, as in the previous version \cite{1}, names play the role of dynamic variables, and dynamic scoping can be encoded by unbinding and re-binding, e.g., in the traditional example\(^5\)

\begin{verbatim}
let x=3 in
let f=lambda y.x+y in
\end{verbatim}

\(^3\) Note that, differently from, e.g., \cite{7}, names are not terms, to keep separate the conventional language, which is here lambda-calculus for simplicity, from the meta-level constructs, whose semantics is in principle independent.

\(^4\) The same happens, e.g., in Common Lisp with special variables.

\(^5\) Interpreting as usual the \texttt{let} construct as syntactic sugar for application.
let \( x = 5 \) in
\[
\text{f1}
\]
dynamic scoping, which leads to result 6 rather than 4, can be encoded as follows:
let \( x = 3 \) in
\[
\begin{align*}
\text{let } f &= \lambda y . <x\to X \mid x+y> \text{ in} \\
\text{let } x &= 5 \text{ in} \\
&!(<f\ 1>\#<X\to x>)
\end{align*}
\]
In the current language, differently from [1], if we have more than one dynamic variable, we are not forced to provide rebindings for both variables, so let \( x = 3 \) in
\[
\begin{align*}
\text{let } f &= \lambda y . <x\to X, z\to Z \mid x+y+z> \text{ in} \\
\text{let } x &= 5 \text{ in} \\
&!(<f\ 1>\#<X\to x>)
\end{align*}
\]
evaluates to \(<z\to Z \mid 5+1+2>\), whereas it would not be correct in [1].

By \textit{rebinding} is usually meant a policy where standard binding is static, but in some cases, for instance when values or computations are marshalled from a running system and moved elsewhere, some of the identifiers need to be dynamically rebound. Assuming to enrich the calculus with primitives for concurrency, we can model exchange of mobile code, which may contain unbound variables to be rebound by the receiver, as outlined above.

\[
\begin{align*}
\text{let } f &= \lambda x . \lambda y . <x\to X, y\to Y \mid f> \text{ in} \\
&!(<f\ 1>\#<X\to x>)
\end{align*}
\]

||
\[
\begin{align*}
\text{let } x &= \ldots \text{ in} \\
\text{receive}(f).send \ (f\#<X\to x>).\text{nil}
\end{align*}
\]

Note that incremental rebinding allows the process on the right-hand-side to receive open code, to provide a new version of the resource \( x \), and to resend still open code.

Finally, unbound terms can be seen as code whose evaluation is delayed, in the sense of \textit{meta-programming}. Indeed, brackets and the run operator correspond to, and are inspired by, the analogous constructs in MetaML [9]. However, there are two main differences: in our approach we can manipulate open code, where free variables are explicitly indicated, and we have no analogous of the \textit{escape} annotation which allows to force evaluation inside boxed code. We leave to further work the investigation of a similar mechanism for our calculus. In the following example:

\[
f = \lambda x_1 . \lambda x_2 . (y_1 \mapsto X, y_2 \mapsto X \mid (x_1\#\{X \mapsto y_1\}) \ x_2\#\{X \mapsto y_2\})
\]
f is a function manipulating open code: it takes two open code fragments, with the same global nominal interface containing the sole name \( X \), and, after rebinding both, it combines them by means of function application; finally, it unbinds the result so that the resulting nominal interface contains again the sole name \( X \). The fact that the unbinding map is not injective means that the free variables of the two combined open code fragments will be finally rebound to the same value (that is, the same value will be shared). For instance, \( (f \ x \mapsto X \mid \lambda y. y + x) \ x \mapsto X \mid x))\#\{X \mapsto 1\} \) reduces to 2.
3 Typed calculus

The syntax of the language is extended, since variables and names are now annotated with types. Types includes function types, unbound types \( \langle \Delta \mid T \rangle \), and rebinding types \( \langle \Delta \rangle^\nu \) (for simplicity we have removed basic types for primitive values such as integers or Boolean). Unbound types \( \langle \Delta \mid T \rangle \) correspond to open code: \( \Delta \) is a map from names to types called name context, and represented by a finite sequence \( X_1; T_1, \ldots, X_m; T_m \), where we assume that all names occurring in the sequence are distinct. The type specifies that the open code needs the rebinding of the names \( X_i \) \( (1 \leq i \leq m) \) to terms of type \( T_i \) \( (1 \leq i \leq m) \) in order to correctly produce a term of type \( T \). Rebinding types \( \langle \Delta \rangle^\nu \) correspond to rebinding maps; the name context \( \Delta = X_1; T_1, \ldots, X_m; T_m \) specifies that the rebinding map associates each name \( X_i \) with a term of type \( T_i \) \( (1 \leq i \leq m) \). If the type is annotated with \( \nu = \ast \), then we say that the type is open (or non-exact), and the rebinding map is allowed to contain more associations than those specified in the name context. The annotation \( \nu = \circ \) is used for closed (or exact) types, to enforce that the domain of the rebinding map exactly coincides with the domain of \( \Delta \).

Renamings, as well as values, evaluation contexts, and substitutions are defined as for the untyped language. The subtyping relation is defined in Figure 3.

Subtyping between unbound types obeys a rule similar to that for function types: the relation is contravariant in the name context, and covariant in the type returned after rebinding. Subtyping between name contexts is defined by the usual rule for record subtyping: both width and depth subtyping are allowed. Width and depth subtyping are also allowed between rebinding types, in case the right-hand-side (rhs for short) type in the relation is open, because a closed type can always be considered as an open type, but not the other way around. This is a consequence of the fact that closed types express more restrictive constraints on rebinding maps: for instance, the rebinding term \( \langle X; T_X \mapsto t_x, Y; T_Y \mapsto t_y \rangle \) has type \( \langle X; T_X, Y; T_Y \rangle^\nu \) for both \( \nu = \ast \) and \( \nu = \circ \), whereas it has type \( \langle X; T_X \rangle^\nu \) only for \( \nu = \ast \); note also that, in this case, the more precise type is \( \langle X; T_X, Y; T_Y \rangle^{\ast \circ} \). When the rhs type in the subtyping relation is a closed rebinding type, then the lhs type must be closed as well, and, therefore, it must define the same set of names; in this case only depth subtyping is allowed.

---

*Fig. 2: Typed calculus: syntax*

<table>
<thead>
<tr>
<th>t ::=</th>
<th>x</th>
<th>( \lambda x : T \cdot t \mid t_1 \cdot t_2 \mid \langle u \mid t \rangle \mid { \tau } \mid t_1 \cdot t_2 \mid t_1 \cdot t_2 \cdot t \mid t \cdot \sigma ) term</th>
</tr>
</thead>
<tbody>
<tr>
<td>u ::=</td>
<td>( x_1; T_1 \mapsto X_1, \ldots, x_m; T_m \mapsto X_m ) unbinding map</td>
<td></td>
</tr>
<tr>
<td>r ::=</td>
<td>( X_1; T_1, \ldots, X_m; T_m \mapsto t_m ) rebinding map</td>
<td></td>
</tr>
<tr>
<td>( \Gamma ) ::=</td>
<td>( x_1; T_1, \ldots, x_m; T_m ) context</td>
<td></td>
</tr>
<tr>
<td>( \Delta ) ::=</td>
<td>( X_1; T_1, \ldots, X_m; T_m ) name context</td>
<td></td>
</tr>
<tr>
<td>T ::=</td>
<td>( T_1 \mapsto T_2 \mid \langle \Delta \mid T \rangle \mid \langle \Delta \rangle^\nu ) type</td>
<td></td>
</tr>
<tr>
<td>( \nu ) ::=</td>
<td>( \circ \mid + ) variance annotation</td>
<td></td>
</tr>
</tbody>
</table>

---

Proceedings of ICTCS 2013
The typing rules are specified in Figure 4. The type system supports subtyping. Overriding $t_1 \prec t_2$ is always well-formed, providing that $t_1$ and $t_2$ are two well-typed expressions having rebinding types; the name context of the type of $t_1$ is deterministically split in two parts $\Delta_1$ and $\Delta'_1$ (the components undefined in $t_2$, and those defined in $t_2$, respectively). The part $\Delta'_1$ is overridden, hence only $\Delta_1$ contributes to the resulting type. Note that $t_2$ is required to have a closed type, otherwise it would not be possible to correctly identify $\Delta_1$ (the names of $t_1$ not defined in $t_2$). For instance, if $t_1 = \langle X:\text{int} \to t_X \rangle$ and $t_2 = \langle X:\text{bool} \to t'_X \rangle$, since $t_2$ has type $(\cdot)^+$, by admitting open types for $t_2$, we would get the wrong type $\langle X:\text{int} \rangle^+$ for $t_1 \prec t_2$. Finally, the annotation of the resulting type is dictated by the annotation of the type of $t_1$.

The term $\langle u \mid t \rangle$ is well-typed if the unbinding map $u$ satisfies the following sanity condition (imposed by the rule (WF-Unb)): for all $1 \leq i, j \leq m$ if $X_i = X_j$, then $T_i = T_j$. If this is the case, then a name context can be correctly extracted from $u$ with the auxiliary function $X_{env}$ (defined at the bottom of Figure 4); the resulting type $T$ after rebinding is obtained by typing $t$ in the environment updated by the variable type assignment extracted by the auxiliary function $x_{env}$ (defined at the bottom of Figure 4).

Rule for rebinding terms $\langle t \rangle$ is straightforward: an exact type can be always deduced.

