Proceedings of the 13th Italian Conference on
Theoretical Computer Science

ICTCS 2012

Villa Toeplitz
Varese, Italy
September 19-21, 2012

Proceedings of ICTCS 2012, 13th Italian Conference on Theoretical Computer Science

Varese, Italy
September 19-21, 2012

Foreword

The 13th Italian Conference on Theoretical Computer Science (ITCTS 2012) was held at University of Insubria in Varese, Italy. It was a three-day conference starting September 19th and ending September 21st, 2012. The conference of the Italian Chapter of the European Association for Theoretical Computer Science, besides being a forum of exchange of ideas, provides the ideal environment where junior researchers and PhD students meet senior researchers.

The papers presented at ICTCS 2012 were, not only from many italian universities, but also from 8 countries including Finland, France, Germany, Moldovia, Netherlands, Portugal, Romania, and Serbia. There were three invited talks given at the conference. They were given by (in alphabetic order) Paolo Ferragina (Pisa), Ugo Montanari (Pisa), and Antonio Restivo (Palermo).

This volume includes all the 36 contributed papers, and the papers or abstracts from the 3 invited speakers. We warmly thank all the invited speakers and all the authors of the submitted papers. Their efforts were the basis of the success of the conference. We would like to thank all the members of the Program Committee and the external referees. Their work in evaluating the papers and their comments during the discussions were essential. Special thanks to Paola Spoletini and to Violetta Lonati for their invaluable help in typesetting the Proceedings and in designing the ICTCS 2012 web site.

Paolo Massazza

Organization

Conference Chair:

Paolo Massazza, *University of Insubria, Varese*

Program Committee:

- Alberto Bertoni, *University of Milano, Milano*
- Tiziana Calamoneri, *University of Roma "La Sapienza", Roma*
- Mario Coppo, *University of Torino, Torino*
- Clelia De Felice, *University of Salerno, Salerno*
- Pierpaolo Degano, *University of Pisa, Pisa*
- Violetta Lonati, *University of Milano, Milano*
- Emanuela Marelli, *University of Camerino, Camerino*
- Paola Spoletini, *University of Insubria, Varese*

Conference Program

On hierarchical graphs: reconciling bigraphs, gs-monoidal theories and gs-graphs[⋆]

Roberto Bruni[1], Ugo Montanari[1], Gordon Plotkin[2], and Daniele Terreni[1]

[1] Computer Science Department, University of Pisa, Italy
[2] LFCS, School of Informatics, University of Edinburgh, UK

Abstract. Compositional graph models for global computing systems must account for two relevant dimensions, namely *nesting* and *linking*. In Milner's *bigraphs* the two dimensions are made explicit and represented as loosely coupled structures: the *place graph* and the *link graph*. Here, bigraphs are compared with an earlier model, gs-graphs, based on gs-monoidal theories and originally conceived for modelling the syntactical structure of agents with α-convertible declarations. We show that gs-graphs are quite convenient also for the new purpose, since the two dimensions can be recovered by introducing two types of nodes. With respect to bigraphs, gs-graphs can be proved essentially equivalent, with minor differences at the interface level. We argue that gs-graphs have a simpler and more standard algebraic structure for representing both states and transitions, and can be equipped with a simple type system (in the style of relational separation logic) to check the well-formedness of *bounded* gs-graphs. Another advantage concerns a textual form in terms of sets of assignments, which can make implementation easier in rewriting frameworks like Maude. Vice versa, the reactive system approach developed for bigraphs needs yet to be addressed in gs-graphs.

1 Introduction

When modelling distributed systems, it is necessary to represent states and their evolutions. Usually, states are seen as terms of an algebra equipped with certain structural axioms, and state transitions are defined via conditional term rewriting rules in the SOS format. A different alternative is to represent states as graphs and transitions as graph transformations. To have the best of the two approaches, it is sometimes possible to characterise the terms up to structural axioms as graphs and the transitions derivable via the SOS inference rules as graph transformations. A good example is provided by arrows of gs-monoidal theories, which can be seen as gs-graphs, and transitions, represented as 2/double cells of a 2/double category, which can be seen as gs-graph transformations. The approach has been applied to π-calculus [9], to CCS with localities [9] and causality [2], to logic programming [3] and to other models of computation.

[⋆] Research supported by the EU Integrated Project 257414 ASCENS and by the Italian MIUR Project IPODS (PRIN 2008).

Gs-monoidal theories [6, 2] are suitable symmetric strict monoidal categories [14] that resemble cartesian (Lawvere) theories, but without the two naturality axioms that allow copying shared subterms and garbage collecting unused terms. In addition, gs-monoidal theories are built out of symbols taken from a hyper-signature instead of an ordinary signature. The difference is that symbols in a hyper-signature are not constrained to have single-sorted codomain, but, like domains, can be a tuple of sorts. The corresponding gs-graphs are a kind of dags where substructures can be shared due to both ordinary duplicators, as in term graphs, and hyper-signature symbols (see e.g. Fig. 2(a)). A useful feature of gs-graphs is that they can be represented in textual form as sets of assignments of the form $x, y := f(z, v)$, where f is a signature symbol that takes two arguments and returns a pair of results and x, y, z, v are α-convertible names. Here 2-cells are essentially like term rewriting rules which can be both contextualised and instantiated. Double cells allow one to model synchronisation of local rewritings.

Recent developments in the area of open-ended systems for global computing have emphasised the need for *hierarchical* graph models: they have two relevant dimensions, namely *nesting* and *linking*. The former has to do with the structural design of processes (e.g., the scoping of a transaction, a compensation, or a session, or the containment of an ambient, a membrane, or an environment); it induces a tree-like hierarchy on nodes. The latter concerns interaction capabilities (e.g., for communication, handshaking, or connectivity) that are flat, and may connect any tree nodes. This is the *pure* case. The situation is more complex in the *binding* case, where a name used for communication is declared at some level in the tree hierarchy and is usable only below its declaration point. Inspired by Cardelli and Gordon's *ambients* [4] and aiming at defining a general, flexible and easy to grasp model, Milner defined *bigraphs*, where the two dimensions are made explicit and represented as essentially orthogonal structures, called *place graph* and *link graph*. Bigraphs [19, 18, 13, 16] have been studied in depth from several points of view, in particular as a basis of the *reactive system* approach, where a labelled transition system, over which bisimilarity is a congruence, is synthesised from a reduction semantics. This part of the theory is not covered for gs-graphs.

Gs-graphs were originally conceived with the syntactical structure of agents with α-convertible declarations in mind, not the hierarchical structure of global computing systems. However it turns out that they are also quite convenient for this new purpose. The *nesting* and *linking* structure can be recovered by defining two types of nodes, *black* nodes, which represent intermediate places in the tree-like hierarchy, and *white* nodes which are communication names/channels.

In this paper, gs-graphs are compared with support equivalent bigraphs. Our first correspondence result states that the two models essentially coincide in the pure case. The only difference is in the interface: gs-graphs have an ordered tuple of connectors and all the names are α-convertible, while bigraph connectors are decorated with names. The "names versus strings in a free monoid" dichotomy can already be found in the simple case of equational theories versus Lawvere theories, where one has to translate between variables and numbers. On the one hand, named connectors facilitate the direct representation in bigraphs of standard process calculi operations; on the other hand, they make the algebraic structure of bigraphs more complex and less standard: e.g. parallel composition is partial and sequential composition is associative only up to isomorphism. Also the rewriting structure does not generate composable cells. On the contrary, gs-graphs inherit from gs-monoidal theories a variety of well-behaved operations.

Our second correspondence result is concerned with (support equivalent) binding bigraphs. In the binding case, it is necessary to constrain the compositional structure to generate only *legal* graphs. For bigraphs [8], additional information is inserted in the interface to allow for a complete axiomatisation. In our approach, we introduce a type system which recognises legal binding gs-graphs. On gs-monoidal theories the type system is represented by membership sentences in membership equational logic [15], while on gs-graphs we exploit a quite simple relational type system (in the style of *relational separation logic* [21]). When the gs-graph is pure, no pairs are generated; parallel composition does not add any pair; and for sequential composition existing pairs are preserved, but new pairs, possibly leading to inconsistency, are generated. The complexity of the proposed typing algorithm is $O(B \cdot W)$, where B is the number of black nodes and W is the number of bound white nodes.

The formal assessment of the analogies and differences between the two different proposals and the definition of transformations to move from one framework to the other allows us to conclude that: (i) bigraphs can be presented at a suitable level of abstraction as arrows of a particular free symmetric monoidal theories, in a perfectly standard way; and (ii) the gs-graphs representation seems to offer some advantages over the others.

Structure of the paper. Section 2 recaps the basics of the models we are comparing. Section 3 addresses the case of pure signatures, while the binding case is discussed in Section 4. Finally, Section 5 contains come concluding remarks.

Additional material is concerned with the formal definition of sequential and parallel compositions of gs-graphs (Appendix A) and of bigraphs (Appendix B), their preservation via the transformations presented in the paper (Appendix C) and the technical details of the transformation from binding bigraphs to gs-graphs (Appendix D).

2 Background on graph-based structures

Notation. For an ordinal n, we write $i \in n$ as a shorthand for $i \in \{0, \ldots, n-1\}$ and let $[n, m]$ denote the set $\{i \mid n \leq i \leq m\}$. We use the symbol $\uplus$ for disjoint union of sets. We let S^* denote the free monoid over the elements in S, whose product is juxtaposition and whose unit is denoted by ϵ. We abbreviate the juxtaposition of n consecutive objects u by u^n, with $u^0 = \epsilon$. We overload $|\,\boldsymbol{.}\,|$ to denote the length of a string, the cardinality of a set and the *support* of place / link / bigraphs (see Definitions 6–8).

$$\text{(ops)}\ \frac{f \in \Sigma_{u,v}}{f : u \to v} \qquad \text{(ids)}\ \frac{u \in S^*}{id_u : u \to u} \qquad \text{(bang)}\ \frac{u \in S^*}{!_u : u \to \epsilon} \qquad \text{(dup)}\ \frac{u \in S^*}{\nabla_u : u \to uu}$$

$$\text{(sym)}\ \frac{u, v \in S^*}{\rho_{u,v} : uv \to vu} \qquad \text{(seq)}\ \frac{t : u \to v \quad t' : v \to w}{t; t' : u \to w} \qquad \text{(par)}\ \frac{t : u \to v \quad t' : u' \to v'}{t \otimes t' : uu' \to vv'}$$

Fig. 1. Inference rules of gs-monoidal theories

2.1 From signatures to gs-graphs

The gs-monoidal approach is based on representing basic computational entities and resources as hyperedges and interaction capabilities by the way in which their tentacles are connected to nodes. (The name *gs* comes after *graph structure*.) For example, nodes can model communication channels and tentacles can express the capability to perform i/o operations on them.

The idea is to consider a particular class of graphs obtained by selecting a few basic shapes for hyper-edges (i.e. fix a hyper-signature) and by freely composing them in series and in parallel to build larger and more complex shapes. Moreover, it is allowed: 1) to rearrange the wirings of tentacles to connect in series edges that otherwise are not "adjacent"; 2) to mark nodes as private to a certain subgraph so that no other tentacle can be attached to them; 3) to attach more than two tentacles to the same node. As only acyclic structures are allowed, hyperedges can be stratified along the implicit partial order defined by tentacle connections.

Definition 1 (hyper-signature). *Given a set S of sorts, a* hyper-signature *(*signature*, for short) Σ is a family $\{\Sigma_{u,v}\}_{u,v \in S^*}$ of sets of operators such that each $f \in \Sigma_{u,v}$ takes $|u|$ arguments typed according to u and returns a tuple of $|v|$ values typed according to v.*

The expressions of interest are generated by the rules depicted in Fig. 1. By rule (ops), the basic expressions include one generator for each operator of the signature. All other basic terms define the wires that can be used to build our graphs: the *identities* (ids), the *dischargers* (bang), the *duplicators* (dup) and the *symmetries* (sym). These are the elementary bricks of our expressions, and we get the remaining ones by closing them with respect to *sequential* (seq) and *parallel* (par) *composition.* Every expression $t\colon u \to v$ generated by the inference rules is typed by a *source* and by a *target* sequence of sorts (u and v, respectively), which are relevant only for the sequential composition, which is a partial operation. A *wiring* is an arrow of $\mathbf{GS}(\Sigma)$ which is obtained from the rules of Fig. 1 without using rule (ops).

Definition 2 (gs-monoidal theory). *Given a signature Σ over a set of sorts S, the* gs-monoidal theory $\mathbf{GS}(\Sigma)$ *is the (symmetric, strict monoidal) category whose objects are the elements of S^* and whose arrows are equivalence classes of* gs-monoidal terms*, i.e., terms generated by the inference rules in Fig. 1 subject to the following conditions*

– *identities and sequential composition satisfy the axioms of categories*
[identity] $id_u ; t = t = t ; id_v$ *for all* $t : u \to v$*;*
[associativity] $t_1 ; (t_2 ; t_3) = (t_1 ; t_2) ; t_3$ *whenever any side is defined,*

– $\otimes$ *is a monoidal functor with unit* id_ϵ*, i.e., it satisfies*
[monoid] $t \otimes id_\epsilon = t = id_\epsilon \otimes t \quad t_1 \otimes (t_2 \otimes t_3) = (t_1 \otimes t_2) \otimes t_3$
[functoriality] $id_{uv} = id_u \otimes id_v$*, and*
$(t_1 \otimes t_2) ; (t_1' \otimes t_2') = (t_1 ; t_1') \otimes (t_2 ; t_2')$ *whenever both sides are defined,*

– ρ *is a symmetric monoidal natural transformation, i.e., it satisfies*
[naturality] $(t \otimes t') ; \rho_{v,v'} = \rho_{u,u'} ; (t' \otimes t)$ *for all* $t : u \to v$*,* $t' : u' \to v'$
[symmetry]
$(id_u \otimes \rho_{v,w}) ; (\rho_{u,w} \otimes id_v) = \rho_{uv,w} \quad \rho_{\epsilon,u} = \rho_{u,\epsilon} = id_u \quad \rho_{u,v} ; \rho_{v,u} = id_{uv}$

– ∇ *and* $!$ *satisfy the following axioms*
[monoidality] $\nabla_{uv} ; (id_u \otimes \rho_{v,u} \otimes id_v) = \nabla_u \otimes \nabla_v \quad !_{uv} = !_u \otimes !_v$
[unit and duplication] $!_\epsilon = \nabla_\epsilon = id_\epsilon \quad \nabla_u ; \rho_{u,u} = \nabla_u$
$\nabla_u ; (id_u \otimes \nabla_u) = \nabla_u ; (\nabla_u \otimes id_u) \quad \nabla_u ; (id_u \otimes !_u) = id_u$

Remark 1. We let $\otimes$ take precedence over ;. We shall focus on two-sorted signatures over $S = \{\bullet, \circ\}$, where $\bullet$ nodes are used for locations, while $\circ$ nodes for links. Furthermore, for ease of modelling bigraphs, we reverse the sense of direction for composing arrows, i.e. we take cogs-monoidal theories. As a matter of notation we swap implicitly the source and target of each arrow, e.g. letting

$$\rho_{\bullet,\circ} : \circ\bullet \to \bullet\circ \qquad \nabla_\bullet : \bullet^2 \to \bullet \qquad !_\circ : \epsilon \to \circ.$$

Moreover, we assume all signatures include the operator $\nu : \circ \to \epsilon$. Note that the expression $!_\circ ; \nu : \epsilon \to \epsilon$ denotes a special arrow that is the counterpart of so-called *idle edges* in bigraphs jargon. While the axiom $!_\circ ; \nu = id_\epsilon$ can be useful in many situations, we decide not to impose it here, because it is not standard for gs-monoidal theories. This point is further discussed in Section 5.

Example 1. Let us consider a (cogs) signature with three operators $f, h : \bullet \to \bullet\circ$ and $g : \bullet \to \bullet\circ^2$. Then we can compose, e.g., the expressions below:

$$\begin{aligned}
&e_1 \triangleq f \otimes id_\circ ; id_\bullet \otimes \nabla_\circ : \bullet\circ \to \bullet\circ\\
&e_2 \triangleq (!_\bullet ; h) \otimes (!_\bullet ; h) ; id_\bullet \otimes \rho_{\bullet,\circ} \otimes id_\circ ; \nabla_\bullet \otimes \nabla_\circ : \epsilon \to \bullet\circ\\
&e_3 \triangleq g \otimes id_\circ ; \rho_{\circ,\bullet} \otimes \rho_{\circ,\circ} : \bullet\circ \to \circ\bullet\circ^2\\
&e_4 \triangleq e_1 \otimes (e_2 ; e_3) ; id_\bullet \otimes (\nabla_\circ ; \nu) \otimes id_{\bullet\circ\circ} : \bullet\circ \to \bullet^2\circ^2
\end{aligned}$$

We note that a cogs-monoidal theory is a *sm Lawvere theory* [11] in which every sort is a commutative monoid.

The algebraic structure of gs-monoidal theories finds suitable realisation in graph-based modelling: arrows can be interpreted as *concrete* acyclic directed hypergraphs with interfaces, taken up to renaming of their nodes; all such graphs are represented by some arrow; any two isomorphic graphs whose interfaces match are represented by the same arrow and are thus equivalent *abstract* graphs.

We find it convenient to represent concrete gs-graphs as sets of *assignments.* We assume V is a denumerable set of S-*sorted names*, equipped with a total order $\leq$ and such that there are infinitely many names for each sort. Names are denoted by $n_1 : s_1, n_2 : s_2, \ldots$ or simply by $n_1, n_2, \ldots$ when the sort is clear from the context. A *name substitution* is a sort-preserving morphism $\sigma : V \to V$.

Remark 2. When the sort $S = \{\bullet, \circ\}$ is considered, we use $p, q, ...$ for names of sort $\bullet$ and $x, y, z, ...$ for names of sort $\circ$. When needed, we assume the order $\leq$ is induced by subscripts, i.e., that $n_i \leq n'_j$ iff $i \leq j$.

Definition 3 (assignment, multi-assignment). *Let $n'_i : s'_i$ for $i \in [1, k]$, $n_j : s_j$ for $j \in [1, h]$, $u = s'_1 \ldots s'_k$ and $v = s_1 \ldots s_h$. A* proper assignment *is written $n'_1 \ldots n'_k := f(n_1, \ldots, n_h)$ where $f \in \Sigma_{u,v}$, When $f \in \Sigma_{u,\epsilon}$ the assignment is written as $n'_1, \ldots, n'_k := f$, while when $f \in \Sigma_{\epsilon,v}$ it is written $f(n_1, \ldots, n_h)$. An* auxiliary assignment *is written either $n := n'$ (aliasing), with n and n' having the same sort, or $!(n)$ (name disposal). A* multi-assignment *G is a multiset of (proper and auxiliary) assignments.*

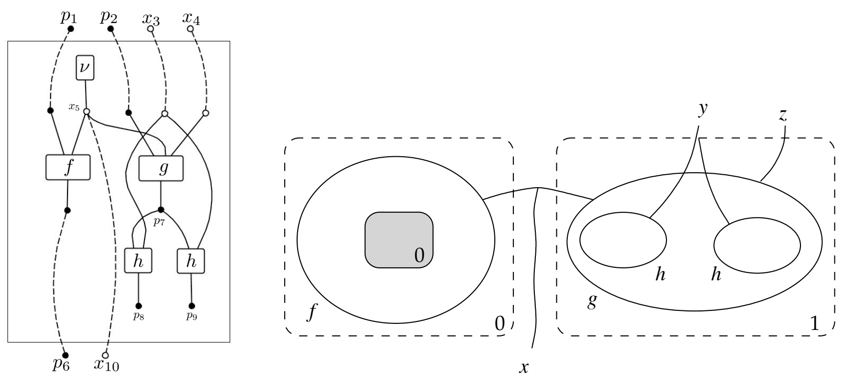
When a name appears in the left member of an assignment we say that it is *assigned*, when it appears in the right member we say that it is *used*. For an auxiliary assignment $n := n'$ we say n is an *inner connection* of the interface. The set of *outer connections* of a multi-assignment consists of all names that are used but not assigned. We denote with $ic(G)$ (resp. $oc(G)$) the list of the inner (resp. outer) connections of a multi-assignment G (ordered according to $\leq$). We say that $n \sqsubset_G n'$ if G contains an assignment where n is used and n' is assigned.

Proper assignments define the hyperedges of the graphs, whose tentacles are attached to nodes named according to their assigned and used names. Node sharing is realised by using the same name more than once. Auxiliary assignments allow to expose more references to the same node in the interface or to prevent certain nodes from appearing in the interface.

Definition 4 (gs-graph). *A* concrete gs-graph *is a multi-assignment G s.t.: (1) every name is assigned at most once; (2) the transitive closure $\sqsubset_G^+$ of $\sqsubset_G$ is irreflexive; (3) every $n \in ic(G)$ is a maximal element of $\sqsubset_G^+$; (4) for each name $n \notin ic(G)$ (exactly) one assignment $!(n)$ is present. Two concrete gs-graphs G and H are isomorphic if H can be obtained from G by applying an injective name substitution that respects the total ordering $\leq$ of the inner and outer connections. An* abstract gs-graph *(or simply* gs-graph*) is the equivalence class of a concrete gs-graph modulo isomorphism.*

Since gs-graphs are taken up to isomorphism, the exact choice of names is immaterial. The constraints on gs-graphs allow us to introduce more concise representation of gs-graphs by: (i) an auxiliary assignment of the form $!(n)$ is omitted whenever n is used in some other assignment; (ii) an auxiliary assignment $n := n'$ is omitted if n' is not an outer connection and it is a maximal element, except for n, of the partial order $\sqsubseteq^+$;

It can be shown that the gs-graphs defined over a signature Σ form a (symmetric monoidal) category that is (naturally) isomorphic to the gs-monoidal theory of Σ (see [9]). The idea is that a gs-graph G whose lists of sorts of inner



(a) A gs-graph (b) A pure bigraph

Fig. 2. Different graphical models for the same structure

connections, $ic(G)$, and outer connection, $oc(G)$ are u and v, respectively, can be regarded as an arrow $G : u \to v$. Then we can fix *atomic* gs-graphs for the basic building blocks of gs-monoidal theories and define how to compose gs-graphs in sequence $G_1; G_2$ and in parallel $G_1 \otimes G_2$ (see Appendix A).

Example 2. The arrow e_4 from Example 1 corresponds to the gs-graph $G = \{\ x_5 := \nu\ ,\ p_6 := f(p_1, x_5)\ ,\ p_7 := g(p_2, x_5, x_4)\ ,\ p_8 := h(p_7, x_3)\ ,\ p_9 := h(p_7, x_3)\ ,\ x_{10} := x_5\ ,\ !(p_8)\ ,\ !(p_9)\ \}$.

2.2 From signatures to bigraphs

The separation between different concerns is made more explicit in *bigraphs*, which are composed by two graphs, the *place graph* and the *link graph*, defined on the same set of nodes. In the literature two main classes of bigraphs have been developed: the *pure bigraphs* [12] and the *binding bigraphs* [17].

In pure bigraphs a node is not allowed to declare a local name, and the nodes communicate using only their global ports.

Definition 5 (pure signature). *A* pure signature *consists of a set $\mathcal{K}$ whose elements, called* controls*, specify the role of system nodes and a function $ar : \mathcal{K} \to \mathbb{N}$ that assigns an* arity *to each control, i.e. the number of its ports.*

A place graph is essentially a forest of unordered trees, and represents the *locality* of the nodes, that is where they are placed in the hierarchy.

Definition 6 (place graph). *Let $\mathcal{K}$ be a pure signature and m, n be a pair of ordinals, then a place graph $P : m \to n$ is a triple $(V_P, ctrl_P, prnt_P)$ where V_P is a finite set of nodes called the* support *of P (also denoted $|P|$), $ctrl_P : V_P \to \mathcal{K}$ is the* control map *assigning a control to each node and $prnt_P : m \uplus V_P \to V_P \uplus n$ is the* parent map *that describes the location of the nodes. The parent map is acyclic in the sense that for each $v \in V$ $prnt^k(v) = v$ implies $k = 0$.*

A link graph is a graph expressing the *connectivity*: an edge represent e.g. a communication medium between attached nodes.

Definition 7 (link graph). *Given a pure signature* $\mathcal{K}$ *a* link graph $L = (V_L, E_L, ctrl_L, link_L) : X \to Y$*, where* X *and* Y *are the sets respectively of* inner *and* outer names*, has finite sets of nodes* V_L *and edges* E_L*, a control map* $ctrl_L : V_L \to \mathcal{K}$ *and a* link map $link : X \uplus P_L \to E_L \uplus Y$ *with* $P_L \triangleq \sum_{v \in V_L} ar(ctrl(v))$ *the set of ports of* L*. The* support *of* L *is* $|L| \triangleq V_L \uplus E_L$*.*

The key point of bigraphs is that their place and link graphs are constructed separately; therefore the locality of a node and its connectivity can not interfere.

Definition 8 (concrete pure bigraph). *A* concrete (pure) bigraph $G = (V_G, E_G, ctrl_G, prnt_G, link_G) : \langle m, X\rangle \to \langle n, Y\rangle$ *consists of a place graph* $G^P = (V_G, ctrl_G, prnt_G) : m \to n$ *and a link graph* $G^L = (V_G, E_G, ctrl_G, link_G) : X \to Y$*. It is* lean *if it has no idle edges. We sometimes write* $G = \langle G^P, G^L\rangle$ *and the* support *of* G *is* $|G| \triangleq V_G \uplus E_G$*.*

Example 3. An example of pure bigraph is in Fig. 2(b). The place graph is represented through the nesting of nodes, while the arcs pertain to the link graph. The interface is given by pairing the interfaces of the place graph and of the link graph. The outer interface of a place graph specifies the number of distinct components forming the bigraph; to each component corresponds a *root* displayed with an enclosing dashed box. In the example we have two roots (numbered 0 and 1). The inner interface consists of the holes of the place graph, called *sites*, that serve to compose with other place graphs. Our example has one hole (numbered 0), displayed with a grey box. For the link graph, outer names are displayed on the top (y and z), and inner names on the bottom (x).

Definition 9 (support equivalence for bigraphs). *Given two bigraphs* $G, H : \langle m, X\rangle \to \langle n, Y\rangle$*, a support equivalence* $\rho : |G| \to |H|$ *is a pair of bijections* $\rho_V : V_G \to V_H$ *and* $\rho_E : E_G \to E_H$ *such that:* $ctrl_H \circ \rho_V = ctrl_G$*,* $prnt_H \circ (Id_m \uplus \rho_V) = (Id_n \uplus \rho_V) \circ prnt_G$ *and* $link_H \circ (Id_X \uplus \rho_P) = (Id_Y \uplus \rho_E) \circ link_G$*, where* $\rho_P : P_G \to P_H$ *maps the ports of* G *in those of* H *and it is defined in terms of* ρ_V*: for each port* $(v, i) \in P_G$*,* $\rho_P((v, i)) = (\rho_V(v), i)$*.*

We write $G \simeq H$ when G and H are *support equivalent*, and $G \mathrel{\hat{\approx}} H$ if they are support equivalent ignoring their idle edges (*lean-support equivalence*). The lean-support quotient yields the (partial) symmetric monoidal category of *abstract bigraphs*, denoted by $BG(\mathcal{K})$, and the lean-support quotient functor $[\![\cdot]\!]$ maps each concrete bigraph in its corresponding abstract bigraph.

Definition 10 (abstract bigraphs). *An* abstract bigraph *over* $\mathcal{K}$ *is a* $\mathrel{\hat{\approx}}$*-equivalence class* $[G] : \langle m, X\rangle \to \langle n, Y\rangle$ *of a concrete bigraph* G*.*

In binding bigraphs, besides the possibility of having local names, there is added a scope discipline for limiting the visibility of such local names. In particular a control may declare some names that only its descendants can use, thus relaxing in part the assumption of independence of the two graphical structures.

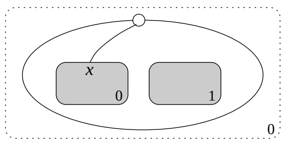


Fig. 3. A binding bigraph

Definition 11 (binding signature). *A* binding signature *has a set of controls $\mathcal{K}$ and an* arity *function $ar : \mathcal{K} \to \mathbb{N} \times \mathbb{N}$. If $ar(K) = (h, k)$, we write $K : h \to k$ and we call, respectively, h and k the* binding arity *and the* free arity *of K and they index respectively the* binding ports *and the* free ports *of K.*

Definition 12 (binding interface). *A* binding interface *is a triple $I = \langle m, loc, X\rangle$ where the ordinal m and the set of names X are as in pure bigraphs, and $loc \subseteq m \times X$ is the* locality *of the interface. If $(i, x) \in loc$ we say that i is a* place *of x. We denote by $I^u = \langle m, X\rangle$ the pure interface underlying I.*

Example 4. The approach of the binding bigraphs for avoiding misleading compositions consists in enriching the interfaces with a locality relation *loc* that establishes to which places, if any, a name belongs to. Fig. 3 denotes a simple binding bigraph with a single control with a local name and two sites in it; the locality relation on the inner interface associates the name to the first site. This restriction prevents controls in the second site from using this name.

Definition 13 (locality relation). *Let $I = \langle m, loc_I, X\rangle$ and $J = \langle n, loc_J, Y\rangle$ be binding interfaces and consider a pure bigraph $G^u : I^u \to J^u$ on the pure underlying interfaces. Then the* locality relation *$loc_G \subseteq (m \uplus n \uplus V) \times (X \uplus Y \uplus P \uplus E)$, is the smallest relation such that:*

- *if $(i, x) \in loc_I$ then $(i, x) \in loc_G$ ($loc_I \subseteq loc_G$)*
- *if $(j, x) \in loc_J$ then $(j, x) \in loc_G$ ($loc_J \subseteq loc_G$)*
- *if p is a binding port of a node v then $(v, p) \in loc_G$*
- *if p is a free port of a node v then $(prnt(v), p) \in loc_G$*
- *if an edge e contains a binding port of v then $(v, e) \in loc_G$*

Definition 14 (binding bigraph). *Given two binding interfaces I and J a* concrete binding bigraph *$G : I \to J$ consists of an underlying pure bigraph $G^u : I^u \to J^u$ such that: (a) any edge has at most one binding port, while an outer name has none; (b) if $link_G(q) = l$ is a local link then q is also local, and whenever $(w, q) \in loc_G$ then there exists w' such that $prnt^k_G(w) = w'$ for some $k \in \mathbb{N}$ and $(w', l) \in loc_G$. The condition (b) is called the* scoping rule.