Rule (T-Run) states that a term of unbound type can be safely run only if its corresponding name context is empty, hence, all variables have been already properly bound in the code.

The typing rule for rebinding $t_1 \triangleright t_2$ is similar to the typing rule for overriding of re-binding terms: to correctly identify the names in $t_1$ that are not bound (denoted by $\Delta_1$), the rule requires an exact type for $t_2$. The bound names of $t_1$ must have the same type of the corresponding names in $t_2$; however, by applying subsumption, it is always possible to bind a name with a term whose type is a subtype of the expected type.

Finally, rules (T-RenameUnb) and (T-RenameReb) deal with renaming. The auxiliary operators $\sigma \circ \Delta$ and $\Delta \circ \sigma$, which are both partial, are defined at the bottom of Figure 4.

The expression $\sigma \circ \Delta$ is well-defined only if $\text{dom}(\Delta) \subseteq \text{dom}(\sigma)$ and the following sanity condition is satisfied: for all $X, Y \in \text{dom}(\Delta)$, if $\sigma(X) = \sigma(Y)$, then $\Delta(X) = \Delta(Y)$; such a condition ensures that the resulting name context $\sigma \circ \Delta$ is still a map from names to types.
The expression \( \sigma \circ \Delta \) is well-defined only if \( \text{rng}(\sigma) \subseteq \text{dom}(\Delta) \); the resulting type of \( t \sigma \) is exact since its domain coincides with the domain of \( \sigma \) which is always exact.

**Soundness.** The type system is safe since types are preserved by reduction, subject reduction property, and closed terms are not stuck, progress property. One of the crucial properties that we have to prove is the substitution lemma, that in presence of subtyping says that if a term is well typed, a free variable \( x \) of type \( T \) can be substituted with a term \( t \) having type \( T' \) for any \( T' \leq T \).

**Remark.** On rebindings we choose the overriding and rename operators, to have a set of primitive operators with which we could express a variety of other operators. For example, in the typed language we can express, \( t_1 + t_2 \), the sum of the rebinding maps \( t_1 \) and \( t_2 \) (which are assumed to have a disjoint domain), by imposing, with the type system, that \( t_1 \) and \( t_2 \) have types \( \langle \Delta_1 \rangle^\circ \) and \( \langle \Delta_2 \rangle^\circ \) with disjoint name domains. Then, the term \( t_1 < t_2 \) could be used for \( t_1 + t_2 \), since it has the same behavior. Expressing \( t_1 < t_2 \) in terms of sum, instead, is problematic. If \( t_1 \) had type \( \langle \Delta_1 \rangle^\circ \) and \( t_2 \) type \( \langle \Delta_2 \rangle^\circ \), then, let \( \sigma \) be the identity on all names in \( \text{dom}(\Delta_1) \setminus \text{dom}(\Delta_2) \), the term \( (t_1 \sigma) + t_2 \) would

---

**Proceedings of ICTCS 2013**
have the same behavior of $t_1 < t_2$. However, if the type of $t_1$ is not closed, as in the case of overriding, the encoding is not correct, since it removes from $t_1$ all the rebindings that are not in $\text{dom}(\Delta_1) \setminus \text{dom}(\Delta_2)$, thus some rebindings of $t_1$ that is not overridden by $t_2$ might be deleted.

4 Conclusion

In this paper we propose a calculus that integrates binding by position and by name, in which rebinding maps are first class values. The calculus is equipped with a flexible type system in which rebindings have record types, that may be closed, in which case the rebinding must rebind exactly the names mentioned in the type, or open, in which case it rebinds at least those names. Rebindings, in addition to be applied to unbound terms, can be combined with the overriding operator, or their interface (the name to be rebound) can be modified via a rename operator. We have shown how to express dynamic variables, rebinding, and some meta-programming features, and also how the operators can be used to encode other operators present in calculi for modules, see [3]. This work continues a stream of research on foundations of binding mechanisms started by the seminal papers [5, 4]. In future work we plan to explore how to get the expressive power of escape annotations as in MetaML [9], and to add some code manipulation operators to enhance the meta-programming capabilities of the calculus.

References

Circular String Matching Revisited

Carl Barton\textsuperscript{1}, Costas S. Iliopoulos\textsuperscript{1,2}, and Solon P. Pissis\textsuperscript{3,4}

\textsuperscript{1} Department of Informatics, King’s College London, UK
\{carl.barton, costas.iliopoulos\}@kcl.ac.uk
\textsuperscript{2} Department of Mathematics & Statistics
University of Western Australia, Crawley WA, Australia
\textsuperscript{3} Heidelberg Institute for Theoretical Studies, Germany
solon.pissis@h-its.org
\textsuperscript{4} Florida Museum of Natural History
University of Florida, FL, USA

Abstract. Circular string matching is a problem which naturally arises in many biological contexts. It consists in finding all occurrences of the rotations of a pattern of length $m$ in a text of length $n$. There exist optimal average-case algorithms for exact circular string matching. In this article, we present a suboptimal average-case algorithm for exact circular string matching requiring time and space $O(n \log n)$. However, we anticipate that this can be easily adapted to deal with approximate circular string matching with $k$-mismatches, under the Hamming distance model, with no additional cost in time or space complexity for moderate values of $k$.

1 Introduction

In order to provide an overview of our results and algorithms, we begin with a few definitions, generally following [11]. We think of a \textit{string} $x$ of \textbf{length} $n$ as an array $x[0..n-1]$, where every $x[i]$, $0 \leq i < n$, is a \textit{letter} drawn from some fixed \textit{alphabet} $\Sigma$ of size $|\Sigma|$. The \textbf{empty string} of length 0 is denoted by $\varepsilon$. A string $x$ is a \textit{factor} of a string $y$ if there exist two strings $u$ and $v$, such that $y = uv$. Let the strings $x, y, u, v, x_0$, such that $y = uv$. If $u = \varepsilon$, then $x$ is a \textit{prefix} of $y$. If $v = \varepsilon$, then $x$ is a \textit{suffix} of $y$.

Let $x$ be a non-empty string of length $n$ and $y$ be a string. We say that there exists an \textit{occurrence} of $x$ in $y$, or, more simply, that $x$ \textit{occurs in} $y$, when $x$ is a factor of $y$. Every occurrence of $x$ can be characterised by a position in $y$. Thus we say that $x$ occurs at the \textit{starting position} $i$ in $y$ when $y[i..i+n-1] = x$. The \textbf{Hamming distance} between strings $x$ and $y$, both of length $n$, is the number of positions $i$, $0 \leq i < n$, such that $x[i] \neq y[i]$. Given a nonnegative integer $k$, we write $x \equiv_k y$ if the Hamming distance between $x$ and $y$ is at most $k$.

A circular string of length $n$ can be viewed as a traditional linear string which has the left- and right-most symbols wrapped around and stuck together in some way. Under this notion, the same circular string can be seen as $n$ different linear strings, which would all be considered equivalent. Given a string $x$ of length $n$, we denote by $x^i = x[i..i+n-1]x[0..i-1]$, $0 \leq i < n$, the $i$-th \textit{rotation} of $x$ and $x^0 = x$. Consider, for instance, the string $x = ababbbc$; this string has the following rotations: $x^1 = babbbca$, $x^2 = ababbcb$, $x^3 = babcaba$, $x^4 = abbcabab$, $x^5 = bbababa$, $x^6 = bcababab$, $x^7 = cabaabb$.

Proceedings of ICTCS 2013
Here we consider the problem of finding occurrences of a pattern $x$ of length $m$ with circular structure in a text $t$ of length $n$ with linear structure. For instance, the DNA sequence of many viruses has circular structure, so if one wishes to find occurrences of a virus in a DNA sequence—which may not be circular—they must locate all positions in $t$ that at least one rotation of $x$ occurs. This is the problem of **circular string matching**.

The problem of exact circular string matching has been considered in $[n]$, where an $O(n)$-time algorithm was presented. A naive solution with quadratic complexity consists in applying a classical algorithm for searching a finite set of strings after having built the trie of rotations of $x$. The approach presented in $[8]$ consists in preprocessing $x$ by constructing a suffix automaton of the string $xx$, by noting that every rotation of $x$ is a factor of $xx$. Then, by feeding $t$ into the automaton, the lengths of the longest factors of $xx$ occurring in $t$ can be found by the links followed in the automaton in time $O(n)$. In $[5]$, the authors presented an optimal average-case algorithm for exact circular string matching, by also showing that the average-case lower bound for single string matching of $O(n \log \sigma m/m)$ also holds for circular string matching. Very recently, in $[1]$, the authors presented two fast average-case algorithms based on word-level parallelism. The first algorithm requires average-case time $O(n \log \sigma m/w)$, where $w$ is the number of bits in the computer word. The second one is based on a mixture of word-level parallelism and $q$-grams. The authors showed that with the addition of $q$-grams, and by setting $q = O(\log \sigma m)$, an optimal average-case time of $O(n \log \sigma m/m)$ is achieved.

The aforementioned algorithms for the exact case exhibit the following disadvantages: first, they cannot be applied in a biological context since both, single nucleotide polymorphisms, as well as errors introduced by wet-lab sequencing platforms might have occurred in the sequences; second, it is not clear whether they could easily be adapted to deal with the approximate case. In this article, we revisit the following problem.

**Problem 1 (Exact Circular String Matching).** Given a pattern $x$ of length $m$ and a text $t$ of length $n > m$, find all factors $u$ of $t$ such that $u = x^i$, $0 \leq i < m$.

We present a new suboptimal average-case algorithm for exact circular string matching requiring time and space $O(n)$. We anticipate that this can be easily adapted to deal with approximate circular string matching with $k$-mismatches, under the Hamming distance model, with no additional cost in time or space complexity for moderate values of $k$.

## 2 Properties of the Partitioning Technique

In this section, we give a brief outline of the **partitioning** technique, introduced in $[12]$, and in some sense earlier in $[10]$, for approximate string matching; and then show some properties of the version of the technique we use for our algorithm.

The idea behind the partitioning technique is to partition the given pattern in such a way that at least one of the fragments must occur exactly in any
valid approximate occurrence of the pattern. It is then possible to search for these fragments exactly to give a set of candidate occurrences of the pattern. It is then left to the verification portion of the algorithm to check if these are valid approximate occurrences of the pattern. It has been experimentally shown that this approach yields very good practical performance on large-scale datasets [6], even if it is not theoretically optimal.

Here we make use of the partitioning technique for exact circular string matching. By choosing an appropriate number of fragments, we ensure that at least one fragment must occur in any valid exact occurrence of a rotation. Lemma 2 together with the following fact provide this number.

**Fact 1** Any rotation of \( x = x[0..m-1] \) is a factor of \( x' = x[0..m-1]x[0..m-2] \).

**Lemma 2** If we partition \( x' = x[0..m-1]x[0..m-2] \) in 4 fragments of length \( \lceil (2m-1)/4 \rceil \) and \( \lceil (2m-1)/4 \rceil \), at least one of the 4 fragments is a factor of any factor of length \( m \) of \( x' \).

**Proof.** Let \( \ell_f \) denote the length of the fragment. By Fact 1, if any factor of length \( m \) of \( x' \) must contain at least one of the fragments, then it holds that

\[
m > 2\ell_f.
\]

We get the following inequality

\[
2m > 4\ell_f,
\]

which gives

\[
2m - 1 \geq 4\ell_f.
\]

Therefore

\[
\ell_f \leq (2m - 1)/4.
\]

Since the length of \( x' \) is \( 2m - 1 \), we have to partition it in at least 4 fragments of length \( \lceil (2m - 1)/4 \rceil \) and \( \lceil (2m - 1)/4 \rceil \).

3 Exact Circular String Matching via Filtering

In this section, we present ECSMF, a new suboptimal average-case algorithm for exact circular string matching via filtering. It is based on the partitioning technique and a series of practical and well-established data structures.

3.1 Longest Common Extension

First we describe how to compute the longest common extension, denoted by \( \text{lce} \), of two suffixes of a string in constant time. lce queries are an important part of the algorithms presented later on.

Let \( \text{SA} \) denote the array of positions of the sorted suffixes of string \( x \) of length \( n \), i.e. for all \( 1 \leq r < n \), we have \( x[\text{SA}[r-1]..n-1] < x[\text{SA}[r]..n-1] \). The inverse \( \text{iSA} \) of the array \( \text{SA} \) is defined by \( \text{iSA}[\text{SA}[r]] = r \), for all \( 0 \leq r < n \).
Let $\text{lcp}(r, s)$ denote the length of the longest common prefix of the strings $x[SA[r] \ldots n - 1]$ and $x[SA[s] \ldots n - 1]$, for all $0 \leq r, s < n$, and $0$ otherwise. Let $\text{LCP}$ denote the array defined by $\text{LCP}[r] = \text{lcp}(r - 1, r)$, for all $1 < r < n$, and $\text{LCP}[0] = 0$. We perform the following linear-time and linear-space preprocessing:

- Compute arrays $\text{SA}$ and $\text{iSA}$ of $x$ [9].
- Compute array $\text{LCP}$ of $x$ [3].
- Preprocess array $\text{LCP}$ for range minimum queries, we denote this by $\text{RMQ}_{\text{LCP}}$ [4].

With the preprocessing complete, the $\text{lce}$ of two suffixes of $x$ starting at positions $p$ and $q$ can be computed in constant time in the following way [7]:

$$\text{LCE}(x, p, q) = \text{LCP}[\text{RMQ}_{\text{LCP}}(\text{iSA}[p] + 1, \text{iSA}[q])].$$

### 3.2 Algorithm

Given a pattern $x$ of length $m$ and a text $t$ of length $n > m$, an outline of algorithm ECSMF for solving Problem 1 is as follows.

1. Construct the string $x' = x[0 \ldots m - 1]|x[0 \ldots m - 2]$ of length $2m - 1$. By Fact 1, any rotation of $x$ is a factor of $x'$.
2. The pattern $x'$ is partitioned in 4 fragments of length $\lfloor (2m - 1)/4 \rfloor$ and $\lceil (2m - 1)/4 \rceil$. By Lemma 2, at least one of the 4 fragments is a factor of any rotation of $x$.
3. Match the 4 fragments against the text $t$ using an Aho Corasick automaton [2]. Let $\mathcal{L}$ be a list of tuples of size $O(nr)$, where $< p_{x'}, \ell, p_t > \in \mathcal{L}$ is a 3-tuple such that $0 \leq p_{x'} < 2m - 1$ is the position that the fragment occurs in $x'$, $\ell$ is the length of the respective fragment, and $0 \leq p_t < n$ is the position that the fragment occurs in $t$.
4. Compute $\text{SA}$, $\text{iSA}$, $\text{LCP}$, and $\text{RMQ}_{\text{LCP}}$ of $T = x't$. Compute $\text{SA}$, $\text{iSA}$, $\text{LCP}$, and $\text{RMQ}_{\text{LCP}}$ of $T_r = \text{rev}(tx')$, that is the reverse string of $tx'$.
5. For each tuple $< p_{x'}, \ell, p_t > \in \mathcal{L}$, we try to extend to the right via computing

$$\mathcal{E}_r \leftarrow \text{LCE}(T, p_{x'} + \ell, 2m - 1 + p_t + \ell);$$

in other words, we compute the length $\mathcal{E}_r$ of the longest common prefix of $x'[p_{x'} + \ell \ldots 2m - 1]$ and $t[p_t + \ell \ldots n - 1]$, both being suffixes of $T$. Similarly, we try to extend to the left via computing $\mathcal{E}_l$ using $\text{lce}$ queries on the suffixes of $T_r$.
6. For each $\mathcal{E}_l, \mathcal{E}_r$ computed for tuple $< p_{x'}, \ell, p_t > \in \mathcal{L}$, we report all the valid starting positions in $t$ by first checking if the total length $\mathcal{E}_l + \ell + \mathcal{E}_r \geq m$; that is the length of the full extension of the fragment is greater than or equal to $m$, matching at least one rotation of $x$. If that is the case, then we report positions

$$\max\{p_t - \mathcal{E}_l, p_t + \ell - m\}, \ldots, \min\{p_t + \ell - m + \mathcal{E}_r, p_t\}.$$
Example 1. Let the pattern \( x = \text{GGGTCTA} \) of length \( m = 7 \), and the text \( t = \text{GATACGATACCTAGGGTGATAGAATAG} \). Then \( x' = \text{GGGTCTAGGGTCT} \) (Step 1). \( x' \) is partitioned in \( \text{GGGT} \) and \( \text{CTA} \), of length \( \ell = 3 \), occurs at starting position \( p_\ell = 10 \) in \( t \) (Step 3). Then \( T = \text{GGGTCTAGGGTCTGATACGATACCTAGGGTGATAGAATAG} \) and \( T_r = \text{TCTGGGATCTGGGGATAGATGCTGGATCCATAGCTAG} \) (Step 4). Extending to the left gives \( \mathcal{E}_\ell = 0 \), since \( T_r[9] \neq T_r[30] \), and extending to the right gives \( \mathcal{E}_r = 4 \), since \( T[7 \ldots 10] = T[26 \ldots 29] \) and \( T[11] \neq T[30] \) (Step 5). We check that \( \mathcal{E}_\ell + \ell + \mathcal{E}_r = 7 = m \), and therefore we report position 10 (Step 6):

\[
p_\ell - \mathcal{E}_\ell = 10 - 0 = 10, \ldots, p_\ell + \ell - m + \mathcal{E}_r = 10 + 3 - 7 + 4 = 10;
\]

that is, \( x^4 = \text{CTAGGGT} \) occurs at starting position 10 in \( t \).

Theorem 3. Given a pattern \( x \) of length \( m \) drawn from alphabet \( \Sigma, \sigma = |\Sigma| \), and a text \( t \) of length \( n > m \) drawn from \( \Sigma \), algorithm ECSMF requires average-case time \( \mathcal{O}(n) \) to solve Problem 1.