3 Characterising Pure Bigraphs

This section shows that pure bigraphs are *essentially* equivalent to a particular class of gs-graphs over the sorts $\{\bullet, \circ\}$. The word "essentially" means that there is a one-to-one relation between the objects of the two models, but only up to certain bijections over the interfaces. Indeed the main difference lies in the way through which the two models view the interfaces: for a bigraph an interface is a pair composed by an ordinal and by a set of names, in a gs-graph instead the interfaces are strings over the alphabet $\{\bullet, \circ\}$. For making them comparable we need to equip each model with some missing information which is present in the other model. In a bigraphical interface $\langle m, X\rangle$ we must form a list out of the elements in $\{0, \ldots, m-1\}$ and the elements in X. For the gs-graphs instead, given a string in $\{\bullet, \circ\}^*$ we have to assign a name to each element of sort $\circ$.

The relation between pure bigraphs and gs-graphs can be sketched by looking at Fig. 2. Places correspond to hyperedges and their hierarchy is built in the gs-graph by exploiting the nodes of sort $\bullet$. Connectivity is represented by the sharing of nodes of sort $\circ$. Closed links are the $\circ$ nodes below a restriction ν. The dashed lines express which nodes are exported to the interfaces.

Interfaces. Given a bigraphical interface $\langle m, X\rangle$, every $i \in m$ is of sort $\bullet$ while the names in X are of sort $\circ$. Nevertheless if we had a gs-graph interface with exactly m elements of sort $\bullet$ and $|X|$ elements of sort $\circ$ we would not have an obvious way to map such interface in $\langle m, X\rangle$, because the $\circ$ elements are ordered unlike the names in X. Indeed, take for example the interfaces of our running example $u = \bullet^2\circ^2$ and $\langle 2, \{y, z\}\rangle$; there are two possible bijections from $\circ^2$ to $\{y, z\}$ but only one allows to establish a correct correspondence (y must match the first $\circ$ and z the second $\circ$). On the contrary, if we knew that $y < z$, the natural way would be that of choosing the right bijection. Thus we introduce a total order on the names used in the bigraphical interfaces.

Definition 15. *Let $\mathscr{X}$ be a denumerable infinite totally ordered (by $\leq$) set of names. Given a pure signature $\mathcal{K}$, we take bigraphs in which the sets of names on the interfaces are replaced by lists of names ordered through $\leq$. Given a list L we denote by $L[j]$ the $(j+1)^{th}$ element of the list for each admissible j.*

The assumption of having total ordered names makes the two type interfaces more similar, but it is not sufficient for establishing a bijective relation between them. Consider the previous example and suppose that in $\mathscr{X}$ we have $y < x < z$. The interface I can be associated, through the unique monotone bijection, to the bigraphical interface with the set $\{y, z\}$, but nothing prevents one using $\{x, z\}$ or $\{x, y\}$ instead. These considerations lead us to the following definition that embeds a particular set of names in a gs-graph interface.

Definition 16 (name choice). *Let $u \in \{\bullet, \circ\}^*$ and let $\#_u$ be the number of elements of sort $\circ$ in u. Then a* name choice *for u is an injective monotone map $\sigma_u : \#_u \to \mathscr{X}$. A gs-graph $G : u \to v$ can be equipped with two name choices σ_u, σ_v for the inner and the outer interfaces, written $G : (u, \sigma_u) \to (v, \sigma_v)$.*

Signatures. Both bigraphs and gs-graphs are based on a signature that describes the allowed operators. Therefore it is necessary to correlate the two typologies of signature. (In the rest of this section we understand that only pure signatures are considered and we use the symbol $\mathcal{K}$ also for gs-graph signatures.)

Definition 17 (equating signatures). *Consider a pure signature $\mathcal{K}$; the equivalent gs-graph signature Σ has an operator $K : \bullet \to \bullet\circ^h$ if and only if the control K of arity h is in $\mathcal{K}$.*

Graphs. With the name choices, to a string over $\{\bullet, \circ\}$ corresponds exactly one bigraphical interface, but the converse is false. Indeed a bigraphical interface over ordered names can be viewed as a concatenation of two ordered lists, the first with elements of sort $\bullet$ and the second with $\circ$-elements, while in a gs-graph interface such elements are mixed. Thus given a bigraph we can add two bijections that "shuffle" these elements as stated below:

Definition 18 (shuffled bigraphs). *A* shuffled bigraph $\langle G : \langle m, X, \phi_{in}\rangle \to \langle n, Y, \phi_{out}\rangle$ *consists of a bigraph* $G : \langle m, X\rangle \to \langle n, Y\rangle$ *and two bijections,* $\phi_{in} : m + |X| \to m + |X|$ *and* $\phi_{out} : n + |Y| \to n + |Y|$*, called shuffle functions, that preserve the relative order of the elements with the same sort, i.e.:*

- $\forall i, j \in m+|X|$ *if* $0 \le i \le j < m$ *or* $m \le i \le j < m+|X|$ *then* $\phi_{in}(i) \le \phi_{in}(j)$
- $\forall i, j \in n+|Y|$ *if* $0 \le i \le j < n$ *or* $n \le i \le j < n+|Y|$ *then* $\phi_{out}(i) \le \phi_{out}(j)$*.*

From shuffled bigraphs to gs-graphs. The first transformation that we define takes a shuffled bigraph $G = \langle V_G, E_G, ctrl_G, prnt_G, link_G\rangle : \langle m, X, \phi_{in}\rangle \to \langle l, Y, \phi_{out}\rangle$ and returns a gs-graph $H = S[\![G]\!]$ that represents it. The underlying idea is relatively simple: there is a proper assignment for each node and edge of the bigraph. In detail the edges cause the creation of assignments with the ν operator, while the nodes give assignments that describe the position of a control in the system and the interactions with the other controls, deriving it from the parent and the link map of the bigraph. In the following we denote with $\mathcal{N}_H$ the set of all names appearing in H, which is partitioned in $\mathcal{N}_H^\bullet$ and $\mathcal{N}_H^\circ$, respectively the set of all names of sort $\bullet$ and of sort $\circ$, appearing in H. Let $\mathcal{N}_H^\bullet = V_G \uplus \{s_0, \ldots, s_{m-1}\} \uplus \{r_0, \ldots, r_{l-1}\}$ and $\mathcal{N}_H^\circ = E_G \uplus \{x_0, \ldots, x_{|X|-1}\} \uplus \{y_0, \ldots, y_{|Y|-1}\}$. Note that x_i and y_j are not the names in sets X and Y, but new names used only in the gs-graph. First we need two auxiliary functions $\overline{prnt} : m \uplus V \to \mathcal{N}_H^\bullet$ and $\overline{link} : P_G \uplus X \to \mathcal{N}_H^\circ$ that translate the results of $prnt_G$ and $link_G$ into the names of the gs-graph:

$$\overline{prnt}(v) \triangleq \begin{cases} w \text{ if } prnt_G(v) = w \in V_G \\ r_i \text{ if } prnt_G(v) = i \in l \end{cases} \qquad \overline{link}(p) \triangleq \begin{cases} e \text{ if } link_G(p) = e \in E_G \\ y_i \text{ if } link_G(p) = Y[i] \end{cases}$$

Next we define the assignments of H: (1) $\forall v \in V_G$ we let $v := f(\overline{prnt}(v), \overline{link}(v, 0), \ldots, \overline{link}(v, h-1))$ where $f = ctrl_G(v)$ and h is the arity of f; (2) $\forall e \in E_G$ we let $e := \nu$; (3) $\forall i \in m$ we let $s_i := \overline{prnt}(i)$; (4) $\forall i \in |X|$ we let $x_i := \overline{link}(X[i])$ Note that $\{s_0, \ldots, s_{m-1}\} \cup \{x_0, \ldots, x_{|X|-1}\}$ are the inner connections of H while $\{r_0, \ldots, r_{l-1}\} \cup \{y_0, \ldots, y_{|Y|-1}\}$ are its outer connections. The order of these names can be retrieved using the shuffle functions: Define $\overline{\phi}_{in}^{-1} : m + |X| \to \mathcal{N}_H$ and $\overline{\phi}_{out}^{-1} : l + |Y| \to \mathcal{N}_H$ as

$$\overline{\phi}_{in}^{-1}(j) \triangleq \begin{cases} s_i & \text{if } \phi_{in}^{-1}(j) = i < m \\ x_{i-m} & \text{if } \phi_{in}^{-1}(j) = i \ge m \end{cases} \qquad \overline{\phi}_{out}^{-1}(j) \triangleq \begin{cases} r_i & \text{if } \phi_{out}^{-1}(j) = i < l \\ y_{i-l} & \text{if } \phi_{out}^{-1}(j) = i \ge l \end{cases}$$

Hence $ic(H) = (\overline{\phi}_{in}^{-1}(0), \overline{\phi}_{in}^{-1}(1), \ldots, \overline{\phi}_{in}^{-1}(m + |X| - 1))$ and $oc(H) = (\overline{\phi}_{out}^{-1}(0), \overline{\phi}_{out}^{-1}(1), \ldots, \overline{\phi}_{out}^{-1}(l + |Y| - 1))$. Finally, the transformation $S[\![G]\!]$ produces two name choices: $\sigma_{in}(i) \triangleq X[i]$ for $i \in |X|$ and $\sigma_{out}(i) \triangleq Y[i]$ for $i \in |Y|$.

From gs-graphs to shuffled bigraphs. Let $H : (u, \sigma_u) \to (v, \sigma_v)$ be a gs-graph over $\mathcal{K}$ with name choices and let us take an instance in its isomorphism class. The first step in transforming the gs-graph H in the corresponding shuffled bigraph $G = B[\![H]\!]$ consists in defining the shuffle functions ϕ_{in} and ϕ_{out}. For this purpose let m and l be the number of elements of sort $\bullet$ in the lists u and v respectively; then for each list, for example u, define a function $u^\bullet : m \to |u|$ that tell us the positions in the list u of the $\bullet$ sort elements, and a similar function $u^\circ : (|u| - m) \to |u|$ that do the same thing on the elements of u of sort $\circ$. For example if $u = \bullet \circ \circ \bullet$, then $u^\bullet = \{0 \mapsto 0, 1 \mapsto 3\}$ and $u^\circ = \{0 \mapsto 1, 1 \mapsto 2\}$. With the aid of this functions we define $\phi_{in} : |u| \to |u|$ and $\phi_{out} : |v| \to |v|$ as:

$$\phi_{in}(i) \triangleq \begin{cases} u^\bullet(i) & \text{if } 0 \leq i < m \\ u^\circ(i - m) & \text{otherwise} \end{cases} \qquad \phi_{out}(i) \triangleq \begin{cases} v^\bullet(i) & \text{if } 0 \leq i < l \\ v^\circ(i - l) & \text{otherwise} \end{cases}$$

Now recall that in a pure signature for gs-graphs every operator, except ν, is of the form $f : \bullet \to \bullet\circ^h$ for some $h \in \mathbb{N}$, thus every proper assignment over those operators takes the form $n := f(n_\bullet, n_0, \ldots, n_{h-1})$ with $n, n_\bullet$ of sort $\bullet$ and the remaining names of sort $\circ$. Then, the bigraph associated to the gs-graph $H : (u, \sigma_u) \to (v, \sigma_v)$ is $G = B[\![H]\!] = (V_G, E_G, ctrl_G, prnt_G, link_G) : \langle m, X, \phi_{in} \rangle \to \langle l, Y, \phi_{out} \rangle$ where $m, l, \phi_{in}, \phi_{out}$ are defined as above, and:

- $X[i] \triangleq \sigma_u(i)$ for each admissible i and $Y[j] \triangleq \sigma_v(j)$ for each admissible j.
- $V_G \triangleq \{n \in \mathcal{N}_H^\bullet \mid n \notin ic(H) \text{ and } n \notin oc(H)\}$ is composed by the $\bullet$-names that are not visible outside the gs-graph. Thus the names in V_G are assigned exactly once with a proper assignment.
- $E_G \triangleq \{n \in \mathcal{N}_H^\circ \mid n := \nu \in H\}$ comprises all "restricted" names of sort $\circ$.
- The control map $ctrl_G : V_G \to \mathcal{K}$ is defined as follows. Being $n \in V_G$ let $n := f(n_\bullet, n_0, \ldots, n_{h-1})$ the unique assignment of n in H, then $ctrl_G(n) = f$
- The parent map $prnt_G : m \uplus V_G \to V_G \uplus l$ is defined separately for the elements in V_G and m. For each $n \in V_G$ let $n := f(n_\bullet, n_0, \ldots, n_{h-1})$ the unique assignment of n in H, then:

$$prnt_G(n) = \begin{cases} n_\bullet & \text{if } n_\bullet \in V_G \\ \phi_{out}^{-1}(j) & \text{if } n_\bullet = oc(H)[j] \text{ for some } j \text{ in the list range} \end{cases}$$

  Take instead $i \in m$ and let $s_i = ic[\phi(i)]$. Since s_i is an inner connection, there exists in H a unique auxiliary assignment $s_i := n$.

$$prnt_G(i) = \begin{cases} n & \text{if } n \in V_G \\ \phi_{out}^{-1}(j) & \text{if } n = oc(H)[j] \text{ for some admissible } j \end{cases}$$

- Finally we define $link_G : P_G \uplus X \to E_G \uplus Y$. Take a port (n, i) with $n \in V_G$ and let $n := f(n_\bullet, n_0, \ldots, n_{h-1})$ be the proper assignment of n, then

$$link_G((n, i)) = \begin{cases} n_i & \text{if } n_i \in E_G \\ Y[\phi_{out}^{-1}(j) - l] & \text{if } n_i = oc(H)[j] \text{ for some admissible } j \end{cases}$$

Consider instead a name $x = X[i]$ and let $\overline{x} = ic(H)[\phi_{in}(m+i)]$. The auxiliary assignment associated to $\overline{x}$ is $\overline{x} := n$. Thus

$$link_G(x) = \begin{cases} n & \text{if } n \in E_G \\ Y[\phi_{out}^{-1}(i) - l] & \text{if } n = oc(H)[j] \text{ for some } j \end{cases}$$

We can now present our first main correspondence result:

Theorem 1. *Shuffled support-equivalent bigraphs over a pure signature $\mathcal{K}$ are isomorphic to gs-graphs over $\mathcal{K}$ with name choices.*

The proof shows that $B[\![S[\![\cdot]\!]]\!]$ is the identity function on shuffled bigraphs and that $S[\![B[\![\cdot]\!]]\!]$ is the identity function on gs-graphs. Although not stressed here for space limitation, the transformations $S[\![\cdot]\!]$ and $B[\![\cdot]\!]$ preserve composition and tensor (see Appendix C), i.e., we can view bigraphs and gs-graphs not only as "essentially" equivalent formulations, but as "essentially" isomorphic algebras.

4 Characterising Binding Bigraphs

While in the pure case the correspondence can be worked out smoothly, the case of binding signatures is more challenging. At the signature level, the idea is just to consider operators of the form $K : \bullet\circ^h \to \bullet\circ^k$ for h the binding arity of K and k the free arity of K. Then we can straightforwardly define an injective transformation from (shuffled) binding bigraphs to gs-graphs as a suitable extension of the one in Section 3 (see Appendix D). The main difference is that now the class of gs-graphs freely generated by the signature may contain some elements that do not correspond to any valid binding graphs, because the scope discipline is not enforced by the free construction. Thus the transformation from binding bigraphs to gs-graphs is not surjective. Moreover the set of gs-graphs that are images of bigraphs is closed under monoidal product, but not under sequential composition. To see this, consider the gs-graphs in Fig. 4: the two gs-graphs on the left trivially respect the scope rule, but their sequential composition links h to the local port x of g despite h and g are siblings.

The main result we present here is the characterisation of "well-scoped" gs-graphs in terms of a relational type system in the style of relational separation logic [21]. We start by rephrasing the location principle for the scoping rule in the context of gs-graphs. We say that a name n is *bound* to location p if p and n appears together in the left hand side of some proper assignment in G (i.e. of the form $p, ..., n, ... := ...$). Sometimes we say just that n is bound, omitting the location p to which it is bound. If a name n is not bound then it is *free* (note that, for the purpose of this section, if n is assigned by $n := \nu$ then it is not bound and thus it is said free). If a location p is not assigned then we say it is *free*. We say that q *exploits* n if q and n appears together in the right hand side of some proper assignment in G (i.e. of the form $... := f(q, ..., n, ...)$).

Definition 19 (legal gs-graph). *A gs-graph G is* legal *iff for any p, q, n such that n is bound to p and q exploits n then $q \sqsubset_G^* p$, where $\sqsubset_G^*$ is the reflexive and transitive closure of $\sqsubset_G$.*

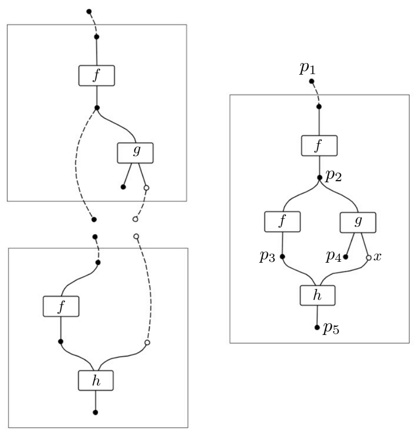


Fig. 4. An example of composition that does not respect the scope rule

As a special case, any "pure" gs-graphs is legal because pure signatures forbid the presence of names bound to locations. Our second main correspondence result is:

Theorem 2. *A gs-graph represents a binding bigraph iff it is legal.*

The proof goes by showing that the image of a binding bigraph via the transformation defined in Appendix D is a legal gs-graph (by contradiction, if it was not legal, then the scoping rule would have been violated) and then by giving a converse transformation from legal gs-graphs to binding bigraphs.

Example 5. Let us consider a binding signature with three operators $f : \bullet \to \bullet$, $g : \bullet\circ \to \bullet\circ^2$ and $h : \bullet \to \bullet\circ$. The gs-graph in Fig. 4 can be defined as $G = \{\ p_2 := f(p_1)\ ,\ p_3 := f(p_2)\ ,\ p_4, x := g(p_2)\ ,\ p_5 := h(p_3, x)\ ,\ !(p_4)\ ,\ !(p_5)\ \}$, which is not legal because p_3 exploits the node x (by the assignment $p_5 := h(p_3, x)$), which is bound to p_4 (by $p_4, x := g(p_2)$) and p_4 is not an ancestor of p_3 (i.e. $p_3 \not\subset^*_G p_4$).

We can conveniently characterise the class of legal gs-graphs by exploiting an elegant type system. The typing relations we are interested in are of the form "p uses n" and "p misuses n". We need just three inference rules:

$$\frac{p \text{ free} \quad p \text{ uses } n}{p \text{ misuses } n} \qquad \frac{p, ... := f(q, ..., n, ...) \quad n \text{ bound}}{q \text{ uses } n}$$

$$\frac{p, n_1, ..., n_k := f(q, ...) \quad p \text{ uses } n \quad \forall i.\, n \neq n_i}{q \text{ uses } n}$$

Roughly the rules says that if q exploits n and n is bound to some other location, say p', then we must check that q be a descendant of p'. This task is accomplished by propagating "upward" the dependency through the location hierarchy until either we discover that p' is an ancestor of q (in which case the propagation stops) or we reach the (free) root location p, in which case the scoping rule is violated and we assert that p misuses n. In the above example we have the typing relation $\{ p_3$ uses x , p_2 uses x , p_1 uses x , p_1 misuses $x \}$.

Proposition 1. *A gs-graph is legal iff it induces an empty "misuses" relation.*

The proof is divided in two parts. First we show that if the "misuses" relation is not empty then the gs-graph is not legal. Conversely, we show that if a gs-graph is not legal, then the "misuses" relation is not empty.

Theorem 3. *The typing relation of $G_1 \otimes G_2$ is the union of the typing relations of G_1 and G_2. The typing relation of $G_1; G_2$ is a superset of the union of the typing relations of G_1 and of G_2.*

Corollary 1. *The parallel composition of two gs-graph is legal iff both are legal. If a non legal gs-graph is used in a sequential composition the result is non legal.*

Note that for computing the typing relation of $G_1; G_2$ it is enough to close the union of the typing relations of G_1 and G_2 w.r.t. the type inference rules (i.e. the reasoning is monotonic). The typing rules induce a straightforward quadratic algorithm for checking if a gs-graph is legal or not (the complexity is $O(B_G \cdot W_G)$ for B_G the number of $\bullet$ nodes in G and W_G the number of bound $\circ$ nodes in G).

5 Concluding Remarks

In conclusion, while bigraphs and gs-graphs are equally expressive, we claim that the presentation of gs-graphs in terms of sets of assignments combines the expressiveness of name links with the simpler and more standard algebraic structure of gs-monoidal theories. We believe that the relational type systems used above to check binding gs-graphs well-formedness may also be useful for establishing important properties of systems represented as gs-graphs.

A few observations are in place that deserves some future work. First, lean support equivalence over bigraphs abstract away from idle edges. Roughly, this corresponds to garbage collect restricted names that are not used and it is convenient for representing process calculi whenever the structural axiom $(\nu\ x)\mathbf{nil} = \mathbf{nil}$ is considered. The corresponding axiom for gs-monoidal theories would be $!_\circ\ ;\ \nu = id_\epsilon$, which has not been considered in this work because it is not part of the standard theory. At the level of abstract gs-graphs, this would correspond to require that the underlying multi-assignments G are such that whenever there is a name x such that G contains both $x := \nu$ and $!(x)$, then there is at least another assignment using x. This is a bit annoying because it requires some additional bookkeeping and cleansing when composing gs-graphs. Still, we are confident that our correspondence results will carry over smoothly to ones between amended gs-monoidal theories / gs-graphs and shuffled lean-support equivalent (binding) bigraphs.

Second, the research on bigraphs finds a main motivation in the reactive system approach mentioned in the introduction, which is based on the existence of so-called *relative push-out* (RPO) and *idem push-out* (IPO) in the category of bigraphs. RPOs/IPOs serve to distil the labelled transitions from the reduction rules and derive a bisimilarity equivalence that is guaranteed to be a congruence. Some preliminary investigation for extending the RPO approach to the case of term graphs has been reported in [7]. We conjecture that the variant of the reactive approach based on so-called *groupoidal RPOs* [20] can be applied to the category of shuffled bigraphs and hence of gs-graphs. Moreover, we would like to exploit the gs-monoidal structure and 2-categorical rewriting techniques, along the lines of [5], to define a reference theory of concurrent rewrites for bigraphs and gs-graphs, which is currently missing.

Third, the fact that legal gs-graphs do not compose may suggest that their interfaces miss some additional information. In fact, while we can always represent binding bigraphs as legal gs-graphs, the interfaces of gs-graphs remain the ones defined in the encoding for pure bigraphs and thus they are not able to pair names and the locations they are bound to. One possible solution could be to fix some convention from which the binding information can be automatically inferred. For example, we can assume that the names ($\circ$) listed in the interface are bound to the rightmost location ($\bullet$) appearing on their right, if any (and they are free otherwise) and use such hypotheses for checking that the gs-graph is legal or not. Yet, the information about the sharing of names between two or more location would get lost. We discarded this approach because, e.g., it would forbid the composition of legal gs-graphs with many arrows (i.e. symmetries like $\rho_{\circ,\bullet}$) that has no effect whatsoever on the essence of the gs-graph, but would change the hypotheses under which it has been tagged as legal. We plan to investigate this issue in more detail, as we think it has still many advantages over other proposals, like [10], which resort to the introduction of a much more powerful closed monoidal structure for the purpose.

Fourth, we would like to extend the comparison between binding bigraphs and legal gs-graphs to the algebra of graphs with nesting proposed in [1].

References

1. R. Bruni, A. Corradini, F. Gadducci, A. Lluch-Lafuente, and U. Montanari. On gs-monoidal theories for graphs with nesting. In *Graph Transformations and Model-Driven Engineering*, volume 5765 of *LNCS*, pages 59–86. Springer, 2010.
2. R. Bruni, F. Gadducci, and U. Montanari. Normal forms for algebras of connection. *Theor. Comput. Sci.*, 286(2):247–292, 2002.
3. R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *TPLP*, 1(6):647–690, 2001.
4. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
5. A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *CTCS'97*, volume 1290 of *LNCS*, pages 87–105. Springer, 1997.
6. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7(4):299–331, 1999.

7. A. Corradini and F. Gadducci. On term graphs as an adhesive category. *Electr. Notes Theor. Comput. Sci.*, 127(5):43–56, 2005.
8. T. C. Damgaard and L. Birkedal. Axiomatizing binding bigraphs. *Nord. J. Comput.*, 13(1-2):58–77, 2006.
9. G. L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Inf. Comput.*, 156(1-2):173–235, 2000.
10. R. H. G. Garner, T. Hirschowitz, and A. Pardon. Variable binding, symmetric monoidal closed theories, and bigraphs. In *CONCUR'09*, volume 5710 of *LNCS*, pages 321–337. Springer, 2009.
11. M. Hyland and J. Power. Two-dimensional linear algebra. *Electr. Notes Theor. Comput. Sci.*, 44(1):227–240, 2001.
12. O. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, 2004.
13. O. H. Jensen and R. Milner. Bigraphs and transitions. In *POPL*, pages 38–49, 2003.
14. S. MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics 5. Springer, 2nd edition, 1998.
15. J. Meseguer. Membership algebra as a logical framework for equational specification. In *WADT'97*, volume 1376 of *LNCS*, pages 18–61. Springer, 1997.
16. R. Milner. Bigraphical reactive systems. In *CONCUR'01*, volume 2154 of *LNCS*, pages 16–35. Springer, 2001.
17. R. Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, University of Cambridge, 2004.
18. R. Milner. Bigraphs and their algebra. *Electr. Notes Theor. Comput. Sci.*, 209:5–19, 2008.
19. R. Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.
20. V. Sassone and P. Sobocinski. Locating reaction with 2-categories. *Theor. Comput. Sci.*, 333(1-2):297–327, 2005.
21. H. Yang. Relational separation logic. *Theor. Comput. Sci.*, 375(1-3):308–334, 2007.

A Composition of GS-Graphs

Atomic gs-graphs are defined as follows:

$$\begin{array}{rl}
\text{(ops)} & f \triangleq \{n'_1, \ldots, n'_{|u|} := f(n_1, \ldots n_{|v|})\} \quad \text{for any } f \in \Sigma_{u,v} \\
\text{(ids)} & id_s \triangleq \{n_2 := n_1\} \quad \text{for } n_1, n_2 \text{ of sort } s \\
\text{(sym)} & \rho_{s,s'} \triangleq \{n_3 := n_2\ ,\ n_4 := n_1\} \quad \text{for } n_1, n_4 \text{ of sort } s' \text{ and } n_2, n_3 \text{ of sort } s \\
\text{(dup)} & \nabla_s \triangleq \{n_2 := n_1\ ,\ n_3 := n_1\} \quad \text{for } n_1, n_2, n_3 \text{ of sort } s \\
\text{(bang)} & !_s \triangleq !(n) \quad \text{for } n \text{ of sort } s
\end{array}$$

Sequential and parallel composition are then defined below.

(seq): Let $G_1 : u \to v$ and $G_2 : v \to w$ such that $oc(G_1) = ic(G_2)$ and that no other names are shared between G_1 and G_2. Let A be the set of assignments in G_2 of the form $n := n'$ and let σ the corresponding name substitution. Their *sequential composition* is the gs-graph: $G_1; G_2 \triangleq (G_1\sigma) \cup (G_2 \backslash A) : u \to w$ with $ic(G_1; G_2) = ic(G_1)$ and $oc(G_1; G_2) = oc(G_2)$.

(par): Let $G_1 : u_1 \to v_1$ and $G_2 : v_2 \to w_2$ such that for every name n in G_1 and n' in G_2 we have $n \leq n'$. Their *parallel composition* is the gs-graph $G_1 \otimes G_2 : u_1u_2 \to v_1v_2 = G_1 \cup G_2$ for which we have $ic(G_1 \otimes G_2) = ic(G_1)ic(G_2)$ and $oc(G_1 \otimes G_2) = oc(G_1)oc(G_2)$.

Note that, if a composition is sound at the level of sorts, it is always possible to find suitable concrete gs-graphs in the equivalence classes that satisfy the constraints on names required to define their composition.

B More on (Pure) Bigraphs

B.1 Composition of Place Graphs

Let $P : m \to n$ and $Q : n \to l$ two concrete place graphs with $V_P \cap V_Q = \emptyset$, then their composition is defined as $Q \circ P \triangleq (V, \mathit{ctrl}, \mathit{prnt}) : m \to l$, where $V \triangleq V_P \uplus V_Q$ and

- for each $v \in V$ $\mathit{ctrl}(v) \triangleq \begin{cases} \mathit{ctrl}_P(v) \text{ if } v \in V_P \\ \mathit{ctrl}_Q(v) \text{ if } v \in V_Q \end{cases}$
- for each $v \in m \uplus V$ $\mathit{prnt}(v) \triangleq \begin{cases} \mathit{prnt}_P(v) \text{ if } v \in m \uplus V_P \text{ and } \mathit{prnt}_P(v) \in V_P \\ \mathit{prnt}_Q(i) \text{ if } v \in m \uplus V_P \text{ and } \mathit{prnt}_P(v) = i \in n \\ \mathit{prnt}_Q(v) \text{ if } v \in V_Q \end{cases}$

The identity place graph at m is $id_m \triangleq (\emptyset, \emptyset_{\mathcal{K}}, Id_m) : m \to m$ where Id_m is the identity function on the ordinal m.