Proof. Constructing and partitioning the string \( x' \) from \( x \) can trivially be done in time \( \mathcal{O}(m) \) (Step 1-2). Building the Aho-Corasick automaton of the 4 fragments requires time \( \mathcal{O}(m) \); and the search time is \( \mathcal{O}(n + \text{Occ}) \) (Step 3) [2]. The preprocessing step for the lce queries on the suffixes of \( T \) and \( T_r \) can be done in time \( \mathcal{O}(n) \) (Step 4)—see Section 3.1. Computing \( \mathcal{E}_\ell \) and \( \mathcal{E}_r \) for each occurrence of a fragment requires time \( \mathcal{O}(\text{Occ}) \) (Step 5)—see Section 3.1. For each extended occurrence of a fragment, we may report up to \( \mathcal{O}(m) \) valid starting positions, thus \( \mathcal{O}(m\text{Occ}) \) in total (Step 6). Since the expected number \( \text{Occ} \) of occurrences of the 4 fragments in \( t \) is \( 4n/\sigma^{2m-1}/4 = \mathcal{O}(\frac{n}{\sigma^{2m-1}}) \), algorithm ECSMF requires average-case time \( \mathcal{O}(1 + \frac{m}{\sigma^{2m-1}})n \). It achieves average-case time \( \mathcal{O}(n) \) iff

\[
f = \frac{4m}{\sigma^{2m-1}} n \leq cn
\]

for some fixed constant \( c \). For \( \sigma = 2 \), the maximum value of \( f \) is attained at

\[
m = \frac{2}{\ln 2} \approx 2.8853
\]

and so for \( \sigma > 1 \) we get

\[
\frac{4m}{\sigma^{2m-1}} n \leq 5.05n.
\]

4 Final Remarks

In this preliminary work, we presented a new average-case algorithm for exact circular string matching requiring average-case time and space \( \mathcal{O}(n) \). Although the algorithm presented is suboptimal, we anticipate that it can be easily adapted to deal with approximate circular string matching with \( k \)-mismatches, under the Hamming distance model, whilst retaining its runtime efficiency.
The presented algorithm performs two LCE queries (left and right extensions) during the verification of a matched fragment. To extend this approach in order to accommodate \( k \) mismatches, we have to first choose the appropriate number of fragments, and then perform \( O(k) \) LCE queries instead with a small additional overhead to report valid matches. Hence, although suboptimal, our approach is flexible and allows for a simple extension to accommodate \( k \) mismatches. This is of interest since approximate circular string matching is not a well-researched problem, with, to the best of our knowledge, no optimal algorithm for this case. For future work, we will consider performing average-case analysis for the extension to Hamming distance model to determine its runtime efficiency.

References

Deciding in practice the Satisfiability of Continuous-time Metric Temporal Logic

Marcello M. Bersani¹, Matteo Rossi¹ and Pierluigi San Pietro¹,²

¹ Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano
² CNR IEIT-MI, Milano, Italy
{marcellomaria.bersani,matteo.rossi,pierluigi.sanpietro}@polimi.it

Abstract. Constraint LTL-over-clocks is a variant of CLTL, an extension of linear-time temporal logic allowing atomic assertions in a concrete constraint system. Satisfiability of CLTL-over-clocks is here shown to be decidable by means of a reduction to a decidable SMT (Satisfiability Modulo Theories) problem. The result is a complete Bounded Satisfiability Checking procedure, which has been implemented by using standard SMT solvers. The importance of this technique derives from the possibility of translating various continuous-time metric temporal logics, such as MITL and QTL, into CLTL-over-clocks itself. Although standard decision procedures of these logics do exist, they are actually very difficult to implement and they have never been realized in practice. Suitable translations into CLTL-over-clocks have instead allowed us the development of the first prototype tool for deciding MITL and QTL.

1 Introduction

Constraint-LTL [9], called CLTL, is an extension of linear-time temporal logic allowing atomic assertions in a concrete constraint system. By carefully choosing the constraint system, CLTL may be decidable, as well as expressive and well-suited to define infinite-state systems and their properties.

In this paper, we define a variant of CLTL, called CLTL-over-clocks (CLTL-oc), where arithmetic variables occurring in atomic assertions are evaluated as clocks. A clock “measures” the time elapsed since the last time the clock itself was “reset” (i.e., the variable was equal to 0); clocks can also be tested against a constant. By definition, in CLTL-oc each (discrete) instant i is associated with a “time delay” corresponding to the “time elapsed” between i and the next instant i + 1. This allows mixing of discrete events with continuous-time, a typical situation arising in many computer-controlled applications.

Satisfiability of CLTL-oc is here shown to be decidable by reducing it to a decidable SMT (Satisfiability Modulo Theories) problem, resulting in a complete Bounded Satisfiability Checking procedure. Although other automata-based decision procedures would also be suitable (e.g., [9]), the novelty of our reduction is that it can easily be implemented (by using standard SMT solvers). In fact, we implemented also a tool, publicly available, to verify CLTL-oc formulae.

Although CLTL-oc may be used to specify and verify timed systems, a further advantage of our approach is that it is possible to translate various continuous-time metric temporal logics, such as MITL (Metric Interval Temporal Logic) [4] and QTL [10], into CLTL-oc itself. Standard decision procedures of MITL and QTL logics do already exist (e.g., [4,11,15]), typically based

Proceedings of ICTCS 2013
on Timed Automata [3], but they are actually very difficult to implement and, to the best of our knowledge, they have never been realized in practice.

In general, the level of support for verification of continuous-time temporal logics is not as well developed as for discrete-time models. Uppaal [6] is the de-facto standard tool for verification of Timed Automata, but it does not support continuous-time temporal logics: not only satisfiability checking is not available in Uppaal, but even the formalization of system properties in temporal logic is not allowed, aside from rather simple invariants and reachability properties. Satisfiability Modulo Theories is a promising but well-consolidated theory, supported by efficient solvers that are able to decide problems of many disciplines. In particular, decidable SMT problems have been already considered in the recent past, for instance to solve reachability [12] and the bounded version of language inclusion [5] for Timed Automata. Nonetheless, also this approach has so far failed to produce a concrete decision procedure for logics such as MITL and QTL. This difficulty is caused by the gap of translating formulae into Timed Automata, a step which is avoided by our approach.

Suitable translations into CLTL-oc have instead allowed us the development of the first available tool for deciding both MITL and QTL, hence showing the generality and effectiveness of our approach. Further evidence of this is provided by the fact that we have also been able to provide a translation of the extension of QTL with so-called Pnueli and counting modalities [14]), thus providing its first concrete decision procedure.

The paper is organized as follows: Sect. 2 defines CLTL-oc, illustrating its usage by means of a running example (a timed lamp). Sect. 3 outlines a SMT-based decision procedure. Sect. 4 illustrates the tool, showing various results of verification. Sect. 5 draws a few conclusions.

2 Constraint LTL (over clocks)

Constraint LTL (CLTL [9,8]) is the decidable logic that we use to solve the satisfiability problem of various metric temporal logics over continuous time. CLTL formulae are defined with respect to a finite set $V$ of variables and a constraint system $D$, which is a pair $(D, R)$ with $D$ being a specific domain of interpretation for variables and constants and $R$ being a family of relations on $D$, such that the set $AP$ of atomic propositions coincides with set $R_0$ of 0-ary relations. An atomic constraint is a term of the form $R(x_1, \ldots, x_n)$, where $R$ is an $n$-ary relation of $R$ on domain $D$ and $x_1, \ldots, x_n$ are variables. A valuation is a mapping $v : V \rightarrow D$, i.e., an assignment of a value in $D$ to each variable. A constraint is satisfied by $v$, written $v \models_D R(x_1, \ldots, x_n)$, if $(v(x_1), \ldots, v(x_n)) \in R$. Given a variable $x \in V$ over domain $D$, temporal terms are defined by the syntax: $\alpha := c \mid x \mid X \alpha \mid Y \alpha \mid U \alpha \mid S \alpha$, where $c$ is a constant in $D$ and $x$ denotes a variable over $D$. Operator $X$ is very similar to next operator of LTL, but it only applies to temporal terms, with the meaning that $X\alpha$ is the value of temporal term $\alpha$ in the next time instant. Well-formed CLTL formulae are defined as follows:

$$\phi := R(\alpha_1, \ldots, \alpha_n) \mid \phi \land \phi \mid \neg \phi \mid X(\phi) \mid Y(\phi) \mid U\phi \mid S\phi$$
where \( \alpha_i \)'s are temporal terms, \( R \in \mathcal{R} \), \( X \), \( Y \), \( U \) and \( S \) are the usual “next”, “previous”, “until” and “since” operators of LTL, with the same meaning. The dual operators “release” \( \text{R} \), and “trigger” \( \text{T} \) may be defined as usual, i.e., \( \phi \text{R} \psi \) is \( \neg(\neg \phi \text{U} \psi) \) and \( \phi \text{T} \psi \) is \( \neg(\neg \phi \text{S} \psi) \). The set \( AP \) of atomic proposition is the set of 0-ary relation, which is denoted by \( \mathcal{R}_0 \).

The semantics of CLTL formulae is defined with respect to a strict linear order representing time \((\mathbb{N}, \prec)\). Truth values of propositions in \( AP \), and values of variables belonging to \( V \) are defined by a pair \((\pi, \sigma)\) where \( \sigma : \mathbb{N} \times V \rightarrow D \) is a function which defines the value of variables at each position in \( \mathbb{N} \) and \( \pi : \mathbb{N} \rightarrow \phi(\text{AP}) \) is a function associating a subset of the set of propositions with each element of \( \mathbb{N} \). The value of terms is defined with respect to \( \sigma \) as \( \sigma(i, \alpha) = \sigma(i + |\alpha|, x_\alpha) \), where \( x_\alpha \) is the variable in \( V \) occurring in term \( \alpha \) and \( |\alpha| \) is the depth of a temporal term, i.e., the total amount of temporal shift needed in evaluating \( \alpha \): \( |x| = 0 \) when \( x \) is a variable, and \( |\text{X} \alpha| = |\alpha| + 1 \). The semantics of a formula \( \phi \) at instant \( i \geq 0 \) over a linear structure \((\pi, \sigma)\) is recursively defined as in Table 1, where \( x_\alpha \) is the variable that appears in temporal term \( \alpha_i \), and \( R \in \mathcal{R} \setminus \mathcal{R}_0 \). A formula \( \phi \) is satisfiable if there exists

\[
(\pi, \sigma), i \models p \iff p \in \pi(i) \text{ for } p \in \text{AP}
\]

\[
(\pi, \sigma), i \models R(\alpha_1, \ldots, \alpha_n) \iff (\sigma(i + |\alpha_1|, x_{\alpha_1}), \ldots, \sigma(i + |\alpha_n|, x_{\alpha_n})) \in R
\]

\[
(\pi, \sigma), i \models \neg \phi \iff (\pi, \sigma), i \not\models \phi
\]

\[
(\pi, \sigma), i \models \phi \land \psi \iff (\pi, \sigma), i \models \phi \text{ and } (\pi, \sigma), i \models \psi
\]

\[
(\pi, \sigma), i \models X(\phi) \iff (\pi, \sigma), i + 1 \models \phi
\]

\[
(\pi, \sigma), i \models \text{Y}(\phi) \iff (\pi, \sigma), i - 1 \models \phi \land i > 0
\]

\[
(\pi, \sigma), i \models \phi \text{U} \psi \iff \exists j \geq i : (\pi, \sigma), j \models \psi \land
\]

\[
(\pi, \sigma), n \models \phi \forall i \leq n < j
\]

\[
(\pi, \sigma), i \models \phi \text{S} \psi \iff \exists 0 \leq j \leq i : (\pi, \sigma), j \models \psi \land
\]

\[
(\pi, \sigma), n \models \phi \forall j < n \leq i
\]

Table 1. Semantics of CLTL.

\( (\pi, \sigma) \) such that \( (\pi, \sigma), 0 \models \phi \).

In this paper, we consider a variant of CLTL, where arithmetic variables are evaluated as clocks and set \( \mathcal{R} \setminus \mathcal{R}_0 \) is \( \{\prec, =\} \). A clock “measures” the time elapsed since the last time the clock was “reset” (i.e., the variable was equal to 0). By definition, in CLTL-oc each \( i \in \mathbb{N} \) is associated with a “time delay” \( \delta(i) \), where \( \delta(i) > 0 \) for all \( i \), which corresponds to the “time elapsed” between \( i \) and the next state \( i + 1 \). More precisely, for all clocks \( x \in V \), \( \sigma(i + 1, x) = \sigma(i, x) + \delta(i) \), unless it is “reset” (i.e., \( \sigma(i + 1, x) = 0 \)). CLTL-oc can be shown to be decidable. Before going further, to motivate our approach, we provide an example of a CLTL-oc formula representing a simple yet realistic timed system.

Example 1. We consider the LTL specification of a timed lamp and its properties (studied in Sect. 4) from [13]. The lamp is controlled by two buttons, labeled ON and OFF resp., which cannot be pressed simultaneously. The lamp itself can be either on or off. When ON is pressed the lamp is immediately turned on, regardless of its current state, while if OFF is pushed then the
lamp is immediately turned off, also regardless of its current state. After ON is pressed, the lamp will not stay on forever, but, if no more buttons are pressed, it will turn off with a delay $\Delta$, a positive real constant. By pressing the ON button before the timeout expiration then the timeout is extended by a new delay $\Delta$.

Our CLTL-oc formula makes use of atomic propositions $on$, $off$ and $l$ representing, respectively, events “push button ON” and “push button OFF” and the state “light is on”. Clocks may be used to measure the exact time elapsed since the last on; clearly some clock must be “reset” (i.e., set to 0, in analogy to Timed Automata) whenever ON is pressed, while when a clock is equal to $\Delta$ then the timeout expires. Since the introduction of clocks is not straightforward, we first define a few shorthands called $reset-c$, $test_{c=\Delta}$ and $test_{0\leq c\leq \Delta}$.

They have the intuitive meaning (which will be formalized after the main specification) that they are true if, and only if, a clock $c$ is reset or, respectively, $c = \Delta$, or $0 < c \leq \Delta$. The specification of the lamp, still lacking the precise clock specification, is defined by the formula $G \left( \bigwedge_{i=0}^{5}(i) \right)$, where $G (\phi)$ is the usual globally operator defined by $\Box R\phi$, of the following formulae:

\[
\neg (on \land off) \tag{1} \\
\text{on} \Leftrightarrow reset-c \\
Y(l) \Rightarrow test_{0\leq c\leq \Delta} \tag{3} \\
\text{turnoff} \Leftrightarrow Y(l) \land (off \lor test_{c=\Delta}) \tag{4} \\
l \Leftrightarrow \neg \text{turnoff} S on. \tag{5}
\]

Formula (1) ensures mutual exclusion; (2) states that the timeout must be (re)started whenever button ON is pressed; (3) constrains the time elapsed since the previous instant if the light was on at that moment (i.e., not more than $\Delta$); (4) defines (for readability) an event $\text{turnoff}$, capturing the two cases when the lamp (supposed to be ON in the previous instant) must be turned off at the current instant (i.e., OFF being pressed or the timeout expiring); finally, (5) gives the specification of the light, as being on if, and only if, there was in the past an on event not followed by a $\text{turnoff}$. Initialization is implicit in the specification (at instant 0, the light is off).

To complete the specification, we must formalize also the clock behavior. In CLTL-oc, “resetting a clock” $c$, e.g., following an on event is as simple as stating that $on \Rightarrow c = 0$; testing a clock $c$ against a constant $\Delta$ and causing say, a $\text{turnoff}$ is a simple as stating that $c = \Delta \Rightarrow \text{turnoff}$. Unfortunately, the same clock cannot be tested and reset at the same time. A possible solution is to introduce two clocks $c_0$ and $c_1$, rather than just one clock, so that they are reset alternatively: only one of the two clocks is reset and a new reset of the same clock will eventually occur only after the occurrence of a reset of the other clock. The behavior of this clock pair is described by the axiom $G \left( (6) \land (7) \right)$, where formulae (6) and (7) are:

\[
\bigwedge_{i\in\{0,1\}} (c_i = 0 \Rightarrow \neg X ((c_{(i+1)\mod 2} > 0 U c_1 = 0))) \tag{6} \\
c_0 = 0 \Rightarrow \neg (c_1 = 0) \tag{7}
\]
and \( \lfloor \cdot \rfloor_2 \) stands for the modulo 2 operator (i.e., \( (1)_2 = 0 \), \( (2)_2 = 0 \)). Finally, the above clock shorthands \( \text{reset-c} \), \( \text{test}_{c=\Delta} \) and \( \text{test}_{0 < c \leq \Delta} \) are defined as follows:

\[
\text{reset-c} \iff c_0 = 0 \lor c_1 = 0
\]

\[
\text{test}_{0 < c \leq \Delta} \iff \bigvee_{i \in \{0, 1\}} 0 < c_i \leq \Delta
\]

\[
\text{test}_{c=\Delta} \iff \bigvee_{i \in \{0, 1\}} (c_i = \Delta \land (c_{(i+1)} > \Delta \lor c_{(i+1)_2} = 0))
\]

### 3 Solving CLTL-over-clocks satisfiability

We outline how to decide the satisfiability for CLTL over clocks by a SMT-based technique. The technique is based on previous work [7], in which we used a \( k \)-bounded satisfiability problem to solve the satisfiability of CLTL formulae by using a polynomial reduction to a SMT problem. \( k \)-bounded satisfiability is complete, and deciding the satisfiability of a CLTL formula can be done by means of a finite amount of \( k \)-bounded satisfiability tests, for increasing values of \( k \). To deal with variables that behave like clocks, the method developed in [7] is extended to represent time progress. A CLTL formula \( \phi \) is called \( k \)-bounded satisfiable if there exists an ultimately periodic sequence of symbolic valuations \( sv_0, sv_{l-1}(sv_l \ldots sv_{k-1})^\omega \), which is a symbolic model of \( \phi \) and such that there is a partial assignment of values to all the variables occurring in \( \phi \) only for a finite number of positions in time, from 0 to \( k + 1 \). In other words, in \( k \)-bounded satisfiability we look for a finite sequence of symbolic valuations \( sv_0, sv_{l-1}(sv_l \ldots sv_{k-1}sv_k) \), where \( sv_k = sv_l \), which admits a \( k \)-bounded arithmetic model and that is representative of an infinite symbolic model for \( \phi \) of the form \( sv_0, sv_{l-1}(sv_l \ldots sv_{k-1})^\omega \). While in PSPACE, in practice \( k \)-bounded satisfiability can be quite efficient, at least when the value of \( k \) is small enough to perform the check: checking \( k \)-bounded satisfiability is then equivalent to solve a few SMT problems in P. Obviously, the upper bound for \( k \) is in general exponential in the size of the formula.