The tensor product $\otimes$ on interfaces is the addition of ordinals and the unit object is 0. For $i \in \{0, 1\}$ let $P_i = (V_{P_i}, \mathit{ctrl}_{P_i}, \mathit{prnt}_{P_i}) : m_i \to n_i$ two place graphs having disjoint supports. Then

$$P_0 \otimes P_1 \triangleq (V_{P_0} \uplus V_{P_1}, \mathit{ctrl}_{P_0} \uplus \mathit{ctrl}_{P_1}, \mathit{prnt}_{P_0 \otimes P_1}) : m_0 + m_1 \to n_0 + n_1$$

where for each $j \in m_0 + m_1$

$$\mathit{prnt}_{P_0 \otimes P_1}(j) \triangleq \begin{cases} \mathit{prnt}_{P_0}(j) & \text{if } j < m_0 \\ \mathit{prnt}_{P_1}(j - m_0) & \text{if } j \geq m_0 \text{ and } \mathit{prnt}_{P_1}(j - m_0) \in V_{P_1} \\ \mathit{prnt}_{P_1}(j - m_0) + n_0 & \text{if } j \geq m_0 \text{ and } \mathit{prnt}_{P_1}(j - m_0) \in n_1 \end{cases}$$

and for each $v \in V_{P_0} \uplus V_{P_1}$

$$\mathit{prnt}_{P_0 \otimes P_1}(v) \triangleq \begin{cases} \mathit{prnt}_{P_0}(v) & \text{if } v \in V_{P_0} \\ \mathit{prnt}_{P_1}(v) & \text{if } v \in V_{P_1} \text{ and } \mathit{prnt}_{P_1}(v) \in V_{P_1} \\ \mathit{prnt}_{P_1}(v) + n_0 & \text{if } v \in V_{P_1} \text{ and } \mathit{prnt}_{P_1}(v) \in n_1 \end{cases}$$

Symmetries $\gamma_{m,n}$ have empty support and their effect is to swap sites:

$$\gamma_{m,n} \triangleq (\emptyset, \emptyset_{\mathcal{K}}, \mathit{prnt}(j) = \begin{cases} j + n \text{ if } j < m \\ j - m \text{ if } j \geq m \end{cases})$$

B.2 Composition of Link Graphs

Let $L : X \to Y$ and $M : Y \to Z$ be two link graphs such that $V_L \cap V_M = E_L \cap E_M = \emptyset$, then their composition is defined as: $M \circ L \triangleq (V, E, \mathit{ctrl}, \mathit{link}) : X \to Z$, where $V \triangleq V_L \uplus V_M$, $E \triangleq E_L \uplus E_M$ and

- for each $v \in V$ $\mathit{ctrl}(v) \triangleq \begin{cases} \mathit{ctrl}_L(v) & \text{if } v \in V_L \\ \mathit{ctrl}_M(v) & \text{if } v \in V_M \end{cases}$
- given a point $p \in X \uplus P_L \uplus P_M$ of $M \circ L$ then

$$\mathit{link}(p) \triangleq \begin{cases} \mathit{link}_L(p) & \text{if } p \in X \uplus P_L \text{ and } \mathit{link}_L(p) \in E_L \\ \mathit{link}_M(y) & \text{if } p \in X \uplus P_L \text{ and } \mathit{link}_L(p) = y \in Y \\ \mathit{link}_M(p) & \text{if } p \in P_M \end{cases}$$

The identity link graph at X is $id_X \triangleq (\emptyset, \emptyset, \emptyset_{\mathcal{K}}, Id_X) : X \to X$, with $Id_X : X \to X$ the identity function on the set X.

The product $\otimes$ is defined only on disjoint link graph interfaces, i.e. on disjoint sets of names, and it is roughly the disjoint set union. Suppose that for $i \in \{0, 1\}$, $L_i = (V_{L_i}, E_{L_i}, \mathit{ctrl}_{L_i}, \mathit{link}_{L_i}) : X_i \to Y_i$ are link graphs with disjoint supports and with $X_0 \cap X_1 = Y_0 \cap Y_1 = \emptyset$. Their product is:

$$L_0 \otimes L_1 \triangleq (V_{L_0} \uplus V_{L_1}, E_{L_0} \uplus E_{L_1}, \mathit{ctrl}_{L_0} \uplus \mathit{ctrl}_{L_1}, \mathit{link}_{L_0} \uplus \mathit{link}_{L_1}) : X_0 \uplus X_1 \to Y_0 \uplus Y_1$$

The unit object is the empty set $\emptyset$ and the symmetries $\gamma_{X,Y}$ are simply identities on $X \uplus Y$: $\gamma_{X_Y} \triangleq id_{X \uplus Y}$.

B.3 Composition of Bigraphs

Given two concrete bigraphs $G = (V_G, E_G, \mathit{ctrl}_G, \mathit{prnt}_G, \mathit{link}_G) : \langle m, X\rangle \to \langle n, Y\rangle$ and $H = \langle H^P, H^L\rangle : \langle l, Z\rangle$ with $V_G \cap V_H = E_G \cap E_H = \emptyset$, their composition is defined componentwise: $H \circ G \triangleq \langle H^P \circ G^P, H^L \circ G^L\rangle : \langle m, X\rangle \to \langle l, Z\rangle$

The identity bigraph at $\langle m, X\rangle$ is $id_{\langle m,X\rangle} \triangleq \langle id_m, id_X\rangle$.

Given two bigraph interfaces $\langle m, X\rangle$ and $\langle n, Y\rangle$ with $X \cap Y = \emptyset$, the product is defined componentwise: $\langle m, X\rangle \otimes \langle n, y\rangle = \langle m \otimes n, X \otimes Y\rangle$. The same happens on bigraphs: let for $i \in \{0, 1\}$ $G_i = \langle G^P, G^L\rangle : \langle m_i, X_i\rangle \to \langle n_i, Y_i\rangle$ be two bigraphs with disjoint supports and $X_0 \cap X_1 = Y_0 \cap Y_1 = \emptyset$, then

$$G_0 \otimes G_1 \triangleq \langle G_0^P \otimes G_1^P, G_0^L \otimes G_1^L\rangle : \langle m_0 + m_1, X_0 \uplus X_1\rangle \to \langle n_0 + n_1, Y_0 \uplus Y_1\rangle$$

The unit object ϵ is the pairing of the unit objects of the place graph and link graph products: $\epsilon = \langle 0, \emptyset\rangle$. Finally, for a pair of interfaces $\langle m, X\rangle$ and $\langle n, Y\rangle$ for which the product is defined, the symmetry is $\gamma_{\langle m,X\rangle,\langle n,Y\rangle} \triangleq \langle \gamma_{m,n}, \gamma_{X,Y}\rangle$.

B.4 Discrete Normal Form

Definition 20 (prime and discrete bigraph). *A bigraphical interface $\langle m, X\rangle$ is* prime *if $m = 1$ and we write it $\langle X\rangle$. A prime bigraph $G : m \to \langle X\rangle$ has no inner names and a prime outer interface. A bigraph D is* discrete *if it has no closed links, and its link map is bijective.*

Proposition 2 (discrete normal form). *Every bigraph $G : \langle m, X\rangle \to \langle n, Z\rangle$ can be expressed uniquely, up to a renaming on Y, as $G = (id_n \otimes \lambda) \circ D$, where $\lambda : Y \to Z$ is a linking and $D : \langle m, X\rangle \to \langle n, Y\rangle$ is discrete.*

Moreover, every discrete bigraph D may be factorised uniquely, up to a permutation of the sites of each factor, as $D = \alpha \otimes ((P_0 \otimes \cdots \otimes P_{n-1}) \circ \pi)$, with α a renaming, each P_i prime and discrete, and π a permutation of all the sites.

C Composition Preserving Transformations

Proposition 3 ($S[\![\cdot]\!]$ preserves operations). *Suppose that $G : \langle m, X, \phi_{in}\rangle \to \langle n, Y, \psi\rangle$, and $G' : \langle n, Y, \psi\rangle \to \langle l, Z, \phi_{out}\rangle$ are two well formed shuffled bigraphs. We have that $S[\![G' \circ G]\!] = S[\![G]\!]; S[\![G']\!]$.*

Now consider $G_0 : \langle m_0, X_0, \phi_0\rangle \to \langle n_0, Y_0, \psi_0\rangle$ and $G_1 : \langle m_1, X_1, \phi_1\rangle \to \langle n_1, Y_1, \psi_1\rangle$ with $X_0 < X_1$ and $Y_0 < Y_1$, then $S[\![G_0 \otimes G_1]\!] = S[\![G_0]\!] \otimes S[\![G_1]\!]$.

Proposition 4 ($B[\![\cdot]\!]$ preserves operations). *Let $H : (u, \sigma_u) \to (v, \sigma_v)$ and $H' : (v, \sigma_v) \to (w, \sigma_w)$ be gs-graphs with name choices, then $B[\![H; H']\!] = B[\![H']\!] \circ B[\![H]\!]$.*

Consider instead $H_0 : (u_0, \sigma_{u_0}) \to (v_0, \sigma_{v_0})$ and $H_1 : (u_1, \sigma_{u_1}) \to (v_1, \sigma_{v_1})$ gs-graphs with name choices such that $Im(\sigma_{u_0}) < Im(\sigma_{u_1})$ and $Im(\sigma_{v_0}) < Im(\sigma_{v_1})$. We have: $B[\![H_0 \otimes H_1]\!] = B[\![H_0]\!] \otimes B[\![H_1]\!]$.

D Transforming binding bigraphs in gs-graphs

The construction of the gs-graph representing a certain binding bigraph is not so different from that relative to pure bigraphs. In fact, as previously mentioned, we can not represent the locality relations in the context of gs-graphs and the transformation is forced to ignore them. Therefore we have to deal only with the added possibility for controls to declare names. Likewise the pure case, we work with gs-graphs equipped with name choices on the interfaces and with shuffled binding bigraphs. The latter are defined exactly like shuffled pure bigraphs (see Definition 18), except that in place of a pure bigraph we have clearly a binding bigraph. Let $G = (V_G, E_G, ctrl_G, prnt_G, link_G) : \langle m, loc_{in}, X, \phi_{in}\rangle \to \langle l, loc_{out}, Y, \phi_{out}\rangle$ be a shuffled binding bigraph on a signature $\mathcal{K}$ and denote with $P_G^{loc} \subseteq P_G$ the set of all its local ports. In particular given a node $v \in V_G$ such that its associated control $ctrl_G(v)$ has a positive internal arity, we call $(v, local_i)$ the $(i+1)^{th}$ local port declared by v.

In the gs-graph $H = S_{bind}[\![G]\!]$ the following names will appear:

$$\begin{aligned}\mathcal{N}_H^{\bullet} &= V_G \uplus \{s_0, \ldots, s_{m-1}\} \uplus \{r_0, \ldots, r_{l-1}\}\\ \mathcal{N}_H^{\circ} &= E_G \uplus \{x_0, \ldots, x_{|X|-1}\} \uplus \{y_0, \ldots, y_{|Y|-1}\}\end{aligned}$$

Note that such sets of names are the same of those defined by the analogous transformation for the pure case. The difference is in the role played by the edges since in binding bigraphs they can be attached to local ports. In the corresponding gs-graph such local edges are not assigned with the restriction operator ν, but within the proper assignment of the node that declares the local port to which it is linked. Since every edge can be attached to at most one local port (see Definition 14) we are guaranteed that each local edge is assigned exactly once. In the following we denote with E_G^{local} the set of all local edges belonging to the bigraph G.

As in Section 3 the overlined maps $\overline{prnt} : m \uplus V \to \mathcal{N}_H^\bullet$ and $\overline{link} : P_G \uplus X \to \mathcal{N}_H^\circ$ will help us in having a more concise representation of the assignments of H and we do not make any change to them. Nevertheless we report their definition here for a more quick reference.

$$\overline{prnt}(v) = \begin{cases} w \text{ if } prnt_G(v) = w \in V_G \\ r_i \text{ if } prnt_G(v) = i \in l \end{cases} \qquad \overline{link}(p) \triangleq \begin{cases} e \text{ if } link_G(p) = e \in E_G \\ y_i \text{ if } link_G(p) = Y[i] \end{cases}$$

Then we give the definitions of the assignments in H.

- $\forall v \in V_G$ with $ctrl_G(v) = f$ we add the assignment

  $v\,\overline{link}(v, local_0)\ \ldots \overline{link}(v, local_{k-1}) := f(\overline{prnt}(v), \overline{link}(v, 0), \ldots, \overline{link}(v, k{-}1))$

  where k and h are respectively the binding and the free arity of f. Note that since in binding bigraph a local port can be linked to an outer name, the link map and its overlined version applied on a local port return always an edge (that clearly is local).
- $\forall e \in E_G \backslash E_G^{local}$ we add $e := \nu$
- $\forall i \in m$ we add $s_i := \overline{prnt}(i)$
- $\forall x \in \{x_0, \ldots, x_{|X|-1}\}$ we add $x_i := \overline{link}(X[i])$

The inner and the outer connections are $\{s_0, \ldots, s_{m-1}\} \cup \{x_0, \ldots, x_{|X|-1}\}$ and $\{r_0, \ldots, r_{l-1}\} \cup \{y_0, \ldots, y_{|Y|-1}\}$ respectively and their order is obtained through the shuffle functions. In particular

$$\begin{aligned} ic(H) &= (\overline{\phi}_{in}^{-1}(0), \overline{\phi}_{in}^{-1}(1), \ldots, \overline{\phi}_{in}^{-1}(m + |X| - 1)) \\ oc(H) &= (\overline{\phi}_{out}^{-1}(0), \overline{\phi}_{out}^{-1}(1), \ldots, \overline{\phi}_{out}^{-1}(l + |Y| - 1)) \end{aligned}$$

where $\overline{\phi}_{in}^{-1}$ and $\overline{\phi}_{out}^{-1}$ are defined as in Section 3.

In conclusion the name choices for the interfaces of $S_{bind}[\![G]\!]$ are $\sigma_{in}(i) \triangleq X[i]$ for $i \in |X|$ and $\sigma_{out}(i) \triangleq Y[i]$ for $i \in |Y|$.

We can now prove that the gs-graphs produced by $S_{bind}[\![\cdot]\!]$ effectively represent binding bigraphs, or better, their bodies. We could not indeed encode the locality relation on the gs-graph interfaces, but we can show that the internal structure of a binding bigraph is faithfully represented.

For this purpose we abstract from the locality relation put on the interfaces and we identify all the binding bigraphs that differ only in these relations. We write $G \approx G'$ if G and G' are such two bigraphs and it follows immediately that $\approx$ is an equivalence relation. Furthermore, since $S_{bind}[\![\cdot]\!]$ does not take into account the locality relation, we have that $G \approx G'$ implies $S_{bind}[\![G]\!] = S_{bind}[\![G']\!]$ and therefore we can give a well-defined mapping on equivalence classes $S_{bind}\backslash_{\approx}[\![\centerdot]\!]$ such that $S_{bind}\backslash_{\approx}[\![[G]]\!] = S_{bind}[\![G]\!]$, where $[G]$ denote the equivalence class of G.

Proposition 5. *The mapping $S_{bind}\backslash_{approx}[\![\centerdot]\!]$ is injective.*

Unfortunately this mapping is not surjective. Next proposition guarantees that in the image of the mapping $S_{bind}[\![\centerdot]\!]$ there are no unacceptable gs-graphs.

Proposition 6. *Let G be a binding bigraph. If in $S_{bind}[\![G]\!]$ there are an assignment $n \ldots := f(v, \ldots, e, \ldots)$ and an assignment $w \ldots e \ldots := g(\ldots)$ then $w \sqsubset^{+} v$.*

Finally we show that $S_{bind}[\![\centerdot]\!]$ preserves the operations.

Proposition 7 ($S_{bind}[\![\centerdot]\!]$ preserves operations). *Let $G : \langle m, loc_I, X, \phi_{in}\rangle \to \langle n, loc_J, Y, \psi\rangle$ and $G' : \langle n, loc_J, Y, \psi\rangle \to \langle l, loc_H, Z, \phi_{out}\rangle$, be shuffled binding bigraphs. Then $S_{bind}[\![G' \circ G]\!] = S_{bind}[\![G]\!]; S_{bind}[\![G']\!]$*

Suppose that $(G_0 : \langle m_0, loc_{I_0}, X_0, \phi_0\rangle \to \langle n_0, loc_{J_0}, Y_0, \psi_0\rangle)$ and $(G_1 : \langle m_1, loc_{I_1}, X_1, \phi_1\rangle \to \langle n_1, loc_{J_1}, Y_1, \psi_1\rangle)$ are shuffled binding bigraphs with $X_0 < X_1$ and $Y_0 < Y_1$. Then $S_{bind}[\![G_0 \otimes G_1]\!] = S_{bind}[\![G_0]\!] \otimes S_{bind}[\![G_1]\!]$

Algorithms and data structures for massive data: what's next?

Paolo Ferragina

Dipartimento di Informatica
University of Pisa, Italy
`ferragina@di.unipi.it`

Abstract. In this talk I'll survey the last 30 years of data-structure design, showing that the storage and indexing of large datasets has led algorithm and software developers to optimize, in the course of these years, several computational resources, such as time, space, I/Os, compression, just to name a few. One of the key results of this impressive flow of research has been that, nowadays, it is known how to index almost any (complex) data type in compressed space and to efficiently support various kinds of query operations. In many cases, the asymptotic performance obtainable over the compressed data is equivalent to the one achievable over the original (and uncompressed) raw data; sometimes, compression induces even a speed-up in practice because of a better use of memory/IO/CPU resources.
Recently, the advent of (big) data centers and the ubiquitous use of mobile devices, has raised the attention toward another resource: energy efficiency. This triggered a significant amount of research in all areas of IT, leading to believe that improvements in the energy efficiency of computing devices will be much more dramatic, and eventually have much greater impact, than in other areas of technology. However, the average power consumption and computation rates of computing devices are intricately tied together, making it difficult to speak of power complexity in isolation. So the next challenge will be, in my opinion, to design algorithms and data structures which optimize, or trade in a principled way, various computational resources simultaneously. This is what system engineers are addressing everyday by means of proper heuristics. But we will argue in this talk that this design is sophisticated to be done in a principled way, and needs a joint effort with other CS fields, such as Operational Research and Graph Theory.

On the expressive power of the shuffle product (Extended Abstract)

Antonio Restivo

Dipartimento di Matematica e Informatica, Università di Palermo, Palermo Italy

1 Introduction

A very general problem in the theory of formal languages is, given a "basis" of languages and a set of operations, to characterize the family of languages expressible from the "basis" by using the operations. In practice, a basis of languages will consists of a set of very simple languages, such as the languages of the form $\{a\}$, where a is a letter of the alphabet. In the theory of *regular* languages, the operations taken into account are usually the Boolean operations, the concatenation and the (Kleene) star operation.

In this setting, two families of languages play a fundamental role: the family *REG* of regular languages, and the family *SF* of star-free languages. *REG* is defined as the smallest family of languages containing the languages of the form $\{a\}$, where a is a letter, and $\{\epsilon\}$, where ϵ is the empty word, and closed under union, concatenation and star. It is well known that the family *REG* is closed also under all Boolean operations. The family *SF* of star-free languages is the smallest family of languages containing the languages of the form $\{a\}$ and $\{\epsilon\}$, and closed under Boolean operations and concatenation.

Another operation that plays an important role in the theory of formal languages is the *shuffle* operation. Recall that the *shuffle product* (or simply *shuffle*) of two languages L_1, L_2 is the language

$$L_1 \sqcup\!\sqcup L_2 = \{u_1v_1...u_nv_n | n \geq 0, u_1...u_n \in L_1, v_1...v_n \in L_2\}.$$

It is well known (cf [5]) that the family *REG* of regular languages is closed under shuffle. The study of subfamilies of regular languages closed under shuffle is a difficult problem, partly motivated by its applications to the modeling of process algebras [1] and to program verification.

In particular, we here consider the smallest family of languages containing the languages of the form $\{a\}$ and $\{\epsilon\}$, and closed under Boolean operations, concatenation and shuffle. Let us call *intermixed* the languages in this family, which is denoted by *INT*. It is perhaps surprising that the following important problem in the theory of regular languages is still open, and to a large extent unexplored.

Problem 1 *Give a (decidable) characterization of the family INT.*

In this talk we discuss this problem: we present some partial results and we introduce new special problems as possible steps in the characterization of the family *INT*. Such partial results and special problems show the deep connections of Problem 1 with other relevant aspects of formal languages theory and combinatorics on words. The results here presented are essentially based on the papers [2] and [3].

2 Star-Free and Intermixed Languages

In [2] it is proved the following theorem showing that the family *INT* of intermixed languages is strictly included in the family *REG* of regular languages and strictly contains the family *SF* of star-free languages.

Theorem 21 $SF \subsetneq INT \subsetneq REG$

Moreover, in [2] it is shown that the family *INT* is closed under quotients, but it is not closed under inverse morphism. Therefore, the family *INT* is not a variety of languages (cf [11]), and so it cannot be characterized in terms of syntactic monoids.

Let us recall (cf [9])that a language $L \subseteq \Sigma^*$ is said to be *aperiodic*, or *non-counting*, if there exists an integer $n > 0$ such that for all $x, y, z \in \Sigma^*$ one has

$$xy^n z \in L \Leftrightarrow xy^{n+1} z \in L.$$

A fundamental theorem of Schutzenberger states that *a regular language is star-free if and only if it is aperiodic.*

The strict inclusion between the families *SF* and *INT* implies that the shuffle of two star-free languages in general is not star-free. This means, roughly speaking, that *the shuffle creates periodicities.*

In order to enlighten on the difficult Problem 1, in this talk we consider the following

Problem 2 *Determine conditions under which the shuffle of two star-free languages is star-free too.*

A first condition is obtained in [3] by introducing a weaker version of the shuffle product, called *bounded shuffle.*

Let k be a positive integer. The *k-shuffle* of two languages $L_1, L_2 \subseteq \Sigma^*$ is defined as follows:

$$L_1 \sqcup\!\sqcup_k L_2 = \{u_1 v_1 ... u_m v_m | m \leq k, u_1 ... u_m \in L_1, v_1 ... v_m \in L_2\}.$$

Any k-shuffle is called *bounded shuffle.* It is not difficult to show that the family *REG* of regular languages is closed under bounded shuffle. In [3] it is proved the following theorem.

Theorem 22 *SF is closed under bounded shuffle, i.e. if* $L_1, L_2 \in SF$ *then* $L_1 \sqcup\!\sqcup_k L_2 \in SF$*, for any* $k \geq 1$.

One can derive the following corollary.

Corollary 23 *The shuffle of a star-free language and a finite language is star-free.*

3 Partial Commutations

The family *SF* is closed under concatenation and it is not closed under shuffle. What is the difference between concatenation and shuffle?

In this section we introduce an operation between languages, that generalizes at the same time concatenation and shuffle, and we investigate the closure of *SF* with respect to this operation. The new operation is defined by introducing a partial commutation between the letters of the alphabet, and its appropriate setting is the theory of *traces* (cf [4]).

Let Γ be a finite alphabet and let $\theta \subseteq \Gamma \times \Gamma$ be a symmetric and irreflexive relation called the *(partial) commutation relation.* We consider the congruence $\sim_\theta$ of Γ^* generated by the set of pairs (ab, ba) with $(a, b) \in \theta$. If $L \subseteq \Gamma^*$ is a language, $[L]_\theta$ denoted the closure of L by $\sim_\theta$, and L is *closed by* $\sim_\theta$ if $L = [L]_\theta$. The closed subsets of Γ^* are called *trace languages.*

Let now L_1 and L_2 be two languages over the alphabet Σ

Let us consider two disjoint copies Σ_1 and Σ_2 of the alphabet Σ, i.e. such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, and the isomorphism σ_1 from Σ_1^* to Σ^* and σ_2 from Σ_2^* to Σ^*.

Let L'_1 (L'_2 resp.) be the subset of Σ_1^* (Σ_2^* resp.) corresponding to L_1 (L_2 resp.) under the isomorphism σ_1 (σ_2 resp.). Let us consider the morphism $\sigma : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma^*$ defined as follows:

$$\sigma(a) = \begin{cases} \sigma_1(a), \text{ if } a \in \Sigma_1^*; \\ \sigma_2(a), \text{ if } a \in \Sigma_2^*. \end{cases}$$

Let θ be of the form $\theta \subseteq \Sigma_1 \times \Sigma_2$. The $\theta - product$ (denoted by $\sqcup\!\sqcup_\theta$) of the languages $L_1, L_2 \subseteq \Sigma^*$ is defined as follows:

$$L_1 \sqcup\!\sqcup_\theta L_2 = \sigma([L'_1 L'_2]_\theta).$$

Remark that the product (concatenation) and the shuffle correspond to two special (extremal) cases of the $\theta{-}product$. Indeed, if $\theta = \emptyset$ then $L_1 \sqcup\!\sqcup_\theta L_2 = L_1 L_2$, and, if $\theta = \Sigma_1 \times \Sigma_2$, then $L_1 \sqcup\!\sqcup_\theta L_2 = L_1 \sqcup\!\sqcup L_2$.

The partial commutation $\theta \subseteq \Sigma_1 \times \Sigma_2$ induces a partial commutation θ' on Σ defined as follows: if $(a, b) \in \theta$ the $(\sigma_1(a), \sigma_2(b)) \in \theta'$.

In [6] it is proved the following theorem.

Theorem 31 *Let L_1 and L_2 be two languages closed under θ', i.e., $[L_1]_{\theta'} = L_1$ and $[L_2]_{\theta'} = L_2$. If L_1 and $L_2 \in SF$, then $L_1 \sqcup\!\sqcup_\theta L_2 \in SF$.*

The theorem states, roughly speaking, that, if *internal* commutation (i.e., the commutations allowed inside each of the languages L_1 and L_2) is the "same" as the *external* commutation (i.e., the commutations between the letters in L_1 and the letters in L_2), then the $\theta - product$ preserves the star-freeness.

Special cases of the previous theorem are the well known result that the concatenation of two star-free languages is star-free, and the result of J.F. Perrot (cf [10]) that *the shuffle of two* commutative *star-free languages is star-free.*

4 Unambiguous Star-Free languages

In this section we investigate some conditions for Problem 2, related to the unambiguity of the product of languages.

A language $L \subseteq \Sigma^*$ is a *marked product* of the languages $L_0, L_1, ..., L_n$ if

$$L = L_0 a_1 L_1 a_2 L_2 ... a_n L_n,$$

for some letters $a_1, a_2, ..., a_n$ of Σ.

It is known (cf [13]) that the family *SF* of star-free languages is the smallest Boolean algebra of languages of Σ^* which is closed under marked product.

A marked product $L = L_0 a_1 L_1 a_2 L_2 ... a_n L_n$ is said to be *unambiguous* if every word u of L admits a *unique* decomposition

$$u = u_0 a_1 u_1 ... a_n u_n,$$

with $u_0 \in L_0, u_1 \in L_1, ..., u_n \in L_n$. For instance, the marked product $\{a, c\}^* a \{\epsilon\} b \{b, c\}^*$ is unambiguous.

Let us define the family *USF* of *unambiguous star-free* languages as the smallest Boolean algebra of languages of Σ^* containing the languages of the form A^*, for $A \subseteq \Sigma$, which is closed under unambiguous marked product (cf [13]).

The family *USF* is a very robust class of languages: the languages in this family admit indeed several other nice characterizations (see [15] for a survey).

It can be shown that *USF* is strictly included in *SF*, and so we have the following chain of inclusions:

$$USF \subsetneq SF \subsetneq INT \subsetneq REG.$$

The following theorem, proved in [3], shows the role of unambiguity in Problem 2.

Theorem 41 *If L_1 and $L_2 \in USF$, then $L_1 \sqcup\!\sqcup L_2 \in SF$.*

5 Cyclic Submonoids and Combinatorics on Words

The languages in the family *USF* can be described by regular expressions in which the star operation is restricted to subsets of the alphabet. Furthermore, Theorem 41 states that the shuffle of languages in this family is star-free. Hence, the critical situations, with respect to Problem 2, occur with languages corresponding to regular expressions in which the star operation is applied to concatenation of letters. So, in this section, we consider the shuffle of languages of the form u^*, where u is a word of Σ^*. Actually, such languages correspond to *cyclic submonoids* of Σ^*.

The special interest of such languages in our context is shown by the following theorem, proved in [2].

Theorem 51 *If the word u contains more than one letter, then the language u^* is intermixed.*

Moreover, next theorem, firstly proved in [9], shows that the combinatorial properties of the word u play a role in Problem 2. Let us first introduce a definition. A word $u \in \Sigma^*$ is *primitive* if it is not a proper power of another word of Σ^*, i.e., if the condition $u = v^n$, for some word v and integer n, implies that $u = v$ and $n = 1$.

Theorem 52 *The language u^* is star-free if and only if u is a primitive word.*

We now consider the shuffle $u^* \sqcup\!\sqcup\, v^*$ of two cyclic submonoids generated by the words u and v, respectively. If u and v are primitive words then, by the previous theorem, u^* and v^* are star-free languages. Remark that the languages u^* and v^* do not belong to *USF*, and their shuffle, in general, is not star-free. Here we study the conditions under which the language $u^* \sqcup\!\sqcup\, v^*$ is star-free.

Let us consider some examples. If $u = b$ and $v = ab$, the language $b^* \sqcup\!\sqcup\,(ab)^* = (b + ab)^*$ is star-free. Let us consider now $u = aab$ and $v = bba$, the language $(aab)^* \sqcup\!\sqcup\, (bba)^*$ is not star-free. Indeed the language

$$((aab)^* \sqcup\!\sqcup\, (bba)^*) \cap (ab)^* = ((ab)^3)^*$$

is not star-free, by the Theorem 52

Problem 3 : *Characterize the pairs of primitive words $u, v \in \Sigma^*$ such that $u^* \sqcup\!\sqcup\, v^*$ is a star-free language.*

This last problem is closely related to some relevant questions in combinatorics on words. Recall that combinatorics on words is a fundamental part of the theory of words and languages. It is deeply connected to numerous different fields of mathematic and its applications, and it emphasizes the algorithmic nature of many problems on words (cf [7]).

Some important problems in combinatorics on words pertain to the non primitive words that appear in the set u^+v^+, where u and v are primitive words.

A remarkable result in this direction is the famous Lyndon-Schutzenberger theorem (cf [8]), originally formulated for the free groups.

Theorem 53 *If u and v are distinct primitive words, then the word $u^n v^m$ is primitive for all $n, m \geq 2$.*

The next theorem, proved by Shyr and Yu ([14]), can be considered as a light improvement of the previous result.

Theorem 54 *If u and v are distinct primitive words, then there is at most one non-primitive word in the language $u^+ v^+$.*

Problem 3 is, in a certain sense, related to those considered in the previous theorems, with the difference that we here take into account the shuffle of the two languages u^+ and v^+, instead of their concatenation. Actually, Problem 3 leads to investigate the non-primitive words that appear in the language $u^+ \sqcup\!\sqcup v^+$, where u and v are primitive words. In particular, we are interested to investigate the exponents of the powers that appear in $u^+ \sqcup\!\sqcup v^+$.

Let us introduce further notation. Let us denote by Q the set of primitive words. For $u, v, w \in Q$, let $p(u, v, w)$ be the integer k such that

$$u^* \sqcup\!\sqcup v^* \cap w^* = (w^k)^*.$$

Remark that, if $u^* \sqcup\!\sqcup v^* \cap w^* = \{\epsilon\}$, then $p(u, v, w) = 0$. For $u, v \in Q$, let us define the set of integers

$$P(u, v) = \{p(u, v, w) | \ w \in Q\}.$$

For instance, if we consider the words $u = a^{10}b$, $v = b$, then $P(u, v) = \{0, 1, 2, 5, 10\}$.

The following problem is closely related to Problem 3.

Problem 4 : *Given two primitive words u, v, characterize the set $P(u, v)$ in terms of the combinatorial properties of u and v.*

References

1. Baeten, J., Weijland, W.: *Process Algebra*, Cambridge University Press, 1990.
2. Berstel, J., Boasson, L., Carton, O., Pin, J.-E., Restivo, A.: The expressive power of the shuffle product, *Inf. Comput.*, **208**(11), 2010, 1258–1272.
3. Castiglione, G., Restivo, A.: On the Shuffle of Star-Free Languages, *Fundam. Inform.*, **116**(1-4), 2012, 35–44.
4. Diekert, V.: *The Book of Traces*, World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995, ISBN 9810220588.
5. Eilenberg, S.: *Automata, Languages, and Machines*, Academic Press, Inc., Orlando, FL, USA, 1976.
6. Guaiana, G., Restivo, A., Salemi, S.: Star-Free Trace Languages, *Theor. Comput. Sci.*, **97**(2), 1992, 301–311.
7. Lothaire, M.: *Algebraic Combinatorics on Words*, Cambridge University Press, 2002.

8. Lyndon, R. C., Schützenberger, M. P.: The equation $a^M = b^N c^P$ in a free group, *Michigan Math. J.*, **9**(4), 1962, 289–298.
9. McNaughton, R., Papert, S.: *Counter-Free Automata,* MIT Press, Cambridge, 1971.
10. Perrot, J. F.: Varietes de Langages et Operations, *Theor. Comput. Sci.*, **7**, 1978, 197–210.
11. Pin, J.-E.: *Varieties of formal languages*, North Oxford, LondonPlenum, New-York, 1986, (Traduction de Variétés de langages formels).
12. Pin, J.-E.: Syntactic semigroups, in: *Handbook of formal languages* (G. Rozenberg, A. Salomaa, Eds.), vol. 1, chapter 10, Springer, 1997, 679–746.
13. Pin, J.-E.: Theme and variations on the concatenation product, *Proceedings of the 4th international conference on Algebraic informatics*, CAI'11, Springer-Verlag, Berlin, Heidelberg, 2011, ISBN 978-3-642-21492-9, 44–64.
14. Shyr, H. J., Yu, S. S.: Non-primitive words in the Language p^+q^+, *Soochow J. Math.*, **20**(4), 1994, 535–546.
15. Tesson, P., Therien, D.: Diamonds Are Forever: The Variety DA, *Semigroups, Algorithms, Automata and Languages, Coimbra (Portugal) 2001*, World Scientific, 2002, 475–500.

Simulating EXPSPACE Turing machines using P systems with active membranes

Artiom Alhazov[1,2], Alberto Leporati[1], Giancarlo Mauri[1], Antonio E. Porreca[1], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
artiom.alhazov@unimib.it
{leporati,mauri,porreca,zandron}@disco.unimib.it
[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
artiom@math.md

1 Introduction

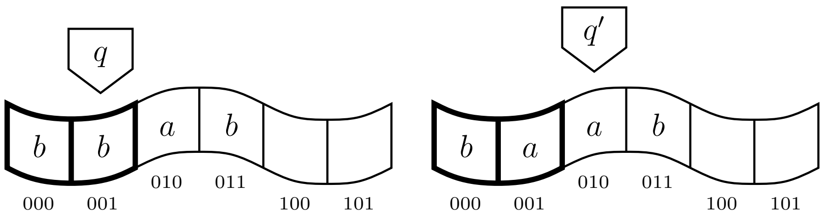
P systems with active membranes [2] are parallel computation devices inspired by the internal working of biological cells. Their main features are a hierarchy of nested membranes, partitioning the cell into regions, and *multisets* of symbol-objects describing the chemical environment. The system evolves by applying rules such as non-cooperative multiset rewriting (i.e., objects are individually rewritten), communication rules that move the objects between adjacent regions, and membrane division rules that increase the number of membranes in the system. The membranes also possess an *electrical charge* that works as a local state, regulating the set of rules applicable during each computation step. The rules, in turn, may change the charge of the membrane where they take place.

In order to solve computational problems one usually employs *polynomial-time uniform families* of P systems with active membranes, consisting of a P system Π_n for each input length n (as for Boolean circuits) and a single Turing machine constructing Π_n from n in polynomial time. The actual input is then encoded as a multiset of objects, and placed inside an input membrane of Π_n. The space required by a family of P systems (in terms of number of membranes and objects) for solving a decision problem can then be analysed as a function of n. It is already known that polynomial-space P systems and polynomial-space Turing machines are equal in computing power [3], but the proof of this result does not generalise to larger space bounds. In this paper we show the key ideas needed in order to prove the exponential-space analogue of that result by directly simulating deterministic exponential-space Turing machines using P systems.

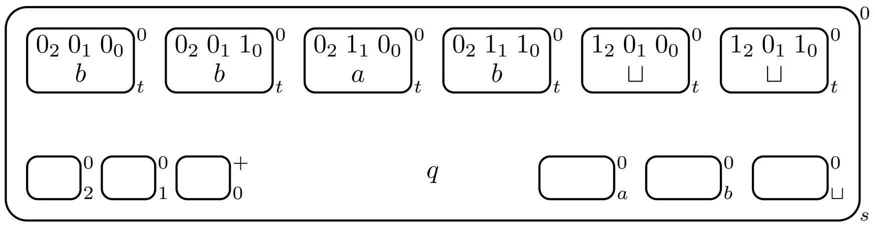
For the full technical details of the results presented here we refer the reader to the paper "The computational power of exponential-space P systems with active membranes" [1] by the same authors.

2 Simulating Turing machines

We describe how deterministic Turing machines working in exponential space can be simulated by P systems by means of an example. Let M be a Turing machine processing an input x of length $n = 2$ and requiring $2^n = 4$ auxiliary tape cells (the total length of the tape is then 6); assume that the alphabet of M consists of the symbols a and b. Suppose that the current configuration $\mathcal{C}$ of M is the one depicted on the left of the following picture, and that the transition it performs leads it to the configuration $\mathcal{C}'$ on the right. In the picture, the tape cells of M are identified by a binary index.



We encode the configuration $\mathcal{C}$ of M as the following configuration of the P system Π simulating it:



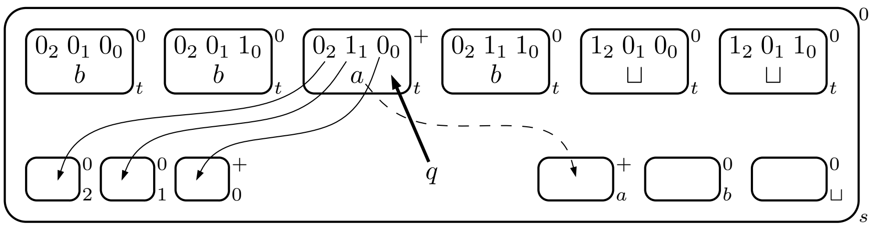
In this picture, the label of a membrane is indicated at its lower-right corner, while its electrical charge (+, 0, or −) is at its upper-right corner. The symbols located inside the membranes represent the objects in the configuration of Π.

The P system, beside its external membrane s, possesses 6 membranes labelled by t (called the *tape-membranes*) corresponding to the tape cells of M; each tape-membrane contains 3 subscripted bit-objects encoding the index of the corresponding tape cell (the subscript represents the position of the bit in the index; for instance, the presence of bit-object 1_0 indicates that 1 is the least significant bit). Furthermore, each tape-membrane contains an object representing the symbol written in the corresponding tape cell of M, where $\sqcup$ represents a blank cell. Only one tape membrane is part of the initial configuration of Π, as it can be at most polynomial in size; the other ones are created by membrane division during an initialisation phase of Π, before simulating the first step of M.

A *state-object* q represents the current state of M; this object will also regulate the simulation of the next step of M. The position of the tape head is encoded in binary as the electrical charges of the membranes $0, 1, 2$ (the *position-membranes*); the label of each membrane represents the position of the corresponding bit, while its charge the value of the bit: a neutral charge represents a 0, and a positive charge a 1. In the example above, the charges of membranes $2, 1, 0$ are $0, 0, +$, encoding the binary number 001 (decimal 1).

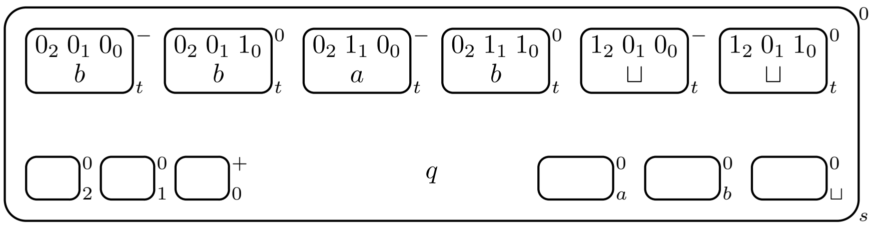
Finally, the auxiliary membranes labelled by $a, b, \sqcup$ (the *symbol-membranes*) in the lower-right corner correspond to the tape symbols of M, and are used in order to read the symbol on the current tape cell.

The following picture shows how the next step of M is simulated.

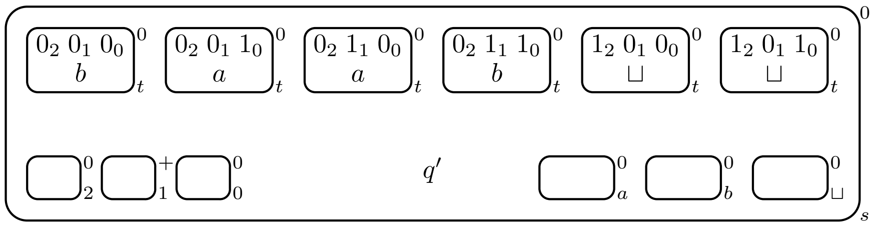


First, the object q nondeterministically guesses a tape-membrane t (all such membranes are indistinguishable from the outside) and enters it (thick arrow in the picture) while changing the charge to positive. The change of charge enables the bit-objects inside it to move to the corresponding position-membranes (along the thin arrows), where their values are compared to the charges of the membranes; this allows us to check whether the tape-membrane we guessed is indeed the one under the tape head of M. In the mean time, the object a is sent to the corresponding symbol-membrane (dashed arrow) in order to change the charge to positive.

Since in the example the tape-membrane that was chosen is not the correct one, an error-object is produced by one of the mismatched position-bits, and the configuration of Π is restored to the initial one, with the following exception: the charge of the tape-membrane is set to *negative*, so it will not be chosen again. The P system then proceeds by guessing another tape-membrane among the remaining (neutrally charged) ones. After a number of wrong guesses, the configuration of Π will be similar to the following one.



When the tape-membrane corresponding to the current cell of M is finally guessed, we can perform the actual simulation of the computation step (updating the position of the head, the symbol on the tape, and the state of M). The state-object may first read the tape symbol by looking at the only positively charged symbol-membrane; it can then update the charges of the position-membranes (from the least to the most significant bit) in order to increment or decrement the binary number they encode, produce the new tape symbol (b in the example) and finally rewrite itself as the new state of M (q' in the example). The charges of all tape- and symbol-membranes are also reset to neutral by using auxiliary objects. The configuration of Π corresponding to the new configuration of M thus becomes the following one.



Now the P system continues simulating the next steps of M, until an accepting (resp., rejecting) state is reached; when this happens, the P system produces a YES (resp., NO) object that is sent out from the outermost membrane as the result of the computation.

3 Conclusions and open problems

The simulation described in the previous section can be carried out by a polynomial time uniform family of P systems with active membranes operating in space $O\big(s(n) \log s(n)\big)$, where $s(n)$ is the space required by the simulated Turing machine on inputs of length n. Since an analogous result holds in the opposite direction [3, Theorem 5], the two classes of devices solve exactly the same decision problems when working withing an exponential space limit.

The techniques employed here do not carry over to the simulation of super-exponential space Turing machines, since they would require a super-polynomial number of subscripted objects in order to encode tape positions; this amount of objects (and their associated rules) cannot be constructed using a polynomial-time uniformity condition. Novel techniques will be probably needed in order to prove that the equivalence of Turing machines and P systems also holds for larger space bounds.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: The computational power of exponential-space P systems with active membranes. In: Martínez-del-Amor, M.A., Păun, Gh., Pérez-Hurtado, I., Romero-Campero, F.J., Valencia-Cabrera, L. (eds.) Tenth Brainstoming Week on Membrane Computing, vol. I, pp. 35–60. Fénix Editora (2012)
2. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
3. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. International Journal of Foundations of Computer Science 22(1), 65–73 (2011)

Behavioural Equivalences over Mobile Membranes with Delays

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
and "A.I.Cuza" University of Iaşi, Romania
Email: baman@iit.tuiasi.ro and gabriel@info.uaic.ro**

Abstract. Mobile membranes with delays represent a biological inspired formalism able to model systems involving timing, explicit locations and mobility. We define a number of behavioural equivalences over this formalism, and prove some relationships between these equivalences.

1 Introduction

During the last years, membrane computing [3, 6] has been applied to Biology. It may have an important impact in understanding how biological systems work, giving also a way to describe, manipulate, analyse and verify them. Behavioural equivalence is an important concept in biology needed for analysing and comparing the organs behaviour. For example, an artificial organ is the functional equivalent of the natural organ, meaning that both behave in a similar manner.

Using mobile membranes, we are interested either in locations, times of evolution, mobility objects, or in combinations of these concepts. Thus we define several equivalences, showing that some of them are finer that others, and that some of them are incomparable. Defining several equivalences, we offer flexibility in selecting the right one when verifying biological systems and comparing them.

What we do in this paper is a first step towards establishing the formal framework used in software verification for biological systems sensitive to timeouts.

2 Systems of Mobile Membranes with Delays

Systems of simple mobile membranes [4] are a particular class of membrane computing [6], while several types of mobile membranes were studied in detail in [2]. We use a rule-based model of computation called systems of mobile membranes with delays in order to model complex biological processes.

Definition 1. *A system of mobile membranes with delays is a construct* $\Pi = (O_t, H, \mu, w_1, \dots, w_n, R)$, $n \geq 1$ *(the degree of the system), where:*

1. $O_t = O \times \mathbb{N}$ *is a set of objects with delays, where* O *is an alphabet of objects, and* $(a, t_a) \in O_t$ *represents an object* a *together with its delay* $t_a \geq 0$*;*

** The work of Gabriel Ciobanu was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919, while the work of Bogdan Aman was supported by POSDRU/89/1.5/S/49944.

2. *H is a finite set of labels for membranes;*
3. *$\mu \subset H \times H$ describes the* membrane structure*, such that $(i, j) \in \mu$ denotes that the membrane labelled by j is contained in the membrane labelled by i;*
4. *$w_1, w_2, \ldots, w_n$ are multisets over O_t, describing the initial multisets of objects with delays placed in the n regions of μ;*
5. *R is a finite set of evolution rules of the following forms, where $h, m \in H$, $(a, 0), (\overline{a}, 0), (c, t_c), (b, t_b), (a, t_a), (a, t_a - 1) \in O_t$:*
   - *(a)* $[\,(a,0)]_h[\,(\overline{a},0)]_m \to [\,[\,(c,t_c)]_h(b,t_b)]_m$ *endocytosis*
     an elementary membrane h containing $(a, 0)$ enters membrane m containing $(\overline{a}, 0)$; $(a, 0)$ is rewritten to (c, t_c), and $(\overline{a}, 0)$ is rewritten to (b, t_b);
   - *(b)* $[\,[\,(a,0)]_h(\overline{a},0)]_m \to [\,(c,t_c)]_h[\,(b,t_b)]_m$ *exocytosis*
     an elementary membrane h containing $(a, 0)$ exits membrane m containing $(\overline{a}, 0)$; $(a, 0)$ and $(\overline{a}, 0)$ are rewritten to (c, t_c) and (b, t_b), respectively;
   - *(c)* $[\,(a,0)]_h \to [\,(c,t_c)]_h[\,(b,t_b)]_h$ *elementary division*
     if containing $(a, 0)$, a membrane h is divided into two membranes with the same label m, and $(a, 0)$ is rewritten to (b, t_b) and (c, t_c);
   - *(d)* $[\,(a,t_a)]_h \rightsquigarrow [\,(a,t_a-1)]_h$ *delay decrementing*
     if $t_a > 0$ then (a, t_a) placed inside membrane h is rewritten to $(a, t_a - 1)$.

In terms of computation, we are working with membrane configurations (ranged over by $M, N, \ldots$) and with the free monoid O_t^* (ranged over by $u_t, v_t, \ldots$).

Definition 2. *The set $\mathcal{M}(\Pi)$ of membrane configurations in a membrane system Π is inductively defined as follows:*

- *if $i \in H$ and u_t is a multiset over O_t then $\langle i; u_t\rangle \in \mathcal{M}(\Pi)$; $\langle i; u_t\rangle$ is called an* elementary membrane configuration*;*
- *if $i \in H$, $M_1, \ldots, M_n \in \mathcal{M}(\Pi)$, $n \geq 1$, and u_t is a multiset over O_t then $\langle i; u_t,\ M_1 \ldots M_n\rangle \in \mathcal{M}(\Pi)$; $\langle i; u_t,\ M_1 \ldots M_n\rangle$ is a* composite membrane.

Definition 3. *For a membrane system Π, if $M, N \in \mathcal{M}(\Pi)$ then:*

- *M* reduces *to N (denoted by $M \mapsto N$) if there exists a rule in R, applicable to membrane M such that we can obtain membrane N. We use $\mapsto$ to stand for both $\to$ and $\rightsquigarrow$. We denote by $\rightsquigarrow^n$ a sequence of $n \geq 1$ reductions $\rightsquigarrow$.*

3 Behavioural Equivalences

These equivalence relationships are useful when checking the "healthiness" of a system. For example, take two healthy systems M and N that are in a relationship of barbed bisimulation. If they are both infected with a virus and evolved into M' and N', through the barbed bisimulation, it is easy to check if they are infected with a virus of the same kind (each virus has an unique behaviour and is activated by an unique trigger).

In what follows, in order to distinguish between normal and abnormal behaviours, we define various barbed bisimulation as in [5], and specify first what is observable. To emphasize the mobility aspects, the objects involved in endocytosis and exocytosis rules are observable.

To avoid ambiguity, we consider that the objects involved in endocytosis and exocytosis rules belong to the sets of objects O_{exo} and O_{endo}, respectively, such that $O_{exo} \in O_t$, $O_{endo} \in O_t$ and $O_{endo} \cap O_{exo} = \emptyset$. In what follows, let $x \in \{endo, exo\}$ represent the possible movements, u'_t, u''_t arbitrary multisets of objects with delays, N, M, M' configurations, and $m \in H$ a membrane label.

A barb predicate $\downarrow_{x(a)}$ ($\downarrow_{x(a)@m}$, $\downarrow^{t_a}_{x(a)}$, $\downarrow^{t_a}_{x(a)@m}$) is defined by the rule:

$$M \downarrow_{x(a)} \ (M \downarrow_{x(a)@m},\ M \downarrow^{t_a}_{x(a)},\ M \downarrow^{t_a}_{x(a)@m}, \text{ respectively})$$
$$\text{iff } M \equiv \langle m; (a, t_a) \uplus u'_t, N\rangle \parallel M', \text{ where } a \in O_x.$$

Definition 4. *A barbed bisimulation $\mathcal{S}$ in terms of mobility is a symmetric binary relation over membrane configurations such that for all $(M, N) \in \mathcal{S}$, $n \in \mathbb{N}$*

1. *if $M \downarrow_{x(a)}$, then $N \downarrow_{x(a)}$ for any barb predicate $\downarrow_{x(a)}$;*
2. *if $M \rightsquigarrow^n\rightarrow M'$, then exists N' such that $N \rightsquigarrow^n\rightarrow N'$ and $(M', N') \in \mathcal{S}$.*

Two membrane configurations are barbed bisimilar, in terms of mobility, denoted $M \sim_{mob} N$, if and only if $(M, N) \in \mathcal{S}$ for some barbed bisimulation $\mathcal{S}$.

It is rather natural to strengthen the observing power of the previous defined barbs by allowing the observer to look also at the label (location) of the membrane containing the object that facilitates a movement.

The barbed bisimulation $\sim_{Lmob}$, in terms of location and mobility, is defined similarly with the barbed bisimulation $\sim_{mob}$, by using the barb predicate $\downarrow_{x(a)@m}$. This bisimulation compares membrane configurations by looking also at the label of the membrane containing an object that facilitates a movement.

Bisimulation relations are represented and studied as sets of pairs of membrane configurations. Thus the comparison between bisimilarities are set-theoretic.

Proposition 1 **($\sim_{mob} \prec \sim_{Lmob}$).** *The located barbed bisimulation is strictly finer than the barbed bisimulation:*

1. *$\sim_{mob} \preceq \sim_{Lmob} \Leftrightarrow \forall M, N$, if $M \sim_{Lmob} N$ then $M \sim_{mob} N$;*
2. *$\sim_{Lmob} \npreceq \sim_{mob} \Leftrightarrow \exists M, N$, s.t. if $M \sim_{mob} N$ then $M \not\sim_{Lmob} N$.*

Proof (Sketch).

1. The located observer (i.e., the located barb) can distinguish in both membrane configurations the same object a placed inside the same membrane m facilitating a movement, and so the located barb implies the simple barb ($M \downarrow_{x(a)@m}$ implies $M \downarrow_{x(a)}$).
2. We give the following counterexample: Take two membrane configurations M and N, and an object $\overline{a} \in O_{exo}$ s.t. $M = \langle l; (\overline{a}, t_{\overline{a}}) \uplus u'_t\rangle$ and $N = \langle k; (\overline{a}, t_{\overline{a}}) \uplus u'_t\rangle$ with $l \neq k$. Both $M \downarrow_{exo(\overline{a})}$ and $N \downarrow_{exo(\overline{a})}$ hold, and thus the two membrane configurations are barbed bisimilar: $M \sim_{mob} N$. However $M \downarrow_{exo(\overline{a})@l}$ and $N \downarrow_{exo(\overline{a})@k}$ also hold, and $l \neq k$; therefore $M \not\sim_{Lmob} N$. □

The bisimulation $\sim_{Dmob}$ is defined similarly with the bisimulation $\sim_{mob}$, by using the barb predicate $\downarrow^{t_a}_{x(a)}$. It relates membrane configurations with the same objects that execute the same movements and have the same delays.

Proposition 2 **($\sim_{mob} \prec \sim_{Dmob}$).** *The delayed barbed bisimulation is strictly finer than the barbed bisimulation:*
1. $\sim_{mob} \preceq \sim_{Dmob} \Leftrightarrow \forall M, N,$ *if* $M \sim_{Dmob} N$ *then* $M \sim_{mob} N$*;*
2. $\sim_{Dmob} \not\preceq \sim_{mob} \Leftrightarrow \exists M, N,$ *s.t. if* $M \sim_{mob} N$ *then* $M \not\sim_{Dmob} N$*.*

The bisimulation $\sim_{DLmob}$ is defined similarly with the bisimulation $\sim_{Dmob}$, by using the barb predicate $\downarrow^{t_a}_{x(a)@m}$. It relates membrane configurations with the same objects located in the same membranes that execute the same movements and have the same delays.

Proposition 3 **($\sim_{Lmob} \prec \sim_{DLmob}$).** *The delayed located barbed bisimulation is strictly finer than the located barbed bisimulation:*
1. $\sim_{Lmob} \preceq \sim_{DLmob} \Leftrightarrow \forall M, N,$ *if* $M \sim_{DLmob} N$ *then* $M \sim_{Lmob} N$*;*
2. $\sim_{DLmob} \not\preceq \sim_{Lmob} \Leftrightarrow \exists M, N,$ *s.t. if* $M \sim_{Lmob} N$ *then* $M \not\sim_{DLmob} N$*.*

The four barbed bisimulations form a lattice in which a directed edge means "is strictly finer":

$$\begin{tikzcd}
& \sim_{DLmob} \arrow[dl] \arrow[dr] \arrow[dd] & \\
\sim_{Dmob} \arrow[dr] & & \sim_{Lmob} \arrow[dl] \\
& \sim_{mob} &
\end{tikzcd}$$

4 Conclusion

A small difference in the behaviour of a biological system could lead to a disease. Such a difference could appear because of the involved elements, their location, their actions and timing. The behavioural equivalences introduced in this paper could make the distinction between "normal" and "abnormal" behaviours, emphasizing also the elements by which behaviours differ. During the presentation, some biological examples will illustrate the use of these bisimulations.

As future work, we are interested in theoretical investigation of other behavioural equivalences and their applicability to Systems Biology. Other behavioural equivalences (other than bisimulations) can also be considered: trace equivalences, barbed congruences and testing equivalences.

References

1. Aman, B., and Ciobanu, G. 2009. Turing Completeness Using Three Mobile Membranes. *Lecture Notes in Computer Science* 5715, 42–55.
2. Aman, B., and Ciobanu, G. 2009. Simple, Enhanced and Mutual Mobile Membranes. *Transactions on Computational Systems Biology XI* 5750, 26–44.
3. Ciobanu, G., Păun, Gh., and Pérez-Jiménez, M.J., eds. 2006. *Applications of Membrane Computing.* Springer.
4. Krishna, S.N., and Păun, Gh. 2005. P Systems with Mobile Membranes. *Natural Computing* 4, 255–274.
5. Milner, R., and Sangiorgi, D. 1992. Barbed Bisimulation. *Lecture Notes in Computer Science* 623, 685–695.
6. Păun, Gh., Rozenberg, G., and Salomaa, A., eds. 2010. *The Oxford Handbook of Membrane Computing.* Oxford University Press.

Global Types for Dynamic Checking of Protocol Conformance of Multi-Agent Systems
(Extended Abstract)

Davide Ancona, Matteo Barbieri, and Viviana Mascardi

DIBRIS, University of Genova, Italy
email: davide@disi.unige.it, matteo.barbieri@oniriclabs.com,
mascardi@disi.unige.it

1 Introduction

Multi-agent systems (MASs) have been proved to be an industrial-strength technology for integrating and coordinating heterogeneous systems. However, due to their intrinsically distributed nature, testing MASs is a difficult task. In recent work [1] we have tackled the problem of run-time verification of the conformance of a MAS implementation to a specified protocol by exploiting global types on top of the Jason agent oriented programming language [2].

Global types [3,6,4] are a behavioral type and process algebra approach to the problem of specifying and verifying multiparty interactions between distributed components.

Our notion of global type closely resembles that of Castagna, Dezani, and Padovani, [4] except for two main differences: our types are interpreted coinductively, rather than inductively, hence they are possibly infinite sets of possibly infinite sequences of interactions between a fixed set of participants; in this way, protocols that must not terminate can be specified. Furthermore, we use global types for dynamic, rather than static, checking of multiparty interactions; errors can only be detected at run-time, but checking is simpler and more flexible, and no notion of projection and session type has to be introduced.

Global types can be naturally represented as cyclic Prolog terms (that is, regular terms), and their interpretation can be given by a transition function, that can be compactly defined by a Prolog predicate. With such a predicate, a Jason monitor agent can be automatically implemented to dynamically check that the message exchange between the agents of a system conforms to a specified protocol.

In this paper we continue our research in two directions: on the one hand, we investigate the theoretical foundations of our framework; on the other, we extend it by introducing a concatenation operator that allows a significant enhancement of the expressive power of our global types. As significant examples, we show how two non trivial protocols can be compactly represented in our framework: a ping-pong protocol, and an alternating bit protocol, in the version proposed by Deniélou and Yoshida [5]. Both protocols cannot be specified easily (if at all) by other global type frameworks, while in our approach they can be expressed by two deterministic types (in a sense made precise in the sequel) that can be effectively employed for dynamic checking of the conformance to the protocol.

2 Global type interpretation

A global type τ represents a set of possibly infinite sequences of sending actions, and is defined on top of the following type constructors:

- λ (empty sequence), representing the singleton set $\{\epsilon\}$ containing the empty sequence ϵ.
- $a{:}\tau$ (*seq*), representing the set of all sequences obtained by adding the sending action a at the beginning of any sequence in τ.
- $\tau_1 + \tau_2$ (*choice*), representing the union of the sequences of τ_1 and τ_2.
- $\tau_1|\tau_2$ (*fork*), representing the set obtained by shuffling the sequences in τ_1 with the sequences in τ_2 .
- $\tau_1 \cdot \tau_2$ (*concat*), representing the set of sequences obtained by concatenating any sequence of τ_1 with any sequence of τ_2.

As an example, $(((a_1{:}\lambda)|(a_2{:}\lambda)) + ((a_3{:}\lambda)|(a_4{:}\lambda))) \cdot ((a_5{:}a_6{:}\lambda)|(a_7{:}\lambda))$ denotes the set of message sequences

$$\left\{\begin{array}{l} a_1a_2a_5a_6a_7, a_1a_2a_5a_7a_6, a_1a_2a_7a_5a_6, a_2a_1a_5a_6a_7, a_2a_1a_5a_7a_6, a_2a_1a_7a_5a_6, \\ a_3a_4a_5a_6a_7, a_3a_4a_5a_7a_6, a_3a_4a_7a_5a_6, a_4a_3a_5a_6a_7, a_4a_3a_5a_7a_6, a_4a_3a_7a_5a_6 \end{array}\right\}$$

Global types are regular terms, that is, can be cyclic: more abstractly, they are finitely branching trees (where nodes are type constructors) whose depth can be infinite, but that can only have a finite set of subtrees. A regular term can be represented by a finite set of syntactic equations, as happens, for instance, in Jason and in most modern Prolog implementations. For instance, the two equations $T_1 = (\lambda + (a_1{:}T_1)) \cdot T_2$, and $T_2 = (\lambda + (a_2{:}T_2))$ represent the following infinite, but regular, global types $(\lambda + (a_1{:}(\lambda + (a_1{:}\ldots)))) \cdot (\lambda + (a_2{:}(\lambda + (a_2{:}\ldots))))$ and $(\lambda + (a_2{:}(\lambda + (a_2{:}\ldots))))$, respectively.