In [7] we show how to solve \( k \)-bounded satisfiability for CLTL formulae over a class of arithmetical constraints that include the family of clock constraints used in Sect. 2. The \( k \)-bounded satisfiability problem is solved through a polynomial-time reduction to the satisfiability problem of a formula in the theory of Equality and Uninterpreted Functions combined with Linear Integers/Reals Arithmetic. The combination of the two theories is decidable and its decision procedure is implemented by many SMT-solvers. The reduction of [7] has been implemented in the \texttt{ase2zot} plugin of the ZoT tool [2]. Therefore, an instance of the \( k \)-bounded satisfiability problem for CLTL formulae has the complexity of the underlying SMT problem, which depends on the arithmetic theory required. In our case, since clocks are in \( \mathbb{R} \), we solve SMT problems in QF-EUF \( \cup \) LRA, whose complexity is P. A peculiarity of the SMT-based approach is that, if the set of symbolic valuations partitions of the space \( D^{|V|} \) (with \( D \) the domain of the variables in \( V \)), then a sequence of valuations uniquely induces a sequence of symbolic valuations. By solving the \( k \)-bounded satisfiability problem for a formula \( \phi \) we obtain, from the model of the QF-EUF formula, the sequence of valuations satisfying the constraints occurring...
in the formula, which induces a symbolic model. Hence, unlike automata-based techniques, our approach does not require the explicit construction of the set \( SV(\phi) \). We exploit this to avoid building the set of clock regions induced by CLTL formulæ.

To solve the satisfiability of CLTL over clocks we still use \( k \)-bounded satisfiability to look for ultimately periodic models, but we extend the method in order to represent clock regions and time progression. Representing clock regions is quite straightforward and exploits the fact that regions partition the space of all possible clock valuations. In other words, a clock valuation identifies a clock region, so it is not necessary to precompute the set of all clock regions from the formula. The only requirement to be enforced is the periodicity of the sequence of clock regions corresponding to clock valuations. If one is looking for a model of length \( k \), the sequence of clock regions is of the form: \( R_0 \ldots R_{k-1} (R_l \ldots R_{k-1})^n \), which is obtained from a finite sequence \( R_0 \ldots R_{l-1} (R_l \ldots R_{k-1} R_k) \) with the periodicity constraint \( R_l = R_k \).

The QF-EUF encoding of CLTL formulæ is defined to enforce periodicity of all atomic formulæ (atomic propositions and clock constraints) between positions \( k \) and \( l \). For instance, given two clocks \( x, y \), if \( x = y \) holds at position \( k \) (i.e., \( x(k) = y(k) \)) then, by the periodicity constraints, it must also hold at position \( l \): \( x(k) = y(k) \iff x(l) = y(l) \). To obtain a periodic sequence of regions we provide the solver with all the clock constraints which may occur in the definition of regions in \( R_\phi \) (but not all the regions). Given a CLTL-oc formula \( \phi \), we define the set \( ac(R_\phi) \) as the set of all clock constraints induced by \( \phi \). Let \( \phi' \) be a CLTL formula, \( C_{\phi'} \) be the set of clocks appearing in \( \phi' \), and \( x, y \in C_{\phi'} \). If \( c(x) \) is the maximum constant with which clock \( x \) is compared in \( \phi' \) and \( \sim \in \{<,=,>\} \), the set \( ac(x) = \{ x \sim c \mid \forall c \in [0,1,\ldots,c(x)] \} \{ x < 0 \} \) is the set of all clock constraints between \( x \) and constant \( c(x) \), while the set \( ac(x, y) = \{ x \sim y \} \) is the set of all clock constraints between \( x \) and \( y \). Then, set \( ac(R_\phi) \) is defined as: \( \bigcup_{x,y \in C_{\phi'}, x \neq y} ac(x) \cup ac(x, y) \). We indicate with \( Per(ac(R_\phi)) \) the set of QF-EUF periodicity constraints on arithmetic constraints \( ac(R) \), that is, \( \forall \theta \in ac(R) \theta(k) \iff \theta(l) \). Time elapsing is obtained by forcing all clocks in \( \phi' \) to progress by the same amount of time between two positions of the model. Let \( i \in [0, k-1] \); then, constraint \( x(i + 1) = x(i) + \delta(i) \lor x(i + 1) = 0 \), where \( \delta(i) > 0 \), is imposed to represent a positive elapsing time, for all clocks \( x \in C_{\phi'} \). We indicate with \( Adv(C_{\phi'}) \) the set of constraints that impose the uniform time advancement of clocks in \( C_{\phi'} \). Since all variable in the theory of Reals are defined over \( \mathbb{R} \) and all \( x \in C_{\phi'} \) behave like clocks, we force all clocks to be nonnegative \( x(i) \geq 0 \) for all \( x \in C_{\phi'} \), and for all \( i \in [0, k] \). We indicate the conjunction of these constraints as \( nNeg(C_{\phi'}) \).

We solve the satisfiability problem of CLTL-oc formula \( \phi \) by feeding the SMT solver the set of constraints \( |\phi'|_k \cup Per(ac(R_\phi)) \cup Adv(C_{\phi'}) \cup nNeg(C_{\phi'}) \), where \( |\phi'|_k \) is the bounded representation of \( \phi' \) described in [7].
4 Implementation & Experimental Results

The decision procedure of Sect. 3 for CLTL-oc is implemented in a plugin, called ae2zot, of our Zot toolkit [2], whereas reductions from MITL and QTL into CLTL-oc are implemented in the qtlsolver tool, available from [1].

We carried out some verification experiments on the example of the Timed Lamp described in Sect. 2. We have built several descriptions of the lamp: (i) the CLTL-oc model presented in Sect. 2; (ii) a MITL specification on signals; (iii) a QTL specification in which predicates on and off are constrained to be true only in isolated instants. On each of these specifications we have carried out three experiments, assuming $\Delta = 5$: a check of the satisfiability of the specification, to show that it is consistent ($sat$); the (dis)proof of property “the light never stays on for more than $\Delta$ time units” ($p_1$); the proof of property “if at some point the light stays on for more than $\Delta$ time units, then there is an instant when on is pressed, and then it is pressed again before $\Delta$ time units” ($p_2$).

Table 2 reports the time and space required for the checks outlined above (all tests have been done using the Common Lisp compiler SBCL 1.1.2 on a 2.13GHz Core2 Duo MacBook Air with MacOS X 10.7 and 4GB of RAM; the solver was z3 4.0). All bounded satisfiability checks have been performed using a bound $k = 20$. The first line of each row shows the total processing time (i.e., parsing and solving) and the time taken by the SMT-solver (both times in seconds). The second line reports the heap size (in Mbytes) required by Z3.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Satisfiable?</th>
<th>CLTL-oc</th>
<th>MITL (l.e.r.o)</th>
<th>QTL (unrest.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sat</td>
<td>Yes</td>
<td>0.48/0.33</td>
<td>15.5/13.84</td>
<td>4.24/3.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.63</td>
<td>66.45</td>
<td>27.12</td>
</tr>
<tr>
<td>p1</td>
<td>Yes</td>
<td>0.52/0.35</td>
<td>36.74/33.16</td>
<td>17.2/14.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.22</td>
<td>102.47</td>
<td>63.5</td>
</tr>
<tr>
<td>p2</td>
<td>No</td>
<td>0.67/0.49</td>
<td>6.61/5.09</td>
<td>257.1/240.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.55</td>
<td>110.27</td>
<td>58.66</td>
</tr>
</tbody>
</table>

5 Conclusions

This paper investigates a bounded approach to satisfiability checking of an extension of CLTL where variables behave like clocks (CLTL-over-clocks). The logic has been encoded into a decidable SMT problem, allowing the implementation in our ae2zot tool. Hence, both in principle and in practice, SMT solvers can be used to check the satisfiability of CLTL-over-clocks. Actually, rather than a specification language itself, CLTL-over-clocks can be considered as a target language to reduce decision problems of various continuous-time formalisms (MITL and QTL, but in principle also Timed Automata or Timed Petri Nets). To the best of our knowledge, our approach is the first allowing an
effective implementation of a fully automated verification tool for continuous-time metric temporal logics such as MITL and QTL. The tool is still a non-optimized prototype, whose performance might also be substantially improved in future, optimized versions. Still, verification of formulae with many clocks may always be infeasible, since satisfiability of MITL is EXPSPACE-complete (but an interesting, PSPACE-complete fragment of MITL is also supported). However, in practice a large number of clocks is not very frequent, and many examples of CLTL-oc, MITL (and QTL) formulae have been verified in a fairly short time.

References

1. qtlsolver. available from qtlsolver.googlecode.com.
Computing the Longest Common Abelian Factor

Ali Alatabbi$^1$, Costas S. Iliopoulos$^1$, Alessio Langiu$^1$, and M. Sohel Rahman$^2$

$^1$ King’s College London, London, UK
{Ali.Alatabbi,C.Iliopoulos,Alessio.Langiu}@kcl.ac.uk
$^2$ Department of CSE, BUET, Dhaka-1000, Bangladesh
msraham@cse.buet.ac.bd

Abstract. In this paper we consider the problem of finding the length of a longest common abelian factor (LCAF) between two binary strings and finding a common abelian factor of length LCAF. We present a quadratic running time algorithm and a sub-quadratic running time solution both having linear space requirement for binary strings. Furthermore, we present a simple extension of the quadratic algorithm to the fixed size alphabet case which has running time $O(n^2 \times \log n \times \sigma)$, where $\sigma$ is the alphabet size and $n$ is the longest of the two considered strings.

1 Introduction

The Longest Common Substring (LCS) problem, in the exact matching case, it is an historical problem. As Dan Gusfield [5, Sec. 7.4] reported in his book “in 1970 Don Knuth conjectured a linear time algorithm for this problem would be impossible”. Decades later, a linear time solution for the LCS problem was obtained by using the linear construction of the suffix tree. An analogous problem in the abelian case is the Longest Common Abelian Factor (LCAF) problem, where, given two strings, one wants computing the length of a longest common abelian factor and finding one of such substrings.