To ensure termination of dynamic checking of protocol conformance, we only consider *contractive* (or *guarded*) types.

Definition 1. *A global type τ is* contractive *if it does not contain paths whose nodes can only be constructors in $\{+, |, \cdot\}$ (such paths are necessarily infinite).*

The type represented by the equation $T_1 = (\lambda + (a_2{:}T_1))$ is contractive: its infinite path contains infinite occurrences of +, but also of the : constructor; conversely, the type represented by the equation $T_2 = (\lambda + ((T_2|T_2) + (T_2 \cdot T_2)))$ is not contractive. Trivially, every finite type (that is, non cyclic) is contractive.

The interpretation of a global type depends on the notion of transition, a total function $\delta{:}\mathcal{T} \times \mathcal{A} \rightarrow \mathcal{P}_{fin}(\mathcal{T})$, where $\mathcal{T}$ and $\mathcal{A}$ denote the set of contractive global types and of sending actions, respectively. As it is customary, we write $\tau_1 \xrightarrow{a} \tau_2$ to mean $\tau_2 \in \delta(\tau_1, a)$. Figure 1 (in the Appendix) defines the inductive rules for the transition function.

The auxiliary function ϵ, inductively defined in Figure 2 (in the Appendix), specifies the global types whose interpretation is equivalent to λ.

Proposition 1. *Let τ be a contractive type. Then $\tau \xrightarrow{a} \tau'$ for some a and τ' if and only if $\epsilon(\tau)$ does not hold.*

Note that the proposition above does not hold if we drop the hypothesis requiring τ to be contractive; for instance, if τ is defined by $T = T + T$, then neither $\epsilon(\tau)$ holds, nor there exist a, τ' s.t. $\tau \xrightarrow{a} \tau'$.

Proposition 2. *If τ is contractive and $\tau \xrightarrow{a} \tau'$ for some a, then τ' is contractive as well.*

The two propositions above ensures termination when the rules defined in Figures 1 and 2 are turned into an algorithm (implemented, for instance, in Prolog clauses, as done for Jason [1]).

Definition 2. *Let τ_0 be a contractive type. A* run *ρ for τ_0 is a sequence $\tau_0 \xrightarrow{a_0} \tau_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} \tau_n \xrightarrow{a_n} \tau_{n+1} \xrightarrow{a_{n+1}} \ldots$ such that*

- *either the sequence is infinite, or there exists k such that $\epsilon(\tau_k)$;*
- *for all τ_i, a_i, and τ_{i+1} in the sequence, $\tau_i \xrightarrow{a_i} \tau_{i+1}$ holds.*

We denote by $\alpha(\rho)$ the possibly infinite sequence of sending actions $a_0 a_1 \ldots a_n \ldots$ contained in ρ.

The interpretation $[\![\tau_0]\!]$ of τ_0 is the set $\{\alpha(\rho) \mid \rho$ is a run for τ_0 $\}$ if τ_0 admits at least one run, $\{\epsilon\}$ otherwise.

Note that, differently from other approaches [4], global types are interpreted coinductively: for instance, the global type defined by $T = a{:}T$ denotes the set $\{a^\omega\}$ (that is, the singleton set containing the infinite sequence of sending action a), and not the empty set. Furthermore, whereas global types are regular trees, in general their interpretation is not a regular language, since it may contain strings of infinite length.

Finally, we introduce the notion of deterministic global type, which ensures that dynamic checking can be performed efficiently without backtracking.

Definition 3. *A contractive global type τ is deterministic if for any possible run ρ of τ and any possible τ' in ρ, if $\tau' \xrightarrow{a} \tau''$, $\tau' \xrightarrow{a'} \tau'''$, and $a = a'$, then $\tau'' = \tau'''$.*

3 Examples

In this section we provide two examples to show the expressive power of our formalism.

3.1 Ping-pong Protocol

This protocol requires that first Alice sends n (with $n \geq 1$, but also possibly infinite) consecutive ping messages to Bob, and then Bob replies with exactly

n pong messages. The conversation continues forever in this way, but at each iteration Alice is allowed to change the number of sent ping messages.

For simplicity we encode with *ping* and *pong* the only two possible sending actions; then, the protocol can be specified by the following contractive and deterministic global type (defined by the variable *Forever*):

$$\begin{aligned}&\mathit{Forever} = \mathit{PingPong} \cdot \mathit{Forever}\\&\mathit{PingPong} = \mathit{ping}{:}((\mathit{pong}{:}\lambda) + ((\mathit{PingPong}) \cdot (\mathit{pong}{:}\lambda)))\end{aligned}$$

3.2 Alternating Bit Protocol

We consider the Alternating Bit protocol, in the version defined by Deniélou and Yoshida [5]. Four different sending actions may occur: Alice sends msg1 to Bob (sending action $\mathit{msg_1}$), Alice sends msg2 to Bob (sending action $\mathit{msg_2}$), Bob sends ack1 to Alice (sending action $\mathit{ack_1}$), Bob sends ack2 to Alice (sending action $\mathit{ack_2}$). Also in this case the protocol is an infinite iteration, but the following constraints have to be satisfied for all occurrences of the sending actions:

- The n-th occurrence of $\mathit{msg_1}$ must precede the n-th occurrence of $\mathit{msg_2}$.
- The n-th occurrence of $\mathit{msg_1}$ must precede the n-th occurrence of $\mathit{ack_1}$, which, in turn, must precede the $(n+1)$-th occurrence of $\mathit{msg_1}$.
- The n-th occurrence of $\mathit{msg_2}$ must precede the n-th occurrence of $\mathit{ack_2}$, which, in turn, must precede the $(n+1)$-th occurrence of $\mathit{msg_2}$.

We first show a non deterministic contractive type specifying such a protocol (defined by the variable $\mathit{AltBit_1}$).

$$\begin{aligned}&\mathit{AltBit_1} = \mathit{msg_1}{:}M_2\\&\mathit{AltBit_2} = \mathit{msg_2}{:}M_1\\&M_1 = (((\mathit{msg_1}{:}\lambda)|(\mathit{ack_2}{:}\lambda)) \cdot M_2) + (((\mathit{msg_1}{:}\mathit{ack_1}{:}\lambda)|(\mathit{ack_2}{:}\lambda)) \cdot \mathit{AltBit_2})\\&M_2 = (((\mathit{msg_2}{:}\lambda)|(\mathit{ack_1}{:}\lambda)) \cdot M_1) + (((\mathit{msg_2}{:}\mathit{ack_2}{:}\lambda)|(\mathit{ack_1}{:}\lambda)) \cdot \mathit{AltBit_1})\end{aligned}$$

Since the type is not deterministic, it would require backtracking to perform the dynamic checking of the protocol. The corresponding minimal deterministic type (defined by the variable $\mathit{AltBit_1}$) is the following:

$$\begin{aligned}&\mathit{AltBit_1} = \mathit{msg_1}{:}M_2\\&\mathit{AltBit_2} = \mathit{msg_2} : M_1\\&M_1 = (\mathit{msg_1} : A_2) + (\mathit{ack_2} : \mathit{AltBit_1})\\&A_1 = (\mathit{ack_1} : M_1) + (\mathit{ack_2} : \mathit{ack_1} : \mathit{AltBit_1})\\&M_2 = (\mathit{msg_2} : A_1) + (\mathit{ack_1} : \mathit{AltBit_2})\\&A_2 = (\mathit{ack_2} : M_2) + (\mathit{ack_1} : \mathit{ack_2} : \mathit{AltBit_2})\end{aligned}$$

References

1. D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic Generation of Self-Monitoring MASs from Multiparty Global Session Types in Jason. In *Declarative Agent Languages and Technologies (DALT 2012). Workshop Notes.*, pages 1–17, 2012.

2. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason.* John Wiley & Sons, 2007.
3. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP'07 (part of ETAPS 2007)*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.
4. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multiparty session. *Logical Methods in Computer Science*, 8(1), 2012.
5. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP'12 (part of ETAPS 2012)*, LNCS. Springer, 2012.
6. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL 2008*, pages 273–284. ACM, 2008.

A Appendix

$$\text{(seq)}\frac{}{a{:}\tau \xrightarrow{a} \tau} \qquad \text{(choice-l)}\frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1 + \tau_2 \xrightarrow{a} \tau_1'} \qquad \text{(choice-r)}\frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1 + \tau_2 \xrightarrow{a} \tau_2'}$$

$$\text{(fork-l)}\frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1|\tau_2 \xrightarrow{a} \tau_1'|\tau_2} \qquad \text{(fork-r)}\frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1|\tau_2 \xrightarrow{a} \tau_1|\tau_2'}$$

$$\text{(cat-l)}\frac{\tau_1 \xrightarrow{a} \tau_1'}{\tau_1 \cdot \tau_2 \xrightarrow{a} \tau_1' \cdot \tau_2} \qquad \text{(cat-r)}\frac{\tau_2 \xrightarrow{a} \tau_2'}{\tau_1 \cdot \tau_2 \xrightarrow{a} \tau_2'}\ \epsilon(\tau_1)$$

Fig. 1. Rules defining the transition function

$$(\epsilon\text{-seq})\frac{}{\epsilon(\lambda)} \qquad (\epsilon\text{-choice})\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1 + \tau_2)} \qquad (\epsilon\text{-fork})\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1|\tau_2)} \qquad (\epsilon\text{-cat})\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1 \cdot \tau_2)}$$

Fig. 2. Rules defining global types equivalent to λ

Circular causality in event structures

Massimo Bartoletti[1], Tiziana Cimoli[1], G. Michele Pinna[1], and Roberto Zunino[2]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, Italy
[2] DISI, Università di Trento and COSBI, Italy

Abstract. We propose a model of events with circular causality, in the form of a conservative extension of Winskel's event structures. We show a correspondence between the configurations in our event structures, and the proofs of a fragment of Propositional Contract Logic.

1 Introduction

Circular reasoning often appears in the compositional modelling and verification of concurrent systems [1, 2, 6, 7]. Circularity is also a common situation when reasoning about contracts [4]. A task may depend on others which have already been executed (dependencies in the past), but also on behalf that some other tasks will be performed in the future. Circularity arises when two or more tasks mutually rely on the guarantees provided by each other (circular dependencies).

Event structures (ES) are one of the classical model for concurrency, since [8]. Notwithstanding the variety of ingredients appeared in the literature, ES are at least equipped with a relation (usually written $\vdash$) modelling *causality*, and another one modeling *conflicts* (or consistency). Extensions to ES often use other relations to model other kind of dependencies, *e.g.* or-causality [3]. ES can provide a basic semantic model for assume/guarantee rules, by interpreting the enabling $b \vdash a$ as: "I will do a *after* you have done b".

However, circularity is usually prohibited in ES, either at the syntactic level, like in Winskel's prime event structures, or at the semantic level, like in Boudol's flow event structures [5]. Indeed, the classical notion of causality among events only captures dependencies in the past, but not of the other kind. For instance, in the ES with enablings $b \vdash a$ and $a \vdash b$, none of the events a and b is reachable, because of the circularity of the constraints.

We propose an extension of Winskel's event structures with a new *circular causality* relation ($\Vdash$). The ES prescribing $a \Vdash b$ (intuitively, "I will do a if you *promise* to do b") together with the other prescription $b \Vdash a$ has a configuration where both a and b have happened, despite of the circular dependencies. The configurations of these new ES do still enjoy the finiteness and finite-completeness properties of classical ES, thought they are not coincidence-free, which is correct from our point of view because of the presence of circular dependencies.

Our main technical result is an encoding of ES with circular causality into a fragment of Propositional Contract Logic PCL [4], through which we show that the problem of deciding if a set of events is a configuration can be reduced to provability in the logic (which is shown in [4] to be decidable).

2 Event structures with circular causality

In Def. 1 below we present our extension to Winskel's ES [8], to which we refer for the details about ES. We assume an irreflexive and symmetric conflict relation $\#$ on events, and for a set of events X, we define the predicate $CF(X)$ as follows: $CF(X) \triangleq (\forall e, e' \in X : \neg(e \# e'))$.

Definition 1. *An event structure with circular causality (*CES*) is a quadruple* $\mathcal{E} = (E, \#, \vdash, \Vdash)$ *where:* (i) E *is a set of events,* (ii) $\# \subseteq E \times E$ *is an irreflexive and symmetric relation, called* conflict *relation,* (iii) $\vdash \subseteq \wp_{fin}(E) \times E$ *is the* enabling *relation,* (iv) $\Vdash \subseteq \wp_{fin}(E) \times E$ *is the* circular enabling *relation. The relations* $\vdash$ *and* $\Vdash$ *are* saturated*, i.e. for all* $X, Y \subseteq_{fin} E$ *and for* $\circ \in \{\vdash, \Vdash\}$, $X \circ e \;\wedge\; X \subseteq Y \;\wedge\; CF(Y) \implies Y \circ e$.

A configuration C is a "snapshot" of the behaviour of the system modeled by an ES, where for each event $e \in C$ it is possible to find a finite justification, *i.e.* a sequence of events containing all the causes of e. We refine the notion in [8] to deal with circular causality. Intuitively, for all events e_i in the sequence $\langle e_0 \ldots e_n \rangle$, either e_i is $\vdash$-enabled by its predecessors, or it is $\Vdash$-enabled by the *whole* sequence. Note that, differently from other event-based models, if C is a configuration, not necessarily a subset of C is a configuration as well (see *e.g.*, $\mathcal{E}_1$ in Fig. 1). This makes it difficult to reason compositionally about configurations, and this is why the notion in Def. 2 below is a little more general than what suggested by our intuition. In an X*-configuration* C, the set C can contain an event e even in the absence of a justification of e through a standard/circular enabling — provided that e belongs to X. This allows, given an X-configuration, to add/remove any event and obtain an Y-configuration, possibly with $Y \neq X$.

Definition 2 (Configuration). *Let* $\mathcal{E} = (E, \#, \vdash, \Vdash)$ *be a* CES*. For all* $C, X \subseteq E$ *we say that* C *is an* X-configuration *of* $\mathcal{E}$ *iff* $CF(C)$ *and:*

$$\begin{aligned} &\forall e \in C.\ \exists e_0, \ldots, e_n \in C.\ e \in \{e_0, \ldots, e_n\} \;\wedge \\ &\qquad \forall i \leq n.\ (e_i \in X \;\vee\; \{e_0, \ldots, e_{i-1}\} \vdash e_i \;\vee\; \{e_0, \ldots, e_n\} \Vdash e_i) \end{aligned}$$

The set of all X*-configurations of* $\mathcal{E}$ *is denoted by* $\mathcal{F}_{\mathcal{E}}(X)$*, or just* $\mathcal{F}_{\mathcal{E}}$ *when* $X = \emptyset$.

Example 1. Consider the four CES in Fig. 1.

- $\mathcal{E}_0$ has enablings $\emptyset \vdash a$, $\emptyset \Vdash b$, and conflict $a \# b$. By Def. 2 we have $\emptyset, \{a\}, \{b\} \in \mathcal{F}_{\mathcal{E}_0}$, but $\{a, b\} \notin \mathcal{F}_{\mathcal{E}_0}$.
- $\mathcal{E}_1$ has enablings $\{a\} \vdash b$ and $\{b\} \Vdash a$. Here $\emptyset, \{a, b\} \in \mathcal{F}_{\mathcal{E}_1}$, $\{b\} \in \mathcal{F}_{\mathcal{E}_1}(\{b\})$ and $\{a\} \in \mathcal{F}_{\mathcal{E}_1}(\{a\})$, while neither $\{a\}$ nor $\{b\}$ belong to $\mathcal{F}_{\mathcal{E}_1}(\emptyset)$.
- $\mathcal{E}_2$ has enablings $\{a, b\} \vdash c$, $\{c\} \Vdash a$, and $\{c\} \Vdash b$. The only non-empty configuration of $\mathcal{E}_2$ is $\{a, b, c\}$.
- $\mathcal{E}_3$ has enablings $\{a, b\} \Vdash c$, $\{a, b\} \Vdash d$, $\{c\} \vdash a$, and $\{d\} \vdash b$. We have that $\{a, b, c, d\} \in \mathcal{F}_{\mathcal{E}_3}$. Note that, were one (or both) of the $\Vdash$ turned into a $\vdash$, then the only $\emptyset$-configuration would have been the empty one.

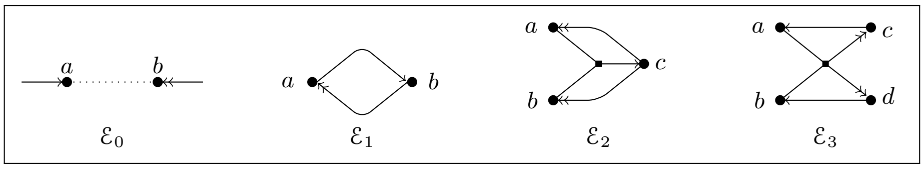


Fig. 1. CES are depicted as directed hypergraphs, where nodes stand for events. An hyperedge from a set of nodes X to node e denotes an enabling $X \circ e$, where $\circ = \vdash$ if the edge has a single arrow, while $\circ = \Vdash$ if the edge has a double arrow. A conflict $a\#b$ is represented by a dotted line between a and b.

Let $\mathcal{E} = (E, \#, \vdash, \Vdash)$ be a CES. The following properties hold.

Property 1. For all $C, C', X, Y \subseteq E$ such that $CF(C \cup C')$: (a) $C \in \mathcal{F}(C)$; (b) $X \subseteq Y \implies \mathcal{F}(X) \subseteq \mathcal{F}(Y)$; (c) $C \in \mathcal{F}(X) \land C' \in \mathcal{F}(X) \implies C \cup C' \in \mathcal{F}(X)$.

For $\mathcal{A} \subseteq \mathcal{F}(X)$, with $\mathcal{A} \uparrow$ we indicate that there exists $C' \in \mathcal{F}(X)$ such that for all $C \in \mathcal{A}$, $C \subseteq C'$. We say that $\mathcal{A}$ is *finitely compatible*, and write $\mathcal{A} \uparrow^{\mathit{fin}}$, iff $\forall \mathcal{A}_0 \subseteq_{\mathit{fin}} \mathcal{A}.\ \mathcal{A}_0 \uparrow$.

Property 2 (Finite-completeness). $\mathcal{A} \subseteq \mathcal{F}_{\mathcal{E}}(X) \land \mathcal{A} \uparrow^{\mathit{fin}} \implies \bigcup \mathcal{A} \in \mathcal{F}_{\mathcal{E}}(X)$.

Property 3 (Finiteness). $e \in C \in \mathcal{F}(X) \implies \exists C_0 \in \mathcal{F}(X).\ e \in C_0 \land C_0 \subseteq_{\mathit{fin}} C$.

Note that CES do not enjoy *coincidence-freeness*, *i.e.*, it is not always true that:

$$\forall C \in \mathcal{F}.\ \forall e, e' \in C.\ \big(e \neq e' \implies (\exists C' \in \mathcal{F}.\ C' \subseteq C \land (e \in C' \iff e' \notin C')\big)$$

A counterexample to coincidence-freeness is $\mathcal{E}_1$ in Fig. 1, where $\{a, b\} \in \mathcal{F}_{\mathcal{E}_1}$, but there exists no configuration including only a or b. Indeed, the absence of coincidence-freeness is a peculiar aspect of circularity: if two events are circularly dependent, then each configuration that contains one of them must contain both.

3 Relation with logics

We define an encoding of (finite) CES into Propositional Contract Logic (PCL, [4]), an extension of intuitionistic logic which allows for circular reasoning through a “contractual implication” connective, written $\twoheadrightarrow$. The Hilbert-style axiomatisation of PCL extend that of IPC with the following axioms:

$$\top \twoheadrightarrow \top \qquad (\phi \twoheadrightarrow \phi) \to \phi \qquad (\phi' \to \phi) \to (\phi \twoheadrightarrow \psi) \to (\psi \to \psi') \to (\phi' \twoheadrightarrow \psi')$$

In [4] a proof system is given which enjoys cut elimination and the subformula property; these imply the decidability of the entailment relation $\vdash_{\mathrm{PCL}}$.

The following property relates CES to PCL. Items (a), (b) and (c) are the CES counterpart, respectively, of PCL Gentzen rules [Cut], [$\to$L] and [Fix].

Property 4. For all $C, C', X, Y \subseteq E$ such that $CF(C \cup C')$:

(a) $C \in \mathcal{F}(X) \ \wedge \ C' \in \mathcal{F}(X \cup C) \implies C \cup C' \in \mathcal{F}(X)$
(b) $C \in \mathcal{F}(X) \ \wedge \ C' \in \mathcal{F}(X \cup Y) \ \wedge \ C \vdash Y \implies C \cup C' \in \mathcal{F}(X)$
(c) $C \in \mathcal{F}(X \cup C') \ \wedge \ C' \in \mathcal{F}(X \cup Y) \ \wedge \ C \Vdash Y \implies C \cup C' \in \mathcal{F}(X)$

In Def. 3 we show a translation from CES into PCL formulae. In particular, our mapping is a bijection into the fragment of PCL which comprises atoms, conjunctions, and non-nested (standard/contractual) implications. For an event structure $\mathcal{E} = \langle E, \#, \vdash, \Vdash \rangle$, we denote with $!E$ the set of events $\{!e \mid e \in E\}$, and we assume $!E$ disjoint from E. For each event in $e \in E \cup !E$, we assume an atom e in the logic. For a set $X \subseteq E$, we write $!X$ for the formula $\bigwedge_{e \in X} !e$.

Definition 3. *Let $\mathcal{E} = \langle E, \#, \vdash, \Vdash \rangle$ be a finite* CES. *The mapping $[\cdot]$ from $\mathcal{E}$ into* PCL *formulae is defined as follows:*

$$[(X_i \circ e_i)_i] = \bigwedge_i [X_i \circ e_i] \qquad [a \# b] = (!a \ \wedge \ !b) \to \bot$$
$$[X \circ e] = \big(!e \ \wedge \ X \ \wedge \ !X\big)[\circ]\, e \qquad \textit{where}\ [\circ] = \begin{cases} \to & \textit{if}\ \circ = \vdash \\ \twoheadrightarrow & \textit{if}\ \circ = \Vdash \end{cases}$$

Theorem 1. *Let $\mathcal{E}$ be a finite* CES. *Then, for all $C \subseteq E$:*

$$C \in \mathcal{F}_{\mathcal{E}}(X) \iff [\mathcal{E}],\, !C,\, X \vdash_{\mathrm{PCL}} C \ \textit{ and } \ [\mathcal{E}],\, !C \nvdash_{\mathrm{PCL}} \bot$$

Example 2. Consider the CES $\mathcal{E}_2$ from Fig. 1. We have that:

$$[\mathcal{E}_2] = \big((!c \wedge !a \wedge !b \wedge a \wedge b) \to c\big) \ \wedge \ \big((!a \wedge !c \wedge c) \twoheadrightarrow a\big) \ \wedge \ \big((!b \wedge !c \wedge c) \twoheadrightarrow b\big)$$

Let $C = \{a, b, c\}$. We have that $C \in \mathcal{F}_{\mathcal{E}_2}$, and $[\mathcal{E}_2],\, !C \vdash_{\mathrm{PCL}} C$. Note that, were the !-ed atoms omitted in the premises of $\to$ / $\twoheadrightarrow$, then we would have, *e.g.*, $[\mathcal{E}_2],\, !a,\, !c \vdash_{\mathrm{PCL}} a \wedge c$, from which by Theorem 1 we would have incorrectly deduced that $\{a, c\} \in \mathcal{F}_{\mathcal{E}_2}$.

Acknowledgments. Work partially supported by Aut. Region of Sardinia under grants L.R.7/2007 CRP2-120 (TESLA), CRP-17285 (TRICS) and P.I.A. 2010 Project “Social Glue”.

References

1. M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1), 1993.
2. M. Abadi and G. D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1), 1993.
3. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Inf. Comput.*, 171(1):1–49, 2001.
4. M. Bartoletti and R. Zunino. A calculus of contracting processes. In *LICS*, 2010.
5. G. Boudol. Flow event structures and flow nets. In *Semantics of Systems of Concurrent Processes*, 1990.
6. P. Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In *FoSSaCS*, 2003.
7. M. Viswanathan and R. Viswanathan. Foundations for circular compositional reasoning. In *ICALP*, 2001.
8. G. Winskel. Event structures. In *Advances in Petri Nets*, pages 325–392, 1986.

Service Interaction Contracts as Security Policies

Davide Basile

Dipartimento di Informatica, Università di Pisa, Italy
`basile@di.unipi.it`
`http://www.di.unipi.it/~basile/`

Abstract. We outline a methodology to check the compliance of service orchestration with respect to service contracts. The contract of a service is expressed by a finite state automaton, while the traces of the orchestration form a context-free language. A contract asserts the security policy controlling resource accesses, including accesses to the communication channels, so making manifest also the client-service interactions. The key idea of the methodology is controlling both access control and compliance by an appropriate model-checking technique. Our approach naturally deals with multi-party contracts.

Keywords: service contracts, compliance, model checking usages

1 Introduction

In Service-Oriented Computing (SOC), applications are built by combining different distributed components, called services. Standard communication protocols are used for the interaction between the parties. Service composition depends on which information about the services is made public. Security issues can make more complex service composition, since a service can impose constraints on the interactions it can hold. Also, the descriptions of services lack semantic information. Behavioral contracts have been introduced to describe the external observable behavior of a service, and can be used for guaranteeing progress. This property ensures that the whole system will never get stuck, i.e. all the components are able to successfully terminate their tasks. We consider two different paradigms for describing contracts. The contracts of Castagna et al. [3, 4] take the form of CCS processes and permit to describe if the interaction between two parties terminates or gets stuck, and when a service can be replaced with a more general one. Instead Bartoletti et al. [1, 2] introduce a core calculus for services that extends the λ-calculus with primitives for composing services in a call-by-contract fashion under security properties. They develop a static technique for extracting the abstract behaviour of a service (called History Expression) that must obey the security policies. An orchestration machinery constructs a plan, i.e. a binding between requests and services guaranteeing that the security properties are always satisfied.
We extends History Expressions to include channel communications and internal/external choice for combining the notions of security access and progress of

interactions, so merging and enriching the above surveyed approaches. We prove that compliance between client and server is a safety property. The main novelty of our approach lies in exploiting standard techniques of model checking for controlling compliance of behavioural contracts. Also differently from [1–3] we manage both multi-party contracts and sessions. Due to lack of space, we cannot compare in more detail the vaste existing literature in this field, and refer the interested reader to [6].

2 Programming Model and Verification

History Expressions (HE) are a suitable process calculus through which we abstractly describe services. Beside the standard operations of process calculi, namely I/O operations, prefixing, concatenation, guarded (tail) recursion, a History Expression H contain access events α, and two non-deterministic choice operators. The external choice $\sum_{i \in I} a_i.H_i$ proceeds according to a value received on channel a_i from the external environment; while internal choice $\bigoplus_{i \in I} \overline{a}_i.H_i$ describes a service that internally decides whether to continue with one of the summands $\overline{a}_i.H_i$. Additionally, HE have the framing $\varphi(\!|H|\!)$ specifying that the policy φ must be enforced in H. A policy is an FSA that recognizes strings of access events. An example of safety policy φ is "never perform write actions(α_{write}) after read actions(α_{read})". A trace that violates this policy is $\alpha_{read}\alpha_{write}$. Finally, HE describe the opening and closing of a session by the expression $\mathtt{open}_{r,\varphi} H \mathtt{close}_r$, where r represents the unique identifier of the request (i.e. a point in the abstract syntax tree of H) and φ is the policy that the responding service must obey. Inside the session, two services can synchronize on I/O actions. Two services are *compliant* if for every output action the other party is ready to perform the corresponding input action. Every services is published at a location ℓ. An orchestrator statically creates a plan π which is a binding between the request r and the location of the service choosen for opening the session. Only two services are involved in a session. Nested sessions are possible, since a service involved in a session can open a new session with another service. A plan π is valid if the two services are compliant and the server does not violates the policy φ. Finally a network N is the parallel composition of different services and sessions.
The operational semantic is defined by a transition system. The configurations of a network have the form $R \triangleright N$, where R is a set of services and N is the active network, i.e. the active clients and services. The set R is partitioned into two parts: (1)$\{\ell_i : H_i\}^?_{i \in 1 \dots k}$ is the set of stand-by available services that can be invoked with a $\mathtt{open}_{r,\varphi}$ operation and (2)$\{\ell_i : H_i\}^!_{i \in 1 \dots k'}$ is the set of busy services, which are involved in sessions. To help intuition, we work out the following running example. Consider the services:

$$H_1 = a \cdot (\mathtt{open}_{2,\varphi_2}\, \overline{d}.(e + f)\ \mathtt{close}_2) \cdot (\overline{b} \oplus \overline{c}) \cdot d \quad H_2 = \beta \cdot d.(\overline{e} \oplus \overline{f}) \cdot \alpha$$

$$H_3 = a.\overline{g} \qquad H_4 = \mathtt{open}_{1,\varphi_1}\, \overline{a}.(b + c)\ \mathtt{close}_1 \qquad H_5 = \mathtt{open}_{3,\varphi_3}\, \overline{a}.g\ \mathtt{close}_3$$

We can see that H_2 performs the access events α, β. Let φ_2 say "never β after α", while the actual definitions of φ_1 and φ_3 is immaterial. By abuse of notation, when it is clear from the context, we indentify a service with the location where it is running. Let the initial configuration of the network be:

$$\{\ell_1 : H_1, \ell_2 : H_2, \ell_3 : H_3\}^? \cup \{\}^! \rhd \ell_4 : H_4 \| \ell_5 : H_5$$

Assume that the orchestrator plan is of the form $\pi = \bigcup_{i\in\{1,2,3\}}(r_i, \ell_i)$. We can see that all the services are compliant and ℓ_2 respects the policy φ_2. Suppose now that ℓ_4 fires the $\texttt{open}_{1,\varphi_1}$ operation, we have:

$$\{\ell_2 : H_2, \ell_3 : H_3\}^? \cup \{\ell_1 : H_1\}^! \rhd [\ell_4 : \overline{a}.(b + c)\ \texttt{close}_1, \ell_1 : \varphi_1(\!|H_1|\!)] \| \ell_5 : H_5$$

Now ℓ_1 is engaged in the session with the service ℓ_4, because $\pi(r_1) = \ell_1$. The service ℓ_1 is marked busy in R and its behaviour is checked against φ_1. The service ℓ_5 opens a new session, and the resulting configuration becames:

$$\{\ell_2 : H_2\}^? \cup \{\ell_1 : H_1, \ell_3 : H_3\}^! \rhd [\ell_4 : \ldots, \ell_1 : \ldots] \| [\ell_5 : \overline{a}.g\ \texttt{close}_3, \ell_3 : \varphi_3(\!|H_3|\!)]$$

There are two parallel sessions. The services ℓ_5 and ℓ_3 will synchronize on the channels a and g so that ℓ_5 closes the session and terminates, restoring ℓ_3 to its initial state as an available service. Then ℓ_1 and ℓ_4 synchronize on channel a:

$$\{\ell_2 : H_2, \ell_3 : H_3\}^? \cup \{\ell_1 : H_1\}^! \rhd [\ell_4 : (b + c)\ \texttt{close}_1, \ell_1 : \texttt{open}_{2,\varphi_2} \ldots]$$

The service in ℓ_3 turns back to available service. Now ℓ_1 opens a new session with ℓ_2 while ℓ_4 is waiting:

$$\{\ell_3 : H_3\}^? \cup \{\ell_1 : H_1, \ell_2 : H_2\}^! \rhd [\ell_4 : \ldots, [\ell_1 : \ldots, \ell_2 : \varphi_2(\!|H_2|\!)]]$$

This is a nested session: ℓ_1 and ℓ_2 synchronize, note that φ_2 is respected. Eventually ℓ_1 closes the session:

$$\{\ell_3 : H_3, \ell_2 : H_2\}^? \cup \{\ell_1 : H_1\}^! \rhd [\ell_4 : (b + c)\ \texttt{close}_1, \ell_1 : (\overline{b} \oplus \overline{c}) \cdot d]$$

Here ℓ_4 will receive the input on channel b or c and it will close the session. We can see that ℓ_1 could receive another message on channel d, but since the session is closed it gets back to its initial state. The final configuration is:

$$\{\ell_1 : H_1, \ell_2 : H_2, \ell_3 : H_3\}^? \cup \{\}^! \rhd \epsilon$$

For generating a valid plan we perform three steps. The first two find the compliant services for each request and check if the selected service respects the policy φ imposed by the client. For checking compliance of a given client with a sub-term of the form $\texttt{open}_{r,\varphi} H_1 \texttt{close}_r$ and an available service $\ell_2 : H_2$; we calculate a projection of H_1 and H_2 on their communication actions; operationally we remove all the policies φ, all the access events α and all the sub-terms $\texttt{open}_{r',\varphi'} \ldots \texttt{close}_{r'}$ nested in H_1 and H_2. Then, we make the product automaton $\mathcal{A}$ of the resulting transition systems. We only have finitely many states in $\mathcal{A}$. We fully characterize compliance of services by checking in each state that the client has not terminated and for all the possible output actions that a service is ready to fire, the other party is ready to perform the corresponding input action. We also check that at least one of the two services can perform one output. It turns out that compliance is an invariant property: a subset of the safety properties [5]. Now H_1 and H_2 are compliant if and only if the language of the product automaton $\mathcal{A}$ is empty: no final states exist in which the above condition do not hold. We also have to check if the choosen service respects the policy φ. For doing so we

discard all the communication actions (possibly transforming $\oplus$ in $+$). Finally, an important property is that a History Expression H is valid under a policy φ if and only if $[\![H]\!] \cap [\![\varphi]\!] = \emptyset$, i.e. if and only if the languages $[\![H]\!]$ of the traces of access events of H and $[\![\varphi]\!]$ of the offending traces of φ do not intersect. Since $[\![H]\!]$ is context-free and $[\![\varphi]\!]$ is regular, and emptiness of a context-free language is decidable, so is our problem. Indeed several algorithms and tools show this approach feasible. The following example explains how to resolve the request $\mathtt{open}_{2,\varphi_2}$ occuring in H_1 of our running example. The projection of H_2 on his communication actions give raise to the service $d.(\overline{e} \oplus \overline{f})$. The product automaton $\overline{d}.(e+f) \otimes d.(\overline{e} \oplus \overline{f})$ has three states: $\{\langle \overline{d}.(e+f), d.(\overline{e} \oplus \overline{f})\rangle, \langle e+f, \overline{e} \oplus \overline{f}\rangle, \langle \epsilon, \epsilon\rangle\}$. The set of final states is empty: no state satisfies the conditions described above. Recall that φ_2 says "never β after α", the projection of H_2 on the access events give raise to the service $\beta \cdot \alpha$, we have $[\![\varphi_2]\!] = \{\Sigma^* \alpha \Sigma^* \beta \Sigma^*\}$ and $[\![\beta \cdot \alpha]\!] = \{\beta \cdot \alpha\}$ and the intersection is trivially the empty language, therefore the two services are compliant and $(r_2, \ell_2) \in \pi$. Proceeding in this way, we generate the set of compliant services for each request. Finally the third last step ensures that a service never gets stuck while is waiting to opening a session. First we generate a "locally"viable plan for each service. One of the compliant services for each request is selected such that it will never be the case that two nested request r_1, r_2 are resolved by the same service. For example, the following network has no "locally"viable plan:
$\{\ell_2 : \overline{a}.(b+c+d+e)\}^? \rhd \ell_1 : \mathtt{open}_{1,\_}a.\mathtt{open}_{2,\_}a.(\overline{b} \oplus \overline{c})\mathtt{close}_2(\overline{b} \oplus \overline{c} \oplus \overline{d})\mathtt{close}_1$
Indeed the service at ℓ_2 is compliant with both the request r_1 and r_2. The plan $\pi = \{(r_1, \ell_2), (r_2, \ell_2)\}$ is not locally viable for ℓ_1: the service at ℓ_2 will never be available to open the session r_2 since it is still involved in r_1.
The algorithm for generating a locally viable plan for a service at every iteration picks the outermost *open/close* subterm and selects one of the services compliant with that request. Then it removes from the selected service the set of compliant services of the nested *open/close* subterms. All the locally viable plans are merged into a global plan. The tecnique we adopt consists in building the state graph of the network and in checking that there are no cycles where a service is able to open a session and it will never do it. Finally we plan to implementing the algorithms outlined above in one of the existing model-checker.

References

1. Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, Roberto Zunino: Call-by-Contract for Service Discovery, Orchestration and Recovery. *LNCS: 232-261.*
2. Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari: Planning and verifying service composition. *Journal of Computer Security 17(5): 799-837 (2009)*
3. Giuseppe Castagna, Niel Gesbert, Luca Padovani: A Theory of Contracts for Web Services. *ACM TOPLAS*, 31(5), 2009.
4. Giuseppe Castagna and Luca Padovani: Contracts for mobile processes. *LNCS: 211-228* 2009.
5. C. Baier and J.-P. Katoen: Principles of Model Checking. *MIT Press*, 2008.
6. Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, Roberto Zunino: Local policies for resource usage analysis. *(TOPLAS)* 31(6) (2009).

Checking Satisfiability of CLTL without Automata

Marcello M. Bersani, Achille Frigeri

Politecnico di Milano `bersani@elet.polimi.it,achille.frigeri@polimi.it`

1 Introduction

Finite-state system verification has attained great successes, both using automata-based and logic-based techniques. Examples of the former are the so-called explicit-state model checkers [1] and symbolic model checkers [2]. However, some of the best results have been obtained by logic-based techniques, such as Bounded Model Checking (BMC) [3], a fully automated (although potentially incomplete) procedure. In BMC, a finite-state machine A (typically, a version of Büchi Automata) and a desired property P expressed in Propositional Linear Temporal Logic (PLTL) are translated into a Boolean formula ϕ to be fed to a SAT solver. The translation is made finite by bounding the number of time instants. However, infinite behaviors, which are crucial in proving, e.g., liveness properties, are also considered by using the well-known property that a Büchi Automaton accepts an infinite behavior if, and only if, it accepts an infinite periodic behavior. Hence, chosen a bound $k > 0$, a Boolean formula ϕ_k is built, such that ϕ_k is satisfiable if and only if there exists an infinite periodic behavior of the form $\alpha\beta^\omega$, with $|\alpha\beta| \leq k$, that is compatible with system A while violating property P. This procedure allows counterexample detection, but it is not complete, since the violations of property P requiring "longer" behaviors, i.e., of the form $\alpha\beta^\omega$ with $|\alpha\beta| > k$, are not detected. However, in many practical cases it is possible to find bounds large enough for representing counterexamples, but small enough so that the SAT solver can actually find them in a reasonable time.

Clearly, the BMC procedure can be used to check satisfiability of a PLTL formula, without considering a finite state system A. This has practical applications, since a PLTL formula can represent both the system and the property to be checked (see, e.g., [4], where the translation into Boolean formulae is made more specific for dealing with satisfiability checking and metric temporal operators). We call this case *Bounded Satisfiability Checking* (BSC), which consists in solving a so-called Bounded Satisfiability Problem: Given a PLTL formula P, and chosen a bound $k > 0$, define a Boolean formula ϕ_k such that ϕ_k is satisfiable if, and only if, there exists an infinite periodic behavior of the form $\alpha\beta^\omega$, with $|\alpha\beta| \leq k$, that satisfies P.

The introduction of many extensions of temporal logic proposed in order to express property of *infinite*-state systems, has lead to the study of CLTL($\mathcal{D}$), a general framework extending the future-fragment of PLTL by allowing arithmetic constraints belonging to a generic constraint system $\mathcal{D}$. The resulting logics are expressive and well-suited to define infinite-state systems and their properties, but, even for the bounded case, their satisfiability is typically undecidable [5], since they can simulate general two-counter machines when $\mathcal{D}$ is powerful enough (e.g., Difference Logic). However, there are some decidability results, which allow in principle for some kind of automatic verification. Most notably, satisfiability of CLTL($\mathcal{D}$) is decidable (in PSPACE) when $\mathcal{D}$ is the class of Integer Periodic Constraints (IPC*) [6], or when it is the structure $(D, <, =)$ with $D \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ [7]. In these cases, decidability is shown by using an automata-based approach similar to the standard case for LTL, by reducing satisfiability checking to emptiness verification of Büchi automata. Given a CLTL($\mathcal{D}$) formula ϕ, with $\mathcal{D}$ as in the above cases, it is in fact possible to define an automaton $\mathcal{A}_\phi$ such that ϕ is satisfiable if, and only if, the language recognized by $\mathcal{A}_\phi$ is not empty. These results, although of great theoretical interest, are not well suited for a direct implementation, since the involved constructions are very inefficient.

We extended [8] the above results to a more general logic, called CLTLB($\mathcal{D}$), which is an extension of PLTLB (PLTL with Both future and past operators) with arithmetic constraints in constraint system $\mathcal{D}$, and consider a procedure for satisfiability verification that does not rely on automata constructions. This procedure is implemented in the $\mathbb{Z}$ot toolkit, verified by standard SMT-solvers, such as z3 [9].

The idea of the procedure is to verify satisfiability by checking a finite number of k-satisfiability problems. Informally, k-satisfiability amounts to looking for ultimately periodic *symbolic* models of the form $\alpha\beta^\omega$, i.e., such that prefix $\alpha\beta$ of length k admits a bounded arithmetic model (up to instant k). Although the k-bounded problem is defined with respect to a bounded arithmetical model, it provides a finite representation of infinite symbolic models by means of ultimately periodic words. When CLTLB($\mathcal{D}$) has the property that its ultimately periodic symbolic models, of the form $\alpha\beta^\omega$, always admit an arithmetic model, then the k-satisfiability problem can be reduced to satisfiability of QF-EU$\mathcal{D}$ (the theory of quantifier-free equality and uninterpreted functions combined with $\mathcal{D}$). In this case, k-satisfiability is equivalent to satisfiability over infinite models.

Symmetrically to standard LTL, where bounded model-checking and SAT-solvers can be used as an alternative to automata-theoretic approaches to model-checking, reducing satisfiability to k-satisfiability allows SMT-solvers to be used in solving satisfiability for CLTLB($\mathcal{D}$) formulae, instead of checking emptiness of a Büchi automaton. Moreover, when the length of all prefixes $\alpha\beta$ to be tested is bounded by some finite K, then the number of bounded problems to be solved is also bounded. Therefore, we also proved that k-satisfiability is *complete* with respect to the satisfiability problem, i.e., by checking at most K bounded problems satisfiability of CLTLB($\mathcal{D}$) formulae can always be answered.

2 Bounded Satisfiability Problem

The k-satisfiability problem for CLTLB($\mathcal{D}$) formulae is defined in terms of the existence of a so-called k-bounded arithmetical model σ_k, which provides a finite representation of infinite symbolic models by means of ultimately periodic words. This allows to prove that k-satisfiability is still representative of the satisfiability problem. In fact, for some constraint systems, a bounded solution can be used to build the infinite model σ for the formula from the k-bounded one σ_k and from its symbolic model. We showed that a formula ϕ is satisfiable if, and only if, it is k-satisfiable and its bounded solution σ_k can be used to derive its infinite model σ. In case of negative answer to a k-bounded instance, we can not immediately entail the unsatisfiability of the formula. However, we proved that for every formula ϕ there exists an upper bound K, which can effectively be determined, such that if ϕ is not k-satisfiable for all k in $[1, K]$, then ϕ is unsatisfiable.

A bounded symbolic model is, informally, a finite representation of infinite CLTLB($\mathcal{D}$) models over the alphabet of symbolic valuations $SV(\phi)$. We restrict the analysis to ultimately periodic symbolic models, i.e., of the form $\rho = \alpha(\beta)^\omega$. The Bounded Satisfiability Problem (BSP) is defined with respect to a k-bounded model σ_k (i.e., an assignment for variable in the first k-instants), a finite sequence ρ' (with $|\rho'| = k+1$) of symbolic valuations and a k-bounded satisfaction relation $\models_k$ defined as follows:

$$\sigma_k, 0 \models_k \rho' \text{ iff } \sigma_k, i \models \rho'(i) \text{ for all } 0 \leq i \leq k.$$

The *k-satisfiability problem* of formula ϕ is defined as follows:

Input A CLTLB($\mathcal{D}$) formula ϕ, a constant $k \in \mathbb{N}$
Problem Is there an ultimately periodic sequence of symbolic valuations $\rho = \alpha(\beta)^\omega$ (with $|\alpha\beta| = k+1$), such that $\rho, 0 \models \phi$ and which admits a k-bounded model σ_k such that $\sigma_k \models_k \rho'$, with $\rho' = \alpha\beta$?

Since the length k is fixed, the procedure for determining the satisfiability of CLTLB($\mathcal{D}$) formulae over bounded models is not complete: even if there is no accepting run of automaton $\mathcal{A}_\phi$ when ρ' as above has length k, there may be accepting runs for a larger ρ'.

Definition 1. *Given a CLTLB($\mathcal{D}$) formula ϕ, its* completeness threshold K_ϕ*, if it exists, is the smallest number such that ϕ is satisfiable if and only if ϕ is K_ϕ-satisfiable.*

Theorem 1. *Let ϕ be a CLTLB($\mathcal{D}$) formula. If $\mathcal{D}$ is $(D, <, =)$, then the completeness threshold exists and is less then $|SV(\phi)| \cdot 2^{|\phi|}$. If $\mathcal{D}$ is IPC*, then the completeness threshold exists and is less then $4|V|^2|\lambda|^2|SV(\phi)| \cdot 2^{|\phi|}$, where λ is an effectivly constant depending on the depth of ϕ.*

3 Encoding for BSP without Automata

We proved that the BSP for a CLTLB($\mathcal{D}$) formula can be reduced to the satisfiability of a quantifier-free formula in the theory EUF $\cup\, \mathcal{D}$ (QF-EU$\mathcal{D}$), where EUF is the theory of Equality and Uninterpreted Functions, provided that $\mathcal{D}$ includes a copy of $\mathbb{N}$ with the successor relation and that EUF $\cup\, \mathcal{D}$ is consistent. The last condition is easily verified in the case of the union of two consistent, disjoint, stably infinite theories (as is the case for EUF and arithmetic). In [10] a similar approach is described for the case of Integer Difference Logic (DL) constraints. It is worth noting that standard LTL can be encoded by a formula in QF-EU$\mathcal{D}$ with $\mathcal{D} = (\mathbb{N}, <)$. In this case, the encoding is more succinct than the Boolean one proposed in [11].

We denote the encoding of the BSP for ϕ with bound k by $|\phi|_k$. We proved the main equivalence result which draws the connection between such encoding and the k-satisfiability problem.

Theorem 2. *Let $\phi \in$ CLTLB($\mathcal{D}$) with $\mathbb{N}$ definable in $\mathcal{D}$ together with the successor relation, ϕ is k-satisfiable with respect to $k \in \mathbb{N}$ if, and only if, $|\phi|_k$ is satisfiable.*

Proposition 1. *Let $\phi \in$ CLTLB($\mathcal{D}$) with $\mathbb{N}$ definable in $\mathcal{D}$ together with the successor relation, ϕ is k-satisfiable with respect to $k \in \mathbb{N}$ if, and only if, ϕ has an ultimately periodic model $\alpha\beta^\omega$ with $|\alpha\beta| = k + 1$.*

References

1. Holzmann, G.: The model checker SPIN. IEEE Transactions on Software Engineering **23**(5) (may 1997) 279 –295
2. Clarke, E., McMillan, K., Campos, S., Hartonas-Garmhausen, V.: Symbolic model checking. In: Computer Aided Verification. Volume 1102 of Lecture Notes in Computer Science. (1996) 419–422
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1579 of Lecture Notes in Computer Science. (1999) 193–207
4. Pradella, M., Morzenti, A., San Pietro, P.: Bounded satisfiability checking of metric temporal logic specifications. ACM Transactions on Software Engineering and Methodology (TOSEM) (2012) To appear.
5. Demri, S., Gascon, R.: The effects of bounding syntactic resources on Presburger LTL. Technical Report LSV-06-5, LSV (2006)
6. Demri, S., Gascon, R.: The effects of bounding syntactic resources on Presburger LTL. In: International Syposium on Temporal Representation and Reasoning (TIME), IEEE Computer Society (2007) 94–104
7. Demri, S., D'Souza, D.: An automata-theoretic approach to constraint LTL. Information and Computation **205**(3) (2007) 380–415
8. Bersani, M.M., Frigeri, A., Morzenti, A., Pradella, M., Rossi, M., San Pietro, P.: Constraint LTL Satisfiability Checking without Automata. ACM Transactions on Computational Logic (submitted)
9. Microsoft Research: Z3: An efficient SMT solver. http://research.microsoft.com/en-us/um/redmond/projects/z3/ (2009)
10. Bersani, M.M., Cavallaro, L., Frigeri, A., Pradella, M., Rossi, M.: SMT-based verification of LTL specification with integer constraints and its application to runtime checking of service substitutability. In: IEEE International Conference on Software Engineering and Formal Methods. (2010) 244–254
11. Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods in Computer Science **2**(5) (2006)

On the Complexity of Pure 2D Context-free Grammars

Marcello M. Bersani, Achille Frigeri, Alessandra Cherubini

Politecnico di Milano
`bersani@elet.polimi.it,{achille.frigeri,alessandra.cherubini}@polimi.it`

1 Introduction

Picture languages generalize classical string languages to two-dimensional arrays. Several approaches have been proposed during the years; consequently, a general classification and a detailed comparison of the classes proposed turns to be necessary. We studied in detail closure properties of (regular) pure 2D context-free grammars (R)P2DCFG [1], and the complexity of the membership problem [2].

2 Preliminaries

General definitions Let Σ be a finite alphabet. A two-dimensional array of elements of Σ is a picture over Σ. The set of all pictures over Σ is denoted by Σ^{++}. For $h,k \geq 0$, $\Sigma^{(h,k)}$ denotes the set of pictures of size (h,k), and $\Sigma^{**} = \Sigma^{++} \cup \lambda$, where λ is the empty picture. Conversely, if $p \in \Sigma^{**}$, we denote by $|p|_{row}$ and $|p|_{col}$, the number of rows and columns of p, respectively. The size of p is the pair $|p| = (|p|_{row}, |p|_{col})$. As in the one-dimensional case, a *picture language* is a subset of Σ^{**}. For $1 \leq i \leq |p|_{row}$, $1 \leq j \leq |p|_{col}$, the element of p in the i-th row and j-th column is called a *pixel* and denoted by $p(i,j)$.

Operations on picture languages Let Γ and Σ be two finite alphabets and $\pi : \Gamma \to \Sigma$ a function between them, if $p \in \Gamma^{(h,k)}$, the *projection* of p by π is the picture $p' \in \Sigma^{(h,k)}$ such that $p'(i,j) = \pi(p(i,j))$, for all $1 \leq i \leq |p|_{row}$, $1 \leq j \leq |p|_{col}$. Projection naturally extends to languages. *Row and column concatenations* are partial operations on pictures denoted $\ominus$ and $\oplus$, respectively. If $p, q \in \Sigma^{(h,*)}$ (resp. $p, q \in \Sigma^{(*,k)}$) $p \oplus q$ (resp. $p \ominus q$) is the horizontal (resp. vertical) juxtaposition of p and q. With $p^{n\oplus}$ (resp. $p^{n\ominus}$) is denoted the horizontal (resp. vertical) juxtaposition of n copies of p; $p^{+\oplus}$ (resp. $p^{+\ominus}$) is the corresponding closure. Concatenations also extend to languages.

Pure 2D Context-free Grammars Context free grammars which make use of only terminal symbols (i.e., *pure grammars*) have been well investigated in the theory of string languages. Pure 2D context-free grammars [3], unlike Matrix grammars ([4,5]), admit rewriting any row/column of pictures with no priority of columns and rows. Row/column sub-arrays of pictures are rewritten in parallel by equal length strings and by using only terminal symbols.

Definition 1. *A* pure 2D context-free grammar (P2DCFG) *is a 4-tuple* $G = (\Sigma, P^c, P^r, S')$ *where:*

1. *Σ is a finite set of symbols;*
2. *$P^c = \{c_i \mid 1 \leq i \leq m\}$ is the set of* column rule tables*, where a table c_i is a set of context-free rules of the form $a \to \alpha$, $a \in \Sigma$, $\alpha \in \Sigma^+$ s.t. for any two rules $a \to \alpha$, $b \to \beta$ in c_i, $|\alpha| = |\beta|$ where $|\alpha|$ denotes the length of α.*
3. *$P^r = \{r_i \mid 1 \leq i \leq n\}$ is the set of* row rule tables*, where a table r_i is a set of context-free rules of the form $a \to {}^t\alpha$, $a \in \Sigma$, $\alpha \in \Sigma^+$ s.t. for any two rules $a \to {}^t\alpha$, $b \to {}^t\beta$ in r_i, $|\alpha| = |\beta|$.*
4. *$S' \subseteq \Sigma^{++}$ is a finite set of* axioms.

For any two arrays $p_1, p_2 \in \Sigma^{**}$, p_2 is derived from p_1 in G, in symbols $p_1 \Rightarrow p_2$, if p_2 is obtained from p_1 by either rewriting a column of p_1 by applying to each letter of the column a rule in a table $c_i \in P^c$, or rewriting a row of p_1 by applying to each letter of the row a rule in a table $r_i \in P^r$. The set of symbols occurring in the column (resp. row) that will be rewritten by $c_i \in P^c$ (resp. $r_i \in P^r$) must be a subset of $\{a \mid a \to \alpha \in c_i\}$ (resp. $\{a \mid a \to {}^t\alpha \in r_i\}$). Otherwise, the derivation can not be achieved because there are some symbols for which c_i (resp. r_i) does not provide a rewriting rule. Derivation $\Rightarrow$ is a binary relation over Σ^{**} and its reflexive and transitive closure is denoted by $\overset{*}{\Rightarrow}$. The language $\mathcal{L}(G)$ generated by the P2DCF grammar G is the set $\{p \mid S \overset{*}{\Rightarrow} p \in \Sigma^{++}$ for some $S \in S'\}$. The family of languages generated by some P2DCF grammar is denoted by P2DCFL. It is worth noticing that all pictures derived at each step by applying a rewriting rule from the set P^c or P^r are legal pictures. Since non-terminals are not admitted by P2DCF grammars, each derivation consists of characters of Σ. To augment the expressive power of P2DCF grammars, the sequence of rules to be used can be led by a *control language*.

Definition 2. *A* pure 2D context-free grammar with regular control (RP2DCFG) *is a tuple $G_r = (G, \Gamma, \mathcal{C})$ where:*

1. *G is a P2DCF grammar;*
2. *Γ is the control alphabet, actually the set of labels of the rule tables in $P^c \cup P^r$;*
3. *$\mathcal{C} \subseteq \Gamma^*$ is the regular control associated to the grammar.*

If $p \in \Sigma^{**}$ and $S \in S'$, p is derived from S in G_r by means of a control word $w = w_1 w_2 \ldots w_n \in \mathcal{C}$, in symbols $S \Rightarrow_w p$, if p is obtained from S by applying the column/row rules defined by w. The language $\mathcal{L}(G)$ generated by the RP2DCF grammar G_r is the set of pictures $\{p \mid S \Rightarrow_w p \in \Sigma^{++}$ for some $w \in C\}$. The family of languages generated by some RP2DCF grammar is denoted by RP2DCFL. The family P2DCFL is strictly included in RP2DCFL, indeed each P2DCF language is a RP2DCF language with control $C = \Gamma^*$. On the other hand, the language of squares over the symbol a is not a P2DCF language but can be generated by the RP2DCF grammar $(G, \{c, r\}, (cr)^*)$ where $G =$

$(\{a\}, \{c\}, \{r\}, S)$ and $S \to a$, $c = \{a \to aa\}$, $r = \{a \to {}^t(aa)\}$. In order to refine the given definition of this class of grammars, we consider RP2DCFG whose alphabet is $\Sigma = \Sigma_T \cup \Sigma_C$ where Σ_T is the alphabet of final symbol defining the pictures, and Σ_C is a set of auxiliary characters, or *control symbols*, which are involved only in the process of derivation. Yet, control symbols can not be considered as proper non-terminal symbols since they have to be rewritten by means of derivations guided by the control language, so that no control symbols appear in the final picture and the generating device can still be seen as a pure grammar. We showed that the use of control symbols is needed to reach the full expressiveness of RP2DCFG, i.e., there exist RP2DCF languages that can not be defined without the use of control symbols.

3 Normal form and parsing complexity

Normal forms of generating grammars are useful tools to get in a easier way properties of languages and make comparisons between different generating devices. Normal forms, in general, force some constraints on the size and on the alphabet of the strings/pictures occurring in the left and right parts of the productions. Since the model we are considering is a pure grammar, and productions in the row/column tables always rewrite a single character into the right part that is a string (or the transpose of a string), the normal form we ask for has to fix the length of the strings in the right part of each production: formally a (R)P2DCFG is in normal form if all productions have the form $a \to \alpha$ or $a \to {}^t\beta$ with $|\alpha| = |\beta| = 2$. Pure 2D context-free grammars do not have a normal form. Indeed, the language $L_3(a)$ of pictures of size (hn, kn) (where h, k are positive integers) on the alphabet $\{a\}$ are generated by the P2DCFG $(\{a\}, \{c_1\}, \{r_1\}, S)$, where c_1 and r_1 are, respectively, composed by the unique rules $a \to a^n$, $a \to {}^t(a^n)$ and S is the square of a of size (n, n), but no P2DCFG with column and row productions of length 2. However, we have the following.

Proposition 1. *Each (R)P2DCF grammar is equivalent to a RP2DCFG in normal form.*

Actually, a more general result holds: each pure 2D context-free grammar with a control language belonging to a class of languages closed with respect to finite substitutions admits a normal form.

Theorem 1. *The general problem of the membership of a picture into a language generated by a P2DCFG is NP-complete.*

We proved the NP-hardness by providing a (polynomial-time) reduction of SAT to the membership problem for a P2DCFL with an alphabet of at least 5 symbols. So, the question concerning the parsing complexity for pure 2D grammars with smaller alphabets is quite natural. We have the following results.

Theorem 2. *The parsing of a language generated by (R)P2DCF grammars with unary alphabet is in P.*

On the other hand, the NP-completeness of the membership for RP2DCFL characterizes all the languages with at least two symbols. The proof consists of a reduction of the set-covering problem to the membership for RP2DCFG.

Theorem 3. *The general problem of the membership of a picture to a language generated by a RP2DCFG with (at least) two symbols is NP-complete.*

4 Closure properties

In this section, we present some closure properties of the class of $(R)P2DCFL$. Some of them are known from [3] but here we provide new results. First we considered projections:

Proposition 2. *Let $G = (\Sigma, P^c, P^r, S)$ be a $P2DCFG$ and let π be a projection from the alphabet Σ to the alphabet Δ. Then $\pi(L(G))$ is a subset of the language generated by a P2DCFG $\overline{G}$ such that $\pi(L(G)) = L(\overline{G}) \cap \Delta^{++}$.*

Proposition 3. *Let $G_{reg} = (G, \Gamma, \mathcal{C})$ with $G = (\Sigma, P^c, P^r, S)$ be a RP2DCFG and let π be a projection from the alphabet Σ to the alphabet Δ. Then $\pi(L(G))$ is a subset of the language generated by a RP2DCFG $\overline{G}_{reg} = (\overline{G}, \overline{\Gamma} \cup \{c_\pi\}, \overline{\mathcal{C}}\{c_\pi\}^*)$ such that $\pi(L(G_{reg})) = L(\overline{G}_{reg}) \cap \Delta^{++}$.*

The two previous propositions show how projection may change the expressiveness of the class of languages of P2DCFG. A similar result is obtained also for Tiling Systems which are the projection of Local languages. In [3] the authors proved that P2DCFL are not closed under union and under row/column concatenation and proved that the closure under union can be retained when a regular control language is added to the grammars. Yet, no results is provided concerning the closure under intersection. We have the following:

Proposition 4. *Let $G_r^1 = (G_1, \Gamma_1, \mathcal{C}_1)$ and $G_r^2 = (G_2, \Gamma_2, \mathcal{C}_2)$ be two RP2DCFG. Then, the language $\mathcal{L}(G_r^1) \cup \mathcal{L}(G_r^2)$ is RP2DCFL.*

Proposition 5. *The family of P2DCFL is not closed under intersection.*

The family of P2DCFL was shown not to be closed under row/colum concatenation in [3]. We conjecture that this holds also for the family of RP2DCFL.