Abelian properties concerning words have been studied since the very beginning of Formal Languages and Combinatorics on Words. Abelian powers were first considered in 1961 by Erdős [4] as a natural generalization of usual powers. In 1966, Parikh [7] defined a vector having length equal to the alphabet cardinality, which reports the number of occurrences of each alphabet symbol inside a given string. Later on, the scientific community started referring to such a vector as a Parikh vector. Clearly, two strings having the same Parikh vector are permutations one of the other and there is an abelian match between them.

The trivial solution of the LCAF problem for constant alphabet has $O(n^3)$ time complexity, where $n$ is the length of the longest string. We present a $O(n^2)$ time algorithm and a $O(n^2 / \log n)$ solution for binary strings, both requiring linear space, and a $O(n^2 \times \log n \times \sigma)$ solution for alphabet of size $\sigma$.

This paper is organized as follows. In Sec. 2 we state some preliminary definitions. In Sec. 3 we present a quadratic solution for binary strings. In Sec. 4 we present a simple solution for strings over a finite size alphabet and in Sec. 5 we present a sub-quadratic solution for the binary alphabet.
2 Definitions

An alphabet $\Sigma$ of size $\sigma > 0$ is a finite set whose elements are called letters. A string on an alphabet $\Sigma$ is a finite, possibly empty, sequence of elements of $\Sigma$. The zero-letter sequence is called the empty string, and is denoted by $\varepsilon$. The length of a string $S$ is defined as the length of the sequence associated with the string $S$, and is denoted by $|S|$. We denote by $S[i]$ the $i$-th letter of $S$, for all $1 \leq i \leq |S|$ and $S = S[1..|S|]$. A string $w$ is a factor of a string $S$ if there exist two strings $u$ and $v$, such that $S = uvw$.

Given a string $S$ over the alphabet $\Sigma = \{a_1, \ldots, a_\sigma\}$, we denote by $|S|_{a_j}$ the number of $a_j$'s in $S$, for $1 \leq j \leq \sigma$. We define the Parikh vector of $S$ as $P_S = (|S|_{a_1}, \ldots, |S|_{a_\sigma})$.

In the binary case, we denote $\Sigma = \{0, 1\}$, the number of 0's in $S$ by $|S|_0$, the number of 1's in $S$ by $|S|_1$ and the Parikh vector of $S$ as $P_S = (|S|_0, |S|_1)$.

In what follows we focus on binary strings. The general alphabet case will be considered later on in this paper.

For a given binary string $S$ of length $n$, we define a $n \times n$ matrix $M_S$ as follows.

Each row of $M_S$ is dedicated to a particular length of factors of $S$. So, Row $\ell$ of $M_S$ is dedicated to $\ell$-length factors of $S$. Each column of $M_S$ is dedicated to a particular starting position of factors (of different lengths) of $S$. So, Column $i$ of $M_S$ is dedicated to the position $i$ of $S$. So, $M_S[\ell][i]$ is dedicated to the $\ell$-length factor that starts at position $i$ of $S$ and it reports the number of 1's of such factor. Now, $M_S[\ell][i] = m$ if and only if the $\ell$-length factor that starts at position $i$ of $S$ has a total of $m$ 1's, that is, $|S[i \ldots i + \ell - 1]|_1 = m$.

**Definition 1.** Given a binary string $S$ of length $n$, we define a $n \times n$ matrix $M_S$ such that $M_S[\ell][i] = |S[i \ldots i + \ell - 1]|_1$, for $1 \leq \ell \leq n$ and $1 \leq i \leq (n-\ell+1)$, and $M_S[\ell][i] = 0$, otherwise.

In what follows, we will use $M_S[\ell]$ to refer to Row $\ell$ of $M_S$.

Assume that we are given two strings $A$ and $B$ on an alphabet $\Sigma$. For the sake of ease, we assume that $|A| = |B| = n$. We want to find the length of a longest common abelian factor between $A$ and $B$.

**Definition 2.** Given two strings $A$ and $B$ over the alphabet $\Sigma$, we say that $w$ is a common abelian factor for $A$ and $B$ if there exist a factor (or substring) $u$ in $A$ and a factor $v$ in $B$ such that $P_w = P_u = P_v$.

**Definition 3.** The Longest Common Abelian Factor (LCAF) between $A$ and $B$ is the length of a longest factor $u$ of $A$ and a factor $v$ of $B$ such that $P_u = P_v$.

Suppose the matrices $M_A$ and $M_B$ for the binary strings $A$ and $B$ have been computed. Now we have the following easy lemma.

**Lemma 1.** There is a common abelian factor of length $\ell$ between $A$ and $B$ if and only if there exists $p, q$ such that $1 \leq p, q \leq n - \ell + 1$ and $M_A[\ell][p] = M_B[\ell][q]$.
Let us first focus on the computation of the length of the highest common abelian factor of length \( \ell \) between \( A \) and \( B \). The other way is also obvious by definition.

Clearly, if we have \( M_A \) and \( M_B \) we can compute the LCAF by identifying the highest \( \ell \) such that there exists \( p \), \( q \) having \( 1 \leq p, q \leq n - \ell + 1 \) and \( M_A[\ell][p] = M_B[\ell][q] \). Then we can say that the LCAF between \( A \) and \( B \) is the length \( \ell \) and common abelian factors of length \( \ell \) are \( A[p \ldots p + \ell - 1] \) and \( B[q \ldots q + \ell - 1] \).

3 A Quadratic Algorithm

Let us first focus on the computation of \( M_A \). Computation of \( M_B \) is identical. We first consider the computation of a row of \( M_A \). Suppose that we are computing \( M_A[\ell] \).

1. For \( i = 1 \) to \( n - \ell + 1 \) do the following
2. \( \text{compute } M_A[\ell][i] = |A[i \ldots i + \ell - 1]|_1 \)

The above computation runs in linear time because we can compute \( |A[i + 1 \ldots i + 1 + \ell - 1]|_1 \) from \( |A[i \ldots i + \ell - 1]|_1 \) in constant time by simply checking whether \( A[i] = 1 \) and \( A[i + \ell] = 1 \). So, computation for each row requires \( O(n) \) and hence the total matrix computation runs in \( O(n^3) \).

Now once \( M_A \) and \( M_B \) are computed, we simply need to apply the idea of Lemma 1. The idea is to check for all values of \( \ell \) whether there exists a pair \( p, q \) such that \( 1 \leq p, q \leq n - \ell + 1 \) and \( M_A[\ell][p] = M_B[\ell][q] \). Then return the highest possible value of \( \ell \) and corresponding values of \( p, q \). A straightforward implementation would require \( O(n^3) \). But we can do better.

Since we only want the length of the factor, then all we need to do is compute \( M_A[\ell] \cap M_B[\ell] \) for all values of \( \ell \). Clearly, if \( M_A[\ell] \cap M_B[\ell] = \emptyset \) then there does not exist any common abelian factor of length \( \ell \). So, we find the highest \( \ell \) such that \( M_A[\ell] \cap M_B[\ell] \neq \emptyset \) and return \( \ell \) as the length of a longest common abelian factor. Since we need to do at most \( n \) intersections, on sets having cardinality \( n \), the total time remains \( O(n^2) \).

However, we may need to identify the actual factor. We can do this with some book-keeping during the computation of \( M_A \) and \( M_B \). We will compute \( M_A[\ell] \) and \( M_B[\ell] \) in tandem, i.e., simultaneously. We will maintain some auxiliary arrays \( R_A, ind_A \) and \( R_B, ind_B \) each of length \( n \) for book-keeping. We also maintain another array of length \( n \times 2 \) called \textit{record} to keep track of the factors. Note that \( R_A, ind_A \) and \( R_B, ind_B \) would be local to the computation of a particular row; but \textit{record} is a global array. We initialize these arrays to all zero.

Suppose we have computed \( M_A[\ell][i] = k_1 \) and \( M_B[\ell][i] = k_2 \). We will then increment \( R_A[k_1] \) and \( R_B[k_2] \). Whenever a particular index of \( R_A \) or \( R_B \) is incremented, we check whether that index for both \( R_A \) and \( R_B \) is non-zero. If both \( R_A, R_B \) are nonzero for a particular index \( k \), it clearly means that both strings have a common factor of length \( \ell \) having \( k \) 1’s.
In particular we do the following:

1. Compute $M_A[\ell][i] = k_1$
2. Set $R_A[k_1] = R_A[k_1] + 1$
3. Set $\text{ind}_A[k_1] = i$
4. if $(R_A[k_1] \neq 0)$ and $(R_B[k_1] \neq 0)$ then $\text{record}[\ell] = (\text{ind}_A[k_1], \text{ind}_B[k_1])$
5. Compute $M_B[\ell][i] = k_2$
6. Set $R_B[k_2] = R_B[k_2] + 1$
7. Set $\text{ind}_B[k_2] = i$
8. if $(R_A[k_2] \neq 0)$ and $(R_B[k_2] \neq 0)$ then $\text{record}[\ell] = (\text{ind}_A[k_2], \text{ind}_B[k_2])$

In this way we keep track of common abelian factors in $\text{record}[1..\ell]$. Clearly, this can be done in $O(n^2)$ in total. Notably, $\text{record}$ will retain the record of the last occurrence of a factor of length LCAF in case of multiple occurrences.