References

1. Bersani, M.M., Frigeri, A., Cherubini, A.: On Some Classes of 2D Languages and Their Relations. In: IWCIA. (2011) 222–234
2. Bersani, M.M., Frigeri, A., Cherubini, A.: Expressiveness and Complexity Of Regular Pure 2D Context-free Languages (RP2DCFL). International Journal of Computer Mathematics (to appear)
3. Subramanian, K.G., Nagar, A.K., Geethalakshmi, M.: Pure 2D picture grammars (P2DPG) and P2DPG with regular control. In: IWCIA. (2008) 330–341
4. Siromoney, G., Siromoney, R., Krithivasan, K.: Abstract families of matrices and picture languages. Computer Graphics and Image Processing **1** (1972) 284–307
5. Siromoney, G., Siromoney, R., Krithivasan, K.: Picture languages with array rewriting rules. Information and Control **23(5)** (1973) 447–470

A Secure Coordination of Agents with Nonmonotonic Soft Concurrent Constraint Programming[⋆,⋆⋆]

Stefano Bistarelli[1,2] and Francesco Santini[1,3]

[1] Dipartimento di Matematica e Informatica, Università di Perugia, Italy
bista,francesco.santini@dmi.unipg.it
[2] Istituto di Informatica e Telematica, IIT-CNR, Pisa, Italy
stefano.bistarelli@iit.cnr.it
[3] Centrum Wiskunde & Informatica, Amsterdam, Netherlands
F.Santini@cwi.nl

Extended abstract. In the context of distributed/concurrent systems, the ability to coordinate the agents coupled with the possibility to control the actions they perform is significantly important. The necessity of guaranteeing security properties is rapidly arising: in an open and untrusted environment, an attacker can threat the integrity and confidentiality properties of the exposed data. The ingredient at the basis of our research is *Nonmonotonic Soft Concurrent Constraint Programming* (*NSCCP*) [4]. The *NSCCP* language extends the classical *Soft Concurrent Constraint Programming* (*SCCP*) language [3] with the possibility of relaxing (i.e. retracting or removing constraints) the store with a *retract* action, which clearly improves the expressivity of the language [4]. However, non-monotonicity raises further security concerns, since the store σ is a shared and centralized resource accessed in a concurrent manner by multiple agents at the same time: may an agent A relax a constraint c added to σ by the agent B? Since in this case we are reasoning about soft constraints instead of crisp ones, "how much" of c can agent A relax? Even if *(S)CCP* has been successfully used to analyse security issues [1], the paradox is that security aspects linked to the language itself have not been inspected yet. For these reasons, a constraint-based language modeling the interactions among agents in an untrusted environment needs to support security by providing some access control mechanisms with a granularity at the level of the single constraints. Therefore, our intent is to equip the core actions of the *NSCCP* language [4] with a formal system of rights on the constraints and then study the execution of agents from this new point of view. We take inspiration from the *Access Control List* (*ACL*) model [6], which is one of the security concepts in the design of secure computing systems. An ACL specifies which users or system processes are granted access to objects, as

⋆ Work carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. This Programme is supported by the Marie Curie Co-funding of Regional, National and International Programmes (COFUND) of the European Commission.

⋆⋆ Research partially supported by MIUR PRIN 20089M932N project: "Innovative and multi-disciplinary approaches for constraint and preference reasoning".

well as what operations are allowed on given objects. In this paper, when an agent A_1 adds a piece of information to the store, i.e., a constraint c, it specifies also the confidentiality and integrity rights [9] on that constraint, for each agent A_i participating to the protected computation. For instance, how much of c the agent A_3 may remove from the store (i.e. the *retract rights*) and how much of c the agent A_2 may query with an *ask* operation (i.e. the *ask rights*).

We use control mechanisms in order to guarantee some form of security and privacy on the shared store of constraints. However, since we work on the semiring-based formalism [3], our checks are focused on the quantitative, rather than qualitative, point of view, differently from previous works on Linda [8, 5, 7]. In fact, our approach is able to set "how much" of the current store each agent can *retract* or *ask*. Therefore, also the rights, together with the information they are applied on (i.e., soft constraints) are soft, in the sense they may concern "part" of the added information. In a crisp vision, if c_1 is added to the store, it is possible to prevent only the removal of the entire c_1, but not part of it. When an agent add a constraint to the store by performing a *tell* action, it also specifies the rights that all the other agents have on that constraint. We define three kinds of rights: the *tell rights*, stating how much the added constraint can be "worsened" by the other agents, the *ask rights*, which specify how much of the constraint can be "read" by each agent and the *retract rights*, describing how much of the added constraint can be removed via a retract action. The *tell* and *retract rights* can be classified as *integrity rights* [9], while the *ask rights* are classified as *confidentiality rights* [9]. In Def. 1 we define the *tell, ask* and *retract rights*.

Definition 1 (Tell, Ask and Retract Rights). *Let n be the number of agents participating to the concurrent computation.* ***Tell rights****. Each constraint c_k added to the store is associated with a vector $\Re_t = \langle c_{t_1}, c_{t_2}, \dots c_{t_n} \rangle$. c_{t_i} represents the* tell right *imposed on agent A_i. In particular, c_{t_i} represents how much the agent A_i can add (with a* tell *operation) to the constraint c_k, that is how much A_i can worsen c_k.* ***Ask rights****. Each constraint c_k added to the store is associated with a vector $\Re_a = \langle c_{a_1}, c_{a_2}, \dots c_{a_n} \rangle$. c_{a_i} represents the* ask right *imposed on agent A_i. In particular, c_{a_i} represents how much of the added c_k constraint can be read (with an* ask *operation in the common store) by agent A_i.* ***Retract rights****. Each constraint c_k added to the store is associated with a vector $\Re_r = \langle c_{r_1}, c_{r_2}, \dots c_{r_n} \rangle$. c_{r_i} represents the* retract rights *imposed on agent A_i. In particular, c_{r_i} represents how much of c_k can be removed (with a* retract *operation) by agent A_i.*

We suppose that each agent knows the name (and, consequently, also the number) of the other agents participating to the secure computation on the shared store. This is a general premise for a secure computation, as for example given in Operating Systems Primitives. Moreover, also in the other references in literature an identifier is defined for each entity whose computation is controlled [5]. Supposing to know the number of agents at the beginning of the computation is a common practice in many security-related fields, as the execution of multiple threads on the same shared memory. We propose *NSCCP* as a language to enforce a secure access over general shared resources, checking if quantitative rights over them are respected, e.g. "*Peter* may not eat more than 10% of the birthday cake". Moreover, we can suppose that the names of agents are instead names of (security) classes each agent belongs to. The rights of each class are then shared by all the included agents; in this way it is not necessary to set the rights for each single agent, or even to know their number.

With an abuse of notation we define the composition operation of rights as $\Re' = \Re \otimes \bar{\Re}$, where $\Re'$ models the new rights in the computation state after the update, while $\bar{\Re}$ represents the new rights that modify the state (parameter of the *tell* action in Fig. 1). $\Re' = \Re \otimes \bar{\Re}$ is implemented with equations $(1)\forall i. \Re'_t[i] = \Re_t[i] \otimes \bar{\Re}_t[i]$, $(2)\forall i. \Re'_a[i] = \Re_a[i] \otimes \bar{\Re}_a[i]$ and $(3)\forall i. \Re'_r[i] = \Re_r[i] \otimes \bar{\Re}_r[i]$ (i.e. respectively *tell*, *ask* and *retract rights*): for example, if we have two agents A_1 and A_2, we use the *Weighted* semiring $\langle R^+, min, +, \infty, 0\rangle$ and $\Re$, $\bar{\Re}$ are: $\Re = (\Re_t = \langle x, \bar{5}, x + y\rangle, \Re_a = \langle y, x, \bar{\mathbf{1}}\rangle, \Re_r = \langle x, z, \bar{2}\rangle)$ $\bar{\Re} = (\bar{\Re}_t = \langle y, x, \bar{3}\rangle, \bar{\Re}_a = \langle \bar{\mathbf{1}}, \bar{\mathbf{1}}, \bar{\mathbf{1}}\rangle, \bar{\Re}_r = \langle \bar{\mathbf{1}}, x, \bar{6}\rangle)$ then the $\Re'$ composition of rights is given by $\Re' = \Re \otimes \bar{\Re} = (\Re'_t = \langle x + y, x + \bar{5}, x + y + \bar{3}\rangle, \Re'_a = \langle y, x, \bar{\mathbf{1}}\rangle, \Re'_r = \langle x, x + z, \bar{8}\rangle)$.

The Secure NSCCP Language. Given a soft constraint system [3], in Fig. 1 we present the syntax of the secure *NSCCP* language [2] , which can be used in a secure coordination of agents. In Fig. 1, P is the class of programs, F is the class of sequences of procedure declarations (or clauses), A is the class of agents, c ranges over constraints, X is a set of variables and Y is a tuple of variables.

$$
\begin{aligned}
P &::= F.A \\
F &::= p(Y) :: A \mid F.F \\
A &::= secfail \mid success \mid tell(c, \bar{\Re}) \rightarrowtail A \mid retract(c) \rightarrowtail A \mid E \mid A \| A \mid \exists x.A \mid p(Y) \\
E &::= ask(c) \rightarrowtail A \mid E + E
\end{aligned}
$$

Fig. 1: Syntax of the *NSCCP* language.

The difference w.r.t. [4] is that the tell action has a new parameter (in addition to c), that is the $\bar{\Re}$ rights. When executing $tell(c, \bar{\Re})$, it is not obviously possible to quantitatively impose more rights on c than c itself: therefore, the syntactic conditions on $\bar{\Re}$ when writing NSCCP programs are that $\forall i.\ \ c \vdash \bar{\Re}_t[i],\ \ c \vdash \bar{\Re}_a[i],\ \ c \vdash \bar{\Re}_r[i]$.

To give an operational semantics to our language we describe an appropriate transition system $\langle \Gamma, T, \rightarrow\rangle$ where Γ is a set of possible configurations, $T \subseteq \Gamma$ is the set of *terminal* configurations and $\rightarrow \subseteq \Gamma \times T$ is a binary relation between configurations. The set of configuration is $\Gamma = \{\langle A, \sigma, \Re\ \rangle\}$ where $\sigma \in C$ and $\Re$ is the matrix of rights, while the set of terminal configuration is instead $T = \{\langle success, \sigma, \Re\rangle\}$. To remember also the rights, we need to extend the representation of a computation state in *NSCCP* in Def. 2.

Definition 2 (Computation States). *The state of a computation in* NSCCP *is represented by the triple $\langle A, \sigma, \Re\rangle$, where A is the description of the agent still to be executed, σ is the constraint store, and $\Re$ is the set of the rights on the constraints. $\Re$ is initialized as $\forall i. \Re_t[i] = \emptyset, \Re_a[i] = \emptyset, \Re_r[i] = \emptyset$.*

$$\textbf{R1}\ \frac{\sigma \neq \emptyset \quad \Re_t[i] \vdash c \quad \Re_t[i] = \Re_t[i] \ominus c \quad \mathit{check}(\sigma \otimes c)_{\rightarrowtail}}{\langle \mathrm{tell}^i(c, \bar{\Re}) \rightarrowtail A, \sigma, \Re \rangle \longrightarrow \langle A, \sigma \otimes c, \Re \otimes \bar{\Re} \rangle}$$

$$\textbf{R2}\ \frac{\sigma = \emptyset \quad \Re_t[i] = \Re_t[i] \ominus c \quad \mathit{check}(\sigma \otimes c)_{\rightarrowtail}}{\langle \mathrm{tell}^i(c, \bar{\Re}) \rightarrowtail A, \sigma, \Re \rangle \longrightarrow \langle A, \sigma \otimes c, \Re \otimes \bar{\Re} \rangle}$$

$$\textbf{R3}\ \frac{\Re_r[i] \vdash c \quad \Re'_r[i] = \Re_r[i] \ominus c \quad \sigma \vdash c \quad \sigma' = \sigma \ominus c \quad \mathit{check}(\sigma \ominus c)_{\rightarrowtail}}{\langle \mathit{retract}^i(c) \rightarrowtail A, \sigma, \Re \rangle \longrightarrow \langle A, \sigma', \Re' \rangle}$$

$$\textbf{R4}\ \frac{\Re_t[i] \vdash \bar{\Re}_t \quad \Re_a[i] \vdash \bar{\Re}_a \quad \Re_r[i] \vdash \bar{\Re}_r \quad p(Y) :: B \in F \quad \mathit{check}(\sigma)_{\rightarrowtail}}{\langle \mathit{execp}^i(p(Y), \bar{\Re})) \rightarrowtail A, \sigma, \Re \rangle \longrightarrow \langle A \parallel B, \sigma, \Re \cup \bar{\Re} \rangle}$$

$$\textbf{R5}\ \frac{\langle E_j, \sigma, \Re \rangle \longrightarrow \langle A_j, \sigma', \Re' \rangle \quad j \in [1, n]}{\langle \Sigma_{i=1}^n E_i, \sigma, \Re \rangle \longrightarrow \langle A_j, \sigma', \Re' \rangle}$$

$$\textbf{R6}\ \frac{\Re_a[i] \vdash c \quad \sigma \vdash c \quad \mathit{check}(\sigma)_{\rightarrowtail}}{\langle \mathrm{ask}^i(c) \rightarrowtail A, \sigma, \Re \rangle \longrightarrow \langle A, \sigma, \Re \rangle}$$

$$\textbf{R7}\ \frac{\Re_a[i] \vdash c \quad \sigma \not\vdash c \quad \mathit{check}(\sigma)_{\rightarrowtail}}{\langle \mathit{nask}(c) \rightarrowtail A, \sigma \rangle \longrightarrow \langle A, \sigma \rangle}$$

$$\textbf{R8}\ \frac{\langle A, \sigma, \Re \rangle \longrightarrow \langle A', \sigma', \Re' \rangle}{\begin{array}{c}\langle A \parallel B, \sigma, \Re \rangle \longrightarrow \langle A' \parallel B, \sigma', \Re' \rangle \\ \langle B \parallel A, \sigma, \Re \rangle \longrightarrow \langle B \parallel A', \sigma', \Re' \rangle\end{array}}$$

$$\textbf{R9}\ \frac{\langle A, \sigma, \Re \rangle \longrightarrow \langle \mathit{success}, \sigma', \Re' \rangle}{\begin{array}{c}\langle A \parallel B, \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle \\ \langle B \parallel A, \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle\end{array}}$$

$$\textbf{R10}\ \frac{\langle A[x/y], \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle}{\langle \exists x.A, \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle} \quad y\ \mathit{fresh}$$

$$\textbf{R11}\ \frac{\langle A, \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle}{\langle p(Y), \sigma, \Re \rangle \longrightarrow \langle B, \sigma', \Re' \rangle} \quad p(Y) :: A \in F$$

Fig. 2: The transition system for *NSCCP*.

In Fig. 2 we describe the operational semantics of secure *NSCCP*. A full explanation of the rules is given in [2]. In this paper we add rule **R4**; with this rule we are able to create a new agent in parallel with the other already being executed. The "body" of the new agent is described by one of the procedures defined in the declaration section F, as presented in Fig. 1: in the precondition of the rule, $p(Y) :: B \in F$. The creating agent can pass to the son a part of his right, and at most all of his rights. These rights are *not* revoked from the creator.

References

1. Bella, G., Bistarelli, S.: Soft constraint programming to analysing security protocols. TPLP 4(5-6), 545–572 (2004)
2. Bistarelli, S., Campli, P., Santini, F.: A secure coordination of agents with nonmonotonic Soft Concurrent Constraint Programming. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2012. pp. 1551–1553. ACM (2012)
3. Bistarelli, S., Montanari, U., Rossi, F.: Soft concurrent constraint programming. ACM Trans. Comput. Logic 7(3), 563–589 (2006)
4. Bistarelli, S., Santini, F.: A nonmonotonic soft concurrent constraint language to model the negotiation process. to appear in Fundamenta Informaticae (2011)
5. Gorrieri, R., Lucchi, R., Zavattaro, G.: Supporting secure coordination in SecSpaces. Fundam. Inform. 73(4), 479–506 (2006)
6. Sandhu, R., Samarati, P.: Access control: Principles and practice. IEEE Communications 32(9), 40–48 (1994)
7. Udzir, N.I., Wood, A.M., Jacob, J.L.: Coordination with multicapabilities. Sci. Comput. Program. 64(2), 205–222 (2007)
8. Vitek, J., Bryce, C., Oriol, M.: Coordinating processes with secure spaces. Sci. Comput. Program. 46(1-2), 163–193 (2003)
9. Whitman, M.E., Mattord, H.J.: Principles of Information Security. Course Technology Press, Boston, MA, USA, 3rd edn. (2007)

The Binary Perfect Phylogeny with Persistent Characters

Paola Bonizzoni[1], Anna Paola Carrieri[1], Riccardo Dondi[3], and Gabriella Trucco[2]

[1] Dipartimento di Informatica Sistemistica e Comunicazione
Univ. degli Studi di Milano - Bicocca
bonizzoni@disco.unimib.it

[2] Dipartimento di Tecnologie dell'Informazione Univ. degli Studi di Milano, Crema
gabriella.trucco@unimi.it

[3] Dipartimento di Scienze dei Linguaggi, della Comunicazione e degli Studi Culturali
Univ. degli Studi di Bergamo, Bergamo
riccardo.dondi@unibg.it.

1 Introduction

The perfect phylogeny is one of the most investigated models in different areas of computational biology. This model derives from a restriction of the parsimony methods used to reconstruct the evolution of species (taxa) characterized by a set of characters that are gained and/or lost during the evolution. In this paper we focus on the binary characters that can take only the states 0 or 1, usually interpreted as the presence or absence of the attribute in the taxa. Restrictions on the type of changes from zero to one and vice versa lead to a variety of specific models [4]. The most restrictive parsimony assumption is perfect phylogeny: a tree in which each character state can change its state from 0 to 1 at most once. The algorithmic solution of the Perfect Phylogeny model has been investigated in [5], where a well known characterization of matrices admitting a perfect phylogeny[4] and a linear time algorithm are provided. The perfect phylogeny model has been successfully applied in the context of haplotype inference [6] and very efficient polynomial time solutions to this problem have been proposed, including linear-time algorithms [3], [10], [1]. However, this model is quite restrictive to explain the biological complexity of real data, where homoplasy events (such as recurrent mutations or back mutations) are present. Thus a central goal in this model is to extend its applicability, while retaining the computational efficiency where possible.

Following this research direction, in this paper we address the problem of constructing a perfect-phylogeny under the assumption that only a special type of back mutation may occur in the tree: a character may change state only twice in the tree from 0 to 1 and from 1 to 0. These characters have already been considered in the literature and called *persistent* by T. Przytycka [9] in a general framework of tree inference.

[4] A binary matrix M admits a rooted perfect phylogeny if and only if it does not contain a pair of columns and three rows inducing the configurations $(0,1),(1,0)$ and $(1,1)$, also known as *forbidden matrix.*

We consider a binary matrix M of size $n \times m$ that has columns associated with the set $C = \{c_1, \ldots, c_m\}$ of characters and rows associated with the set $S = \{s_1, \ldots, s_n\}$ of species, then $M[i, j] = 1$ if and only if species s_i has character c_j, otherwise $M[i, j] = 0$. The gain of a character c in a phylogenetic tree is usually represented by an edge labelled by the character c. In order to model the presence of persistent characters, the loss of a character c in the tree is represented by an edge that is labelled by the negated character, denoted by $\bar{c}$. Formally, we define the persistent perfect phylogeny model as follows.

Persistent Perfect Phylogeny Let M be a binary matrix of size $n \times m$. Then a *persistent perfect phylogeny*, in short *p-pp tree* for M, is a rooted tree T that satisfies the following properties:

1. each node x of T is labelled by a vector l_x of length m. The root of T is labelled by a vector of all zeros, while for each node x of T the value $l_x[j]$ represents the state, 0 or 1, of character c_j in tree T. Each row of M labels exactly one leaf of T;
2. for each character c_j there are at most two edges $e = (x, y)$ and $e' = (u, v)$ such that e, e' occur along the same path from the root to a leaf of T; if e is closer to the root than e', then the edge e is labelled c_j, while edge e' is labelled $\bar{c_j}$;

Let us state the main problem investigated in the paper.

The **Persistent Perfect Phylogeny problem (P-PP):** Given a binary matrix M, returns a p-pp tree for M if such a tree exists.

We say that two positive characters c, c' of matrix M are in *conflict* in matrix M, if and only if the pair of columns c, c' of M induces the four gametes $(0, 1)$, $(1, 1)$, $(1, 0)$ and $(0, 0)$. Then the *conflict graph*[5] associated with a binary matrix M is the undirected graph $G_c = (C, E \subseteq C \times C)$ where a pair $(u, v) \in E$ if and only if u, v are in conflict in matrix M. Notice that a conflict graph with no edges (called *e-empty*) does not necessarily imply the existence of a rooted perfect phylogeny, because of the occurrence of the forbidden matrix with only the three configurations $(1, 1)$, $(1, 0)$ and $(1, 0)$. However, by allowing a character to be persistent, the matrix admits a rooted persistent perfect phylogeny.

In this paper we propose a graph-based solution of the problem of the reconstruction of the persistent perfect phylogeny (in short P-PP problem) that is obtained by restating the problem as a variant of the Incomplete Directed Perfect Phylogeny [8], called Incomplete Perfect Phylogeny with Persistent Completion (IP-PP problem). We show a polynomial-time algorithm that finds a solution for the input matrices described by an e-empty conflict-graph. Then we use it to develop an optimized version of the exact algorithm, that has been presented in [2] and having a a worst time complexity that is polynomial in the number n of rows of the matrix and exponential in the number m of characters. An experimental analysis shows that the new optimized version outperforms the

[5] The conflict graph is a well known concept that has been used several times in the framework of the perfect phylogeny is a graph representation of the four gametes $(0, 1)$, $(1, 1)$, $(1, 0)$ and $(0, 0)$.

previously proposed algorithm and has a wider applicability, since it can solve all input matrices within fixed time bounds, while the previous algorithm was not able to finish on some of them.

2 Solving the Persistent Perfect Phylogeny problem

Let M be a binary $n \times m$ matrix which is an instance of the P-PP problem. The *extended matrix* associated with M is a matrix M_e of size $n \times 2m$ over alphabet $\{0, 1, ?\}$ which is obtained by replacing each column c of M by a pair of columns $(c, \bar{c})$, where c is the positive character, while $\bar{c}$ is the negated character, moreover for each row s of M, it holds:

if $M[s, c] = 1$, then $M_e[s, c] = 1$ and $M_e[s, \bar{c}] = 0$,

if $M[s, c] = 0$, then $M_e[s, c] =\ ?$ and $M_e[s, \bar{c}] =\ ?$.

Informally, the assignment of the pair $(?, ?)$ in a species row s for the pair of columns $(c,\bar{c})$ means that character c could be persistent in species s, i.e. it is gained and then lost. On the contrary, the pair $(1, 0)$ assigned in a species row s for the pair $(c,\bar{c})$, means that character c is only gained by the species s. A *completion of a character* c of matrix M_e is obtained by solving the pair $(?, ?)$ given in the pair $(c, \bar{c})$ by the value $(0, 0)$ or $(1, 1)$. A *completion of matrix* M_e is a completion of all characters of M_e, while a *partial completion of* M_e is a completion of zero or more characters of M_e. We introduce below a problem to which we reduce P-PP, as shown in Theorem 1.

Incomplete Perfect Phylogeny with Persistent Completion Problem (IP-PP): given an extended matrix M_e over $\{0, 1, ?\}$ return a completion M' of M_e such that M' admits a pp tree, if it exists.

Thus we state the first result of the paper.

Theorem 1. *Let M be a binary matrix and M_e the extended matrix associated with M. Then M admits a p-pp tree if and only if there exists a completion of M_e admitting a pp tree.*

The notion of red-black graph G_{RB} for a matrix M has been introduced to find a completion of a matrix M_e. It consists of the edge colored graph (V, E) where $V = C \cup S$, given $C = \{c_1, \cdots, c_m\}$ and $S = \{s_1, \cdots, s_n\}$ the set of positive characters and species of matrix M_e, while E is defined as follows: $(s, c) \in E$ is a black edge if and only if $M_e[s, c] = 1$ and $M_e[s, \bar{c}] = 0$.

Realization of a character c and its canonical completion

Let $\mathcal{C}(c)$ be the connected component of graph G_{RB} containing node c. The *realization of character* c in graph G_{RB} consists of:

1. adding red edges connecting character c to all species nodes s that are in $\mathcal{C}(c)$ and such that (c, s) is not an edge of G_{RB},
2. removing all black edges (c, s) in graph G_{RB}. Then c is labelled *active.*
3. if an active character c' is connected by red edges to all species of that are in $\mathcal{C}(c')$, then its outgoing red edges are deleted from the graph and c is labelled c' *free.*

The realization of a character c is associated with a *canonical completion* of character c in matrix M_e that is defined by completing each pair $(?,?)$ occurring in the pair $(c,\bar{c})$ as follows: the pair $(?,?)$ is completed by $(1,1)$ in every species s that is in the connected component $\mathcal{C}(c)$ of graph G_{RB}, (s is connected with c by a red edge) while value $(0,0)$ is assigned in the remaining rows. We call *e-empty* a red-black graph without edges. Since we are interested in computing canonical completions of M_e that admit a pp tree, only canonical completions that are obtained by the realization of special sequences of characters of the red-black graph are considered, as defined below.

Definition 1. *Given a graph G_{RB} for an extended matrix M_e, a* successful reduction *of G_{RB} is an ordering $r =< c_{i_1}, \cdots, c_{i_m} >$ of the set of all positive characters of M_e such that the consecutive realization of each character in r leaves an e-empty red-black graph.*

In [2] we show that finding a solution to an instance of the IP-PP problem is equivalent to computing the existence of a successful reduction for the red-black graph for the input matrix. Furthermore by the Theorem 1 a solution to the IP-PP instance M_e is equivalent to a solution to the P-PP instance M.

Theorem 2. *Let M_e be an extended matrix. Then M_e admits a perfect phylogeny, if and only if there exists a successful reduction of the graph G_{RB} for M_e.*

We propose an algorithm, called **Decide-pp-opt**, for the P-PP problem that is based on Theorems 1 and 2. It builds a decision tree that explores all permutations of the set C of characters of M_e in order to find one that is a successful reduction, if it exists.

The following result is a consequence of two technical Lemmas that are omitted for lack of space.

Theorem 3. *Let M be a binary matrix that has an e-empty conflict graph. Then matrix M admits a persistent perfect phylogeny and there exists a polynomial time algorithm to build the p-pp tree for M.*

We give a polynomial time algorithm, in the size of the input matrix M, to find a successful reduction of graph G_{RB}, thus showing that a p-pp tree for M always exists. Given c, c' columns of M, then $c < c'$ if and only if for each species s of M, it holds that $M[s,c] \leq M[s,c']$. Then given M a binary matrix, the *partial order graph* for M is the partial order P obtained by ordering columns of M under the $<$ relation.

The algorithm constructs the partial order graph P for M. Then it iterates the following step to build a successful reduction r: - add to sequence r all element in the set C_M consisting of the maximal ones in P. Remove characters C_M from P.

Furthermore we propose a optimized version of the exact algorithm presented in [2] that uses the polynomial time algorithm for an e-empty conflict graph.

Algorithm Decide-pp-opt(M, M', x, T)

Input: a binary matrix M of size $n \times m$, a partial depth-first visit tree T of the decision tree $\mathcal{T}$ and a leaf node x of $\mathcal{T}$, a partial completion M' of the extend matrix M_e obtained by the realization of the characters labelling a path π from r to node x of the tree T;

Output: the tree T extended with the depth-first visit of T from node x. The procedure eventually outputs a successful reduction r or fails to find such a successful reduction.

- Step 1: if the incident edge to node x is labelled c, then realize c in G_{RB} and complete the pair of columns (c, c') in M'. If the matrix M' has a forbidden matrix, then label x as a fail node.
- Step 2: compute the conflict graph G_c for the matrix M updated after the realization of the characters along the path π from the root r to node x (i.e. M is obtained after eliminating the rows that correspond to species-nodes that are singletons in G_{RB}),
- Step 3: if the conflict graph G_c is e-empty, then apply the polynomial-time algorithm for an empty conflict-graph and return a successful reduction. Else for each node x_i that is a child of node x in tree T and is labelled by a non-active character in G_{RB}, apply Decide-pp-opt(M, M', x_i, $T \cup \{x_i\}$).

The algorithm **Decide-pp-opt** has been implemented and tested over simulated data produced by the tool *ms* by Hudson [7]. We have implemented the algorithm in C++ and the experiments have been run on a standard Windows workstation with 4 GB of main memory.

Table 1 reports the computation time to solve sets of 50 matrices for each dimension $(50, 15)$, $(100, 15)$, $(200, 15)$, and $(500, 15)$ with a recombination rate 1 over 15. The sets contain only matrices that are solved within 5 minutes. Another experiment has been done with 10 matrices of the same size 50×15 and different number of edges in the conflict graph. The average time was 0.015, 0.031 and 0.051, respectively for the case of $1, 5$ and 10 conflicts. Clearly, the number of unsolved matrices increases with the size of the input matrices but also with the number of conflicts that are present in the conflict graph. In order to test the performance of the algorithm for large matrices in terms of number of species we have processed a matrix of size 1000×15 with a conflict graph having 9 conflicts (edges). It took 35.5 seconds to find the solution to the matrix. We also compared the execution times of the exact algorithm and the optimized algorithm on sets of matrices with fixed number of columns and different numbers of rows. The **Decide-pp-opt** algorithm is able to find a solution for all matrices in contrast to the **Decide-pp** algorithm that in some cases takes more than 10 minutes to find a solution for a single matrix.

References

1. P. Bonizzoni. A linear time algorithm for the Perfect Phylogeny Haplotype problem. *Algorithmica*, 48(3):267–285, 2007.
2. P. Bonizzoni, R. Dondi, C. Braghin, and G. Trucco. The persistent perfect phylogeny model. *Theoretical Computer Science*, to appear, 2012.

nxm	no P-PPH	tot conflicts	average conflicts	solved matrices		total time in s		average time in s	
				alg.	opt. alg.	alg.	opt. alg.	alg.	opt. alg.
50x15	6	236	4.72	47	50	89.12	32.32	1.90	0.65
100x15	4	175	3.5	48	50	436.02	194.63	9.08	3.89
200x15	3	147	2.94	48	50	1583.50	43.21	32.99	0.86
500x15	7	219	4.38	44	50	888.59	889.43	20.20	17.79

Table 1. The table has entries to specify the average time to solve a single matrix (in seconds shortened as s), the number of matrices that do not admit a p-pp tree, the total number of conflicts, measured as the number of edges in the graph G_c of the matrices of each set, and the average number of conflicts. Each considered matrix has a conflict graph G_c that consists of a single non trivial component.