Notice that since we compute each row independently from the others, then we can maintain just one row (for each of the two matrices) at the time by using a linear space or, better, since we do not actually need to store the $M_A[\ell][i]$ ($M_B[\ell][i]$) values at all, the total space needed is just a linear space for auxiliary arrays.

### 4 Fixed size alphabet

We generalize the definition of the matrix $M_S$ for strings over a fixed size alphabet $\Sigma = \{a_1, \ldots, a_\sigma\}$ by defining a $n \times n$ matrix $M_S$ of $(\sigma - 1)$-length vectors. $M_S[\ell][i] = V_{\ell,i}$, where $V_{\ell,i}[j] = |S[i \ldots i + \ell - 1]|_{a_j}$, for $1 \leq \ell \leq n$, $1 \leq i \leq (n - \ell + 1)$ and $1 \leq j < \sigma$, and $V_{\ell,i}[j] = 0$, otherwise. We will refer to the $j$-th element of the array $V_{\ell,i}$ of the matrix $M_S$ by using the notation $M_S[\ell][i][j]$. Notice that the last component of a Parikh vector is determined by using the length of the string and all the other components of the Parikh vector.

Now, $M_S[\ell][i][j] = m$ if and only if the $\ell$-length factor that starts at position $i$ of $S$ has a total of $m$ $a_j$’s, that is $|S[i \ldots i + \ell - 1]|_{a_j} = m$.

Suppose that we are computing $M_S[\ell]$.

1. For $i = 1$ to $n - \ell + 1$ do the following
2. compute $M_S[\ell][i][j] = |S[i \ldots i + \ell - 1]|_{a_j}$ for $1 \leq j < \sigma$.

Similarly to the binary case, the above computation runs in linear time because we can compute $|S[i + 1 \ldots i + 1 + \ell - 1]|_{a_j}$ from $|S[i \ldots i + \ell - 1]|_{a_j}$ in constant time by simply checking whether $S[i] = a_j$ and $S[i + \ell] = a_j$ for $1 \leq j < \sigma$. Assuming that we will use the $V_{\ell,i}$ vectors sequentially, we will maintain just one of such vectors for any row $\ell$. Starting from $V_{\ell,i}$ for $i = 1$, we need to update at most two element in the vector $V_{\ell,i}$ each time in order to obtain the new (virtual) vector $V_{\ell,i+1}$. Therefore, when the $\ell$-window slides from $i$ to $i + 1$, we increment the value of $V_{\ell,i}[S[i + \ell]]$ and decrement $V_{\ell,i}[S[i]]$. So, computation for each row requires $O(n)$ and hence the total matrix computation runs in $O(n^2)$. 

Proceedings of ICTCS 2013
The adapted algorithm finds the highest $\ell$ such that $M_A[\ell] \cap M_B[\ell] \neq \emptyset$ and return $\ell$ as the length of the longest common abelian factor. Since we need to do at most $n$ intersections on sets of $(\sigma - 1)$-size vectors having cardinality $n$, the total time is $O(n^2 \times (\sigma - 1))$.

In order to compute the row wise intersections between the matrices $M_A$ and $M_B$ we build an Aho Corasick automaton [3] $AC_\ell$ of all the unique vectors $V_{\ell,x}$, for any $1 \leq x \leq n$, of the matrix $M_A$. We treat a $V_{\ell,x}$ vector as a sequence of $\sigma$ values over the set $[1..n]$. For a given $\ell$, the computation of $AC_\ell$ takes $O(n \log n)$ time and space.

Then, any vector $V_{\ell,y}$ of $M_B$ is searched within $AC_\ell$ and an abelian match is reported whenever a vector $V_{\ell,y}$ of $M_B$ is already present in $AC_\ell$. This step is repeated for any $1 \leq \ell \leq n$, possibly using decreasing values starting from $n$. The total time is $O((\sigma - 1) n^2 \log n)$. More details and some variants about the general alphabet case will be presented in the extended version of this paper.

We can report the positions on $A$ and $B$ of an LCAF by augmenting the $AC_\ell$ automaton with a field in the automaton states where a value $x$ is stored. If a node $v$ in $AC_\ell$ contains the value $x$, then $V_{\ell,x}$ can be read on $AC_\ell$ from the root (or initial node) to the node $v$.

## 5 A Sub-quadratic Algorithm

In Section 3, we have presented a $O(n^2)$ algorithm to compute the LCAF between two binary strings and two occurrences of common abelian factors, one in each string, of length LCAF. In this section, we show how we can achieve a better running time for the LCAF problem. We will make use of the recent data structure of Moosa and Rahman [6] for indexing an abelian pattern. The results of Moosa and Rahman [6] is presented in the form of following lemmas with appropriate rephrasing to facilitate our description.

**Lemma 2.** (Interpolation lemma). If $S_1$ and $S_2$ are two substrings of a string $S$ on a binary alphabet such that $\ell = |S_1| = |S_2|$, $i = |S_1|$, $j = |S_2|$, $j > i + 1$, then, there exists another substring $S_3$ such that $\ell = |S_3|$ and $i < |S_3| < j$.

**Lemma 3.** Suppose we are given a string $S$ of length $n$ on a binary alphabet. Suppose that maxOne$(S, \ell)$ and minOne$(S, \ell)$ denote, respectively, the maximum and minimum number of 1’s in any substring of $S$ having length $\ell$. Then, for all $1 \leq \ell \leq n$, maxOne$(S, \ell)$ and minOne$(S, \ell)$ can be computed in $O(n^2 / \log n)$ time and linear space.

A result similar to Lemma 2 is contained in the paper of Cicales et al. [2, Lemma 4], while the result of Lemma 3 has been discovered simultaneously and independently by Moosa and Rahman [6] and by Burcsi et al. [1]. In addition to the above results we further use the following lemma.

**Lemma 4.** Suppose we are given two binary strings $A, B$ of length $n$ each. There is a common abelian factor of $A$ and $B$ having length $\ell$ if and only if $\text{maxOne}(B, \ell) \geq \text{minOne}(A, \ell)$ and $\text{maxOne}(A, \ell) \geq \text{minOne}(B, \ell)$.  

*Proceedings of ICTCS 2013*
Lemma Assume that $\min_A = \minOne(A, \ell)$, $\max_A = \maxOne(A, \ell)$, $\min_B = \minOne(B, \ell)$, $\max_B = \maxOne(B, \ell)$. Now by Lemma 2, for all $\min_A \leq k_A \leq \max_A$, we have some $\ell$-length substrings $A(k_A)$ of $A$ such that $|A(k_A)| = k_A$. Similarly, for all $\min_B \leq k_B \leq \max_B$, we have some $\ell$-length factors $B(k)$ of $B$ such that $|B(k_B)| = k_B$. Now, consider the range $[\min_A \ldots \max_A]$ and $[\min_B \ldots \max_B]$. Clearly, these two ranges overlap if and only if $\max_B \not< \min_A$ and $\max_A \not< \min_B$. If these two ranges overlap then there exists some $k$ such that $\min_A \leq k \leq \max_A$ and $\min_B \leq k \leq \max_B$. Then we must have some substring $\ell$-length factors $A(k)$ and $B(k)$. Hence the result follows.

Let us now focus on devising an algorithm for computing the LCAF given two binary strings $A$ and $B$ of length $n$. For all $1 \leq \ell \leq n$, we compute $\maxOne(A, \ell)$, $\minOne(A, \ell)$, $\maxOne(B, \ell)$ and $\minOne(B, \ell)$ in $O(n^2/\log n)$ time (Lemma 3). Now we try to check the necessary and sufficient condition of Lemma 4 for all $1 \leq \ell \leq n$ starting from $n$ down to 1. We compute the highest $\ell$ such that $[\minOne(A, \ell) \ldots \maxOne(A, \ell)]$ and $[\minOne(B, \ell) \ldots \maxOne(B, \ell)]$ overlap.

Suppose that $\mathcal{K}$ is the set of values that is contained in the above overlap, that is $\mathcal{K} = \{ k \mid k \in [\minOne(A, \ell) \ldots \maxOne(A, \ell)] \text{ and } k \in [\minOne(B, \ell) \ldots \maxOne(B, \ell)] \}$. Then by Lemma 4, we must have a set $\mathcal{S}$ of common abelian factors of $A, B$ such that for all $S \in \mathcal{S}$, $|S| = \ell$. Since we identify the highest $\ell$, the length of a longest common factor must be $\ell$, i.e., LCAF.$\ell$. Additionally, we have further identified the number of 1’s in such longest factors in the form of the set $\mathcal{K}$. Also, note that for a $k \in \mathcal{K}$ we must have a factor $S \in \mathcal{S}$ such that $|S| = k$.

Now let us focus on identifying an occurrence of the LCAF. There are a number of ways to do that. But a straightforward and conceptually easy way is to run the folklore $\ell$-window based algorithm in [6] on the strings $A$ and $B$ to find the $\ell$-length factor with number of 1’s equal to a particular value $k \in \mathcal{K}$.

The overall running time of the algorithm is deduced as follows. By Lemma 3, the computation of $\maxOne(A, \ell)$, $\minOne(A, \ell)$, $\maxOne(B, \ell)$ and $\minOne(B, \ell)$ can be done in $O(n^2/\log n)$ time and linear space. The checking of the condition of Lemma 4 can be done in constant time for a particular value of $\ell$. Therefore, in total, it can be done in $O(n)$ time. Finally, the folklore algorithm requires $O(n)$ time to identify an occurrence (or all of them) of the factors. In total the running time is $O(n^2/\log n)$ and linear space.

References