3. Z. Ding, V. Filkov, and D. Gusfield. A linear time algorithm for Perfect Phylogeny Haplotyping (pph) problem. *Journal of Computational Biology*, 13(2):522–553, 2006.
4. J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, 2004.
5. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, pages 19–28, 1991.
6. D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In *Proc. 6th Annual Conference on Research in Computational Molecular Biology (RECOMB 2002)*, pages 166–175, 2002.
7. R. R. Hudson. Generating samples under a wright-fisher neutral model of 31 genetic variation. *Bioinformatics*, 18(2):337–338, 2002.
8. I. Peer, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *Siam Journal on Computing*, 33(3):590–607, 2004.
9. T. M. Przytycka. An important connection between network motifs and parsimony models. In *Proc. 10th Annual Conference on Research in Computational Molecular Biology (RECOMB 2006)*, pages 321–335, 2006.
10. V. Satya and A. Mukherjee. An optimal algorithms for perfect phylogeny haplotyping. *Journal of Computational Biology*, 13(4):897–928, 2006.

On the Complexity of the Swap Common Superstring Problem

Paola Bonizzoni[2], Riccardo Dondi[1], Giancarlo Mauri[2], and Italo Zoppis[2]

[1] DSLCS, Università degli Studi di Bergamo, Bergamo - Italy
[2] DISCo, Università degli Studi di Milano-Bicocca, Milano - Italy
bonizzoni@disco.unimib.it, riccardo.dondi@unibg.it,
mauri@disco.unimib.it, zoppis@disco.unimib.it

Abstract. In several areas, in particular in bioinformatics, Shortest Common Superstring problem (SCS) and variants thereof have been successfully applied for strings comparison. In this paper we consider a variant of SCS that have been recently introduced, Swapped Common Superstring (SWCS), and we investigate its complexity. We show that SWCS is APX-hard even when the input strings have length bounded by a constant (equal to 10) or are over a binary alphabet.

1 Introduction

In several areas, such as bioinformatics [5], the Shortest Common Superstring problem (SCS) has been successfully applied for strings comparison [2,6,7,8]. Recently, some variants of the SCS problem have been proposed to deal with problems in bioinformatics and AI planning [4,3]. In this paper we consider one of these variants, the Swapped Common Superstring (SWCS) problem, where, given a set S of strings over an alphabet Σ and a text $\mathcal{T}$ over Σ, we look for a swap ordering $\mathcal{T}'$ of $\mathcal{T}$ (an ordering of $\mathcal{T}$ obtained by swapping only some pairs of adjacent characters) such that the number of input strings that are substrings of $\mathcal{T}'$ is maximized.

The SWCS is known to be NP-hard [4], while a relaxed version of the problem, where each occurrence of a string in the swap ordering $\mathcal{T}'$ is counted, is polynomial time solvable [4].

We investigate the complexity of SWCS under two natural parameters, i.e. the length of the input strings and the size of the alphabet. We show that SWCS is APX-hard even when the input strings have length bounded by a constant (equal to 10) or they are over a binary alphabet.

Now, we define the problem formally. Given a string s and a substring s_x of s, we say that s_x is *covered* by s.

Problem 1. [4] SWCS
Input: a set $S = \{s_1, \ldots, s_n\}$ of strings over alphabet Σ, a text $\mathcal{T} = t_1 t_2 \ldots t_m$, where each t_i, $1 \leq i \leq m$, is a character in Σ.
Output: an ordering $\mathcal{T}'$ of the text $\mathcal{T}$ (called a *swap ordering of* $\mathcal{T}$) that maximizes the number of strings in S that are covered by $\mathcal{T}'$, where $\mathcal{T}'$ is induced

by a permutation $\pi : \{1,\dots,m\} \to \{1,\dots,m\}$ such that: (1) if $\pi(i) = j$, then $\pi(j) = i$, (2) for all i, $\pi(i) \in \{i-1, i, i+1\}$, (3) if $\pi(i) \neq i$ then $t_{\pi(i)} \neq t_i$.

The definition given above establishes that a swap ordering $\mathcal{T}'$ of $\mathcal{T}$ is obtained by swapping only some pairs of adjacent distinct characters in $\mathcal{T}$.

2 Complexity of SWCS for Bounded Length and Alphabet

In this section, we consider two restrictions of SWCS, namely the case when the input strings length is bounded by 10 (denoted by $10-$SWCS) and the case when the input strings are over a binary alphabet (denoted by SWCS(2)). We show that both these restrictions are APX-hard.

In order to prove that $10-$SWCS and SWCS(2) are APX-hard, we present two L-reductions from the Maximum Independent Set on Cubic Graphs (MAX-ISC). We recall that a graph is cubic when each of its vertices has degree three. Given a cubic graph $G = (V, E)$, with $V = \{v_1, \dots, v_q\}$, MAX-ISC asks for a set $V' \subseteq V$ of maximum cardinality, such that for each $v_i, v_j \in V'$, it holds $\{v_i, v_j\} \notin E$. MAX-ISC is known to be APX-hard [1].

2.1 APX-hardness of $10-$SWCS

We show that $10-$SWCS is APX-hard, giving a reduction from MAX-ISC. Let $G = (V, E)$ be a cubic graph, with $V = \{v_1, \dots, v_q\}$. In what follows, given a vertex $v_i \in V$, denote by v_j, v_h, v_l the three vertices of G adjacent to v_i. Next, we define an instance $(S, \mathcal{T})$ of $10-$SWCS associated with G. The alphabet Σ over which the strings in S range, is defined as follows: $\Sigma = \{w_i, x_i : v_i \in V\} \cup \{a_{i,j} : \{v_i, v_j\} \in E\} \cup \{y\}$.

The set S of input strings is defined as follows: $S = \bigcup_{i:v_i \in V}(I_{i,1} \cup I_{i,2} \cup I_{i,3})$, where $I_{i,1}$, $I_{i,2}$, $I_{i,3}$, with $v_i \in V$, are three sets of strings defined as follows: $I_{i,1} = \{w_i a_{i,j} w_j, w_i a_{i,h} w_h, w_i a_{i,l} w_l\}$, $I_{i,2} = \{w_i x_i a_{i,j}, w_i x_i a_{i,h}, w_i x_i a_{i,l}\}$, $I_{i,3} = \{x_i w_i a_{i,j} w_j x_i w_i a_{i,h} w_h x_i w_i\}$.

Now, we define the text $\mathcal{T}$. For each v_i let $\mathcal{T}_i$ be the following string:

$$\mathcal{T}_i = w_i x_i a_{i,j} w_j w_i x_i a_{i,h} w_h w_i x_i a_{i,l} w_l$$

Then, $\mathcal{T}$ is defined as follows:

$$\mathcal{T} = \mathcal{T}_1 yyy \mathcal{T}_2 \dots yyy \dots yyy \mathcal{T}_n$$

Given a swap ordering $\mathcal{T}'$ of $\mathcal{T}$, we denote by $\mathcal{T}_i'$ the swap ordering in $\mathcal{T}'$ of the substring $\mathcal{T}_i$ of $\mathcal{T}$, with $1 \leq i \leq q$. We say that $\mathcal{T}_i'$ has a *configuration a*, if each pair w_i, x_i of $\mathcal{T}_i$ is swapped in $\mathcal{T}_i'$, that is $\mathcal{T}_i' = x_i w_i a_{i,j} w_j x_i w_i a_{i,h} w_h x_i w_i a_{i,l} w_l$. $\mathcal{T}_i'$ has a *configuration b* if no pair is swapped in $\mathcal{T}_i$, that is $\mathcal{T}_i'$ is identical to $\mathcal{T}_i$.

Next, we can show the following property of a swap ordering $\mathcal{T}'$ of $\mathcal{T}$.

Lemma 1. *Given a swap ordering $\mathcal{T}'$ of $\mathcal{T}$, we can compute in polynomial time a swap ordering $\mathcal{T}''$ of $\mathcal{T}$ such that $\mathcal{T}''$ covers at least as many input strings as*

$\mathcal{T}'$, and such that, denoted as $\mathcal{T}_i''$ the swap ordering in $\mathcal{T}''$ of the substring $\mathcal{T}_i$ of $\mathcal{T}$, it follows that each $\mathcal{T}_i''$ has either a configuration a *or a* configuration b.

Notice that a *configuration a* of $\mathcal{T}'_i$ allows to cover 4 input strings, namely the strings in $I_{i,1} \cup I_{i,3}$, while a *configuration b* of $\mathcal{T}'_i$ allows to cover 3 input strings, namely the strings in $I_{i,2}$. A *configuration a* of $\mathcal{T}'_i$ corresponds to a vertex v_i in an independent set of G, while a *configuration b* of $\mathcal{T}'_i$ corresponds to a vertex v_i in a vertex cover of G. Lemma 2 is a consequence of this observation and of Lemma 1.

Lemma 2. *Let $G = (V, E)$ be an instance of MAX-ISC and let $(S, \mathcal{T})$ be the corresponding instance of* $10 - \mathrm{SWCS}$*. Then, there is an independent set of G of size p if and only if there is a swap ordering of $\mathcal{T}$ that covers* $4p + 3(q - p)$ *input strings.*

A direct consequence of Lemma 2 is the following theorem.

Theorem 1. *The* $10 - \mathrm{SWCS}$ *problem is APX-hard.*

2.2 APX-hardness of SWCS(2)

In this section we show that SWCS(2) is APX-hard. Let $G = (V, E)$ be a cubic graph, with $V = \{v_1, \dots, v_q\}$. For each $v_i \in V$, denote by $\{v_i, v_j\}$, $\{v_i, v_h\}$, $\{v_i, v_k\}$ the three edges incident on v_i. The input text $\mathcal{T}$ consists of $2q - 1$ substrings. More precisely, $\mathcal{T}$ contains a substring $B(v_i)$ for each $v_i \in V$. These substrings are separated by $q - 1$ identical substrings, denoted as SE:

$$\mathcal{T} = SE \cdot B(v_1) \cdot SE \cdot B(v_2) \cdot \dots SE \cdot B(v_q) \cdot SE$$

Each substring SE is defined as follows: $SE = 1111100000$. Now, given a vertex $v_i \in V$, we define the associated substring $B(v_i)$. In order to define $B(v_i)$, we have to introduce some notations. First, given a substring s of size 5, s has an *inactive configuration*, denoted as $I(s)$, if $s = 00000$, s has a *positive configuration*, denoted as $P(s)$, if $s = 00100$, s has a *negative configuration*, denoted as $N(s)$, if $s = 01000$.

Now, we introduce three strings that will be used to define $B(v_i)$. Given a vertex $v_i \in V$, let $s(v_i)$ be a string of length 5, which can have an *inactive* configuration $I(s(v_i))$, a *positive* configuration $P(s(v_i))$ or *negative* configuration $N(s(v_i))$. Now, we define the following strings:

$$B(v_i, e_{i,j}) = I(s(v_1)) \cdot I(s((v_2)) \dots P(s(v_i)) \dots P(s(v_j)) \cdots I(s(v_q))$$

$$B(v_i, e_{i,h}) = I(s(v_1)) \cdot I(s(v_2)) \dots P(s(v_i)) \dots P(s(v_h)) \dots I(s(v_q))$$

$$B(v_i, e_{i,l}) = I(s(v_1)) \cdot I(s(v_2)) \dots P(s(v_i)) \dots P(s(v_l)) \cdots I(s(v_q))$$

The substring $B(v_i)$ is defined as follows: $B(v_i) = B(v_i, e_{i,j}) \cdot B(v_i, e_{i,h}) \cdot B(v_i, e_{i,l})$.

Given a swap ordering $\mathcal{T}'$ of $\mathcal{T}$, we denote by $B'(v_i)$ ($B'(v_i, e_{i,j})$ respectively) the swap ordering of $B(v_i)$ ($B(v_i, e_{i,j})$ respectively) in $\mathcal{T}'$. A string $B'(v_i, e_{i,j})$, with $\{v_i, v_j\} \in E$, has a *positive configuration* in $\mathcal{T}'$ if both $s(v_i)$ and $s(v_j)$

have positive configurations in $\mathcal{T}'$; $B'(v_i, e_{i,j})$, with $\{v_i, v_j\} \in E$, has a *negative configuration* in $\mathcal{T}'$ if both $s(v_i)$ and $s(v_j)$ have negative configurations in $\mathcal{T}'$.

Now, we define the set S of input strings. For each edge $\{v_i, v_j\}$, S contains an input string:

$$s_{i,j} = I(s(v_1)) \dots N(s(v_i)) \dots N(s(v_j)) \cdots I(s(v_q))$$

Furthermore, for each $v_i \in V$, S contains the following two input strings $s'_i = 00000 \cdot B(v_i) \cdot SE$, $s''_i = SE \cdot B(v_i) \cdot 11111$.

First, we will show a property of the instance $(S, \mathcal{T})$ of SWCS.

Lemma 3. *Let $(S, \mathcal{T})$ be an instance of* SWCS*, then a swap ordering $\mathcal{T}'$ of $\mathcal{T}$ can cover in each substring $B'(v_i)$ either a string $s \in \{s'_i, s''_i\}$, or a string $s_{i,j}$, for some $\{v_i, v_j\} \in E$.*

Now, consider an order $\mathcal{T}'$ of $\mathcal{T}$. It can be shown that either $B'(v_i, e_{i,j})$, $B'(v_i, e_{i,h})$, $B'(v_i, e_{i,k})$ have all negative configurations (in this case $B'(v_i)$ covers the strings $s_{i,j}$, $s_{i,h}$, $s_{i,k}$ and corresponds to a vertex in an independent set of G) or $B'(v_i, e_{i,j})$, $B'(v_i, e_{i,h})$, $B'(v_i, e_{i,k})$ have all positive configurations (in this case $B'(v_i)$ covers the strings s'_i, s''_i and corresponds to a vertex in a vertex cover of G).

Lemma 4 follows from this observation and from Lemma 3.

Lemma 4. *Let $G = (V, E)$ be an instance of MAX-ISC and let $(S, \mathcal{T})$ be the corresponding instance of* SWCS(2)*. Then, there is an independent set of G of size p if and only if there is a swap ordering of $\mathcal{T}$ that covers $3p + 2(q - p)$ input strings.*

A direct consequence of Lemma 4 is the following theorem.

Theorem 2. *The* SWCS(2) *problem is APX-hard.*

References

1. P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoretical Comput. Sci.*, 237(1–2):123–134, 2000.
2. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J. ACM*, 41(4):630–647, 1994.
3. R. Clifford, Z. Gotthilf, M. Lewenstein, and A. Popa. Restricted common superstring and restricted common supersequence. In *CPM 2011*, pages 467–478, 2011.
4. Z. Gotthilf, M. Lewenstein, and A. Popa. On shortest common superstring and swap permutations. In *SPIRE 2010*, pages 270–278, 2010.
5. D. Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.
6. S. Ott. Lower bounds for approximating shortest superstrings over an alphabet of size 2. In *WG 1999*, pages 55–64, 1999.
7. Z. Sweedyk. A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM J. Comput.*, 29(3):954–986, 1999.
8. V. Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In *MFCS 2005*, pages 793–800, 2005.

Moore Automata and Epichristoffel Words

Giusi Castiglione and Marinella Sciortino

Dipartimento di Matematica e Informatica
Università di Palermo, via Archirafi, 34 - 90123 Palermo, Italy
{giusi, mari}@math.unipa.it.

In recent years (cf. [1, 3, 4]) a bridge between combinatorics on words and the study of complexity of algorithms for the minimization of finite state automata (*acceptors*) has aroused great interest. In particular, the study of combinatorial properties of *christoffel classes* (or *circular sturmian words*) allowed to prove that Hopcroft's minimization algorithm becomes not ambiguous when applied to the family of cyclic unary acceptors constructed by circular sturmian word. Furthermore, for particular subfamilies the tightness is obtained.

In the field of combinatorics on words, increasing the cardinality of the letters alphabet can give rise to new problematic questions. For example, for *episturmian words* (cf. [5]) representing an extension to a larger alphabet of the notion of infinite sturmian word, many of the crucial properties of such a family of words, as the balancing, are lost. These considerations also hold when finite combinatorial objects as circular words are considered.

We denote by (w) the circular word over the k-ary alphabet A corresponding to all the conjugates of the word w. Given a circular word (w) a factor u is called m-special if there exist exactly m distinct characters $a_1, a_2, \ldots, a_m$ in the alphabet A, such that all ua_i are factors of (w) for each $i = 1, \ldots, m$. Some families of circular words are able to capture many properties of classes of infinite words. For instance, in the binary case, circular sturmian words inherits the balancing from infinite sturmian words. Moreover each circular sturmian word (w) admits a unique 2-special factor for each length up to $|w| - 2$. In [4, 2] other structural characterizations have been also investigated.

Circular epichristoffel words, introduced in [8], are circular words that maintain some structural properties of episturmian words. More formally, we say that (w) is a circular epichristoffel word if it is the image of a letter by an episturmian morphism. One can prove that, for each length up to $|w| - 2$, there exists a unique special factor. In case of k-ary alphabet the problem of determining for each $2 \leq m \leq k$ the maximal length of all m-special factors can be investigated. Several properties of circular sturmian words can not be extended to circular epichristoffel words and there are a lot of open problems connected to such a family. Furthermore, the study of such a class seems to be connected to Fraenkel conjecture.

In this paper we deal with the question of how the process of minimization of a *Moore automaton* (cf. [7]) is influenced by the problems arising in combinatorics on words when alphabets of size greater than 2 are considered. Note that for such automata, differently from acceptors, the output alphabet is not binary. In particular, we analyze the behavior of a variant of Hopcroft's algorithm on a family of unary cyclic Moore automata associated to circular epichristoffel words and we relate the minimization process with particular factorization properties, here introduced, of such words.

Given $p = (p_1, p_2, \ldots, p_k)$ a k-tuple of non-negative integers, in [8] the author gives an algorithm to determine whether a circular epichristoffel word, having p as vector of occurrences of the letters, there exists and a construction is shown. All the steps of the construction determine a sequence of letters, called *directive sequence*, used to construct the circular epichristoffel word.

We prove that each letter a_i of a k-ary alphabet A uniquely determines a circular factorization of a circular epichristoffel word (w) defined over A in a set X_{a_i} containing k circular epichristoffel words. Such a factorization is induced by the directive sequence. Let $z_{a_i}(w)$ be the circular word obtained from (w) by encoding by $a_1, a_2, \ldots, a_k$ the occurrences of the correspondent elements of X_{a_i}. Let us denote by $(i(w))$ the circular epichristoffel word obtained by permuting the letters of (w) such that the associated k-tuple is not increasing, i.e. $p_1 \geq p_2 \geq \ldots \geq p_k$. We prove that the circular word $(i(z_{a_i}(w))$, denoted by $L_{a_i}(w)$, is a circular epichristoffel word. Therefore, we can associate to each epichristoffel word (w) a k-ary tree $\tau(w)$, called *reduction tree*, defined as follows.

– If w is a single letter a_i, $\tau(w)$ is a single node labeled by $(i(a_i)) = (a_1)$.
– If $|w| > 1$, $\tau(w)$ is a tree with root labeled by $(i(w))$ and at most k subtrees. The i-th subtree is $\tau(L_{a_i}(w))$.

Figure 1 shows an instance of reduction tree of a circular epichristoffel word and its correspondent factorizations. It is possible to prove that each circular epichristoffel word is uniquely determined by its reduction tree, as stated in the following theorem.

Theorem 1. *Let (w) and (w') be two circular epichristoffel words over the alphabet $A = \{a_1, \ldots, a_k\}$. Then, $\tau(w) = \tau(w')$ if and only if $(w') = (w)$ (up to a permutation of the letters).*

Let $(w) = (a_1 a_2 \ldots a_n)$ be a circular word over the alphabet A. The *cyclic automaton associated to* (w), denoted by $\mathcal{A}_w$, is a particular deterministic Moore automaton (DMA) $\mathcal{A} = (\Sigma, A, Q, q_0, \delta, \lambda)$ in which $Q = \{1, 2, \ldots, n\}$ is the set of states, $\Sigma = \{0\}$ is the input alphabet, A is the output alphabet, δ is the transition function defined as $\delta(i, 0) = (i + 1)$, $\forall\, i \in Q \setminus \{n\}$ and $\delta(n, 0) = 1$. The choice of q_0 does not affect the minimization process. Moreover, $\lambda : Q \mapsto \Gamma$ is a *output function* that assigns an output to the states of the automaton here defined as $\lambda(i) = a_i$ for each $i \in Q$. See Figure 2(a) for an example.

In this paper we propose a minimization strategy (called L-Minimization algorithm) for DMA that is variant of Hopcroft's minimization algorithm, the most efficient known minimization algorithm for acceptors (cf. [6]) that runs in time $O(n \log n)$. It can operate on a generic deterministic Moore automaton and it is based on two main ingredients. The first one is the notion of m-split operation, defined as follows. Given a partition Π of Q, let $\mathbb{C} \subset \Pi$, we say that $(\mathbb{C}, a)$ *m-splits* the class B if there exist $\{Q_1, Q_2, \ldots, Q_{m-1}\} \subset \mathbb{C}$ such

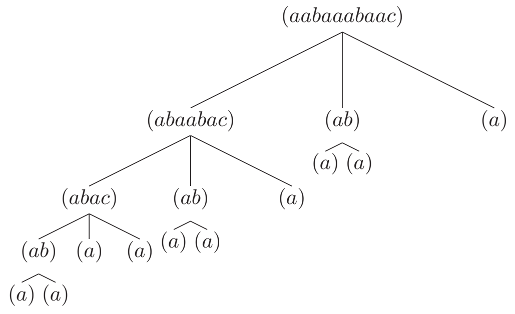


Fig. 1: The reduction tree $\tau(aabaaabaac)$. The circular word $(aabaaabaac)$ can be circularly factorized into circular epichristoffel words, as follows: $(w) = (a)(ab)(a)(a)(ab)(a)(ac)$, $(w) = (aacaab)(aaab)$, $(w) = (aabaaabaac)$. Such factorizations are coded by the circular epichristoffel words $(abaabac)$, (ca) and (c), respectively. Consequently, $L_a(w) = (abaabac)$, $L_b(w) = (ab)$, $L_c(w) = (a)$. Analogously, the other factorizations can be determined.

that $\delta_a^{-1}(Q_i) \cap B \neq \emptyset$ and $B \not\subseteq \delta_a^{-1}(Q_i)$, with $i = 1, \ldots, m-1$. In this case the set B can be, obviously, split into $B_i = \delta_a^{-1}(Q_i) \cap B$, with $i = 1, \ldots, m-1$ and $B_m = B \setminus \bigcup_{i=1,\ldots,m-1} B_i$. The pairs $(\mathbb{C}, a)$ are stored and successively extracted from an auxiliary data structure $\mathcal{W}$, called *waiting set*. The second ingredient is the *all but not the largest* strategy instead of *smaller half* strategy of the classical Hopcroft's minimization algorithm. In particular, we store into the waiting set $\mathcal{W}$ all but not the largest sets obtained from the $m - split$ as the $(m-1)$-tuple $(\mathbb{C}, a)$. Such a strategy is applied throughout the algorithm, starting with the first step in which the set Q of states is split into classes of states having the same output function λ and this is fundamental in order to obtain the minimal automaton. The successive split operations and insertions into $\mathcal{W}$ could be execute by the smaller-half strategy where 2-splits can occur. Also in this case the minimal automaton is produced. Although, in general, L-minimization is not deterministic, we show that there is an infinite family of automata for which, differently from smaller-half strategy, it has a unique execution, as stated in the following theorem.

Theorem 2. *The execution of* L-Minimization *algorithm on cyclic automata associated to circular epichristoffel words is unique.*

Such a result can be proved by using the fact that a m-split occurs in correspondence with m-special factor of the circular word.

However, it is possible to verify that there exist infinite families of Moore automata for which the executions of our algorithm are better than some executions of the smaller-half minimization strategy and vice-versa. It could be interesting to study the tightness and the average time complexity of the two methods.

The refinement process produced during each execution of the algorithm on a generic Moore automaton $\mathcal{A}$, can be represented by a k-ary tree $\mathcal{T}(\mathcal{A})$, also

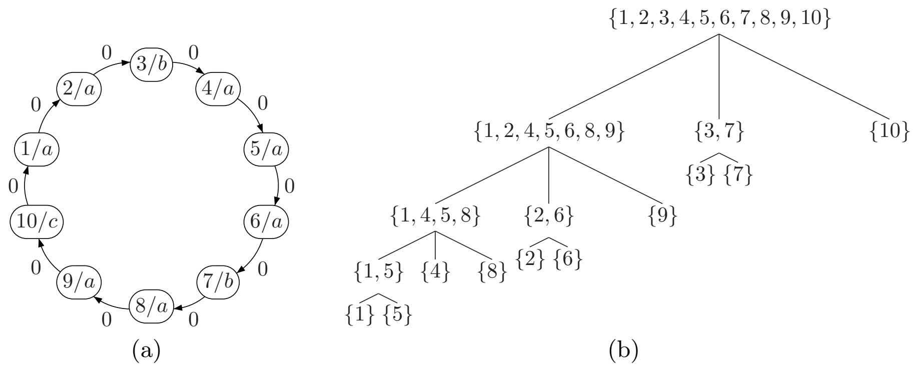


Fig. 2: (a) Cyclic automaton $\mathcal{A}_w$ for $(w) = (aabaaabaac)$; $\Sigma = \{0\}$, $A = \{a, b, c\}$, $\lambda(1) = \lambda(2) = \lambda(4) = \lambda(5) = \lambda(6) = \lambda(8) = \lambda(9) = a$, $\lambda(3) = \lambda(7) = b$, $\lambda(10) = c$. (b) The derivation tree $\mathcal{T}(\mathcal{A}_w)$.

called *derivation tree*, whose nodes are labeled by classes of the partitions and their descendants are the classes produced by the m-split operations. See Figure 2(b) for an example. Note that, in general, the shape of the derivation tree is strongly affected from the non-deterministic choices of minimization algorithm, although the leaves are the same.

The following theorem establishes a relationship between derivation trees and reduction trees.

Theorem 3. *If (w) is a circular epichristoffel word then $\mathcal{T}(\mathcal{A}_w)$ and $\tau(w)$ are isomorphic.*

References

1. J. Berstel, L. Boasson, and O. Carton. Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm. *Theor. Comput. Sci.*, 410:2811–2822, 2009.
2. J.-P. Borel and C. Reutenauer. On christoffel classes. *ITA*, 40(1):15–27, 2006.
3. G. Castiglione, A. Restivo, and M. Sciortino. Hopcroft's algorithm and cyclic automata. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *LATA*, volume 5196 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2008.
4. G. Castiglione, A. Restivo, and M. Sciortino. Circular sturmian words and Hopcroft's algorithm. *Theor. Comput. Sci.*, 410(43):4372–4381, 2009.
5. A. Glen and J. Justin. Episturmian words: a survey. *ITA*, 43(3):403–442, 2009.
6. J.E. Hopcroft. An $n \log n$ algorithm for mimimizing the states in a finite automaton. In *Theory of machines and computations (Proc. Internat. Sympos. Technion, Haifa, 1971)*, pages 189–196. Academic Press, New York, 1971.
7. E.F. Moore. *Gedaken experiments on sequential machines*, pages 129–153. Princeton University Press, 1956.
8. G. Paquin. On a generalization of christoffel words: epichristoffel words. *Theor. Comput. Sci.*, 410(38-40):3782–3791, 2009.

Data mining for a student database

R. Campagni, D. Merlini, and R. Sprugnoli

Dipartimento di Sistemi e Informatica
viale Morgagni 65, 50134, Firenze, Italia
[renza.campagni,donatella.merlini,renzo.sprugnoli]@unifi.it

1 Introduction

Educational data mining is an emerging research area that produces useful, previously unknown issues from educational database for better understanding and improving the performance and assessment of the student learning process (see [2] and the included references, for a detailed description of the state of the art in this context). This paper presents some data mining models to analyze the *careers* of University students and extends the research illustrated in [1], introducing a new approach to this research area. The career of a student can be analyzed from various points of view, among which the following two are particularly important: i) the perspective of the student, who evaluates how difficult and important an exam is, in order to decide to take it immediately at the end of the course, or delay it as much as possible; this aspect is studied in Section 2 with cluster and classification algorithms by introducing a notion of distance between careers; and ii) the perspective of each course, by analyzing the distribution of students with respect to the delay with which they take an examination, to discover common characteristics between two or more courses; this is done in Section 3 in terms of Poisson distributions.

2 The perspective of the student

The methodology we propose is based on a database containing information about students and their exams in a University organization. In particular, for each student, the database contains general information such as the sex, the place of birth, the grade obtained at the high school level, the year of enrollment at the university, the date and the grade of final examination besides information about each exam, that is, the identifier of the exam, the date and the grade. We refer to an organization of the university which allows students to take an exam in different sessions after the end of the course, as in Italy. Some constrains between exams can be fixed in order to force students to take some exams in a specific order, however, usually students have many degrees of freedom to choose their own order of exams. An important information which is a basilar aspect of our methodology is the *semester*; an academic year is divided into two semesters, during which the courses are taken according to the established curriculum. A student can take an exam in the same semester of the course, that is just after the end of the course, or later, with a delay of one or more semesters. This information allows us to define an *ideal path* to be compared with the path of a generic student. More precisely, we consider a database containing the data of N students, each student characterized by a sequence of n exams identifiers and a particular path $\mathcal{I} = (e_1, e_2, \cdots, e_n)$, the *ideal path*[1], corresponding to the ideal student who has taken every examination just after the end of the corresponding course, without delay. Without loss of generality, we can assume that $e_i = i$, $i = 1, \cdots, n$, that is, $\mathcal{I} = (1, 2, \cdots, n)$. The path of a generic student k with $k = 1, \cdots, N$, can be seen as a sequence $\mathcal{S}_k = (e_{\pi_k(1)}, e_{\pi_k(2)}, \cdots, e_{\pi_k(n)})$ of n exams, where $e_{\pi_k(i)}$, $i = 1, \cdots, n$, is the identifier of the exam taken by the student k at time i and π_k indicates the corresponding permutation of $1, \cdots, n$. Therefore, $\mathcal{S}_k$ can be seen as a permutation of the integers 1 through n. The idea is to understand how the order of the exams affects the final result of students. To this purpose, we compare a path $\mathcal{S}_k$ with $\mathcal{I}$ by using the *Bubblesort distance*, which is defined as the number of exchanges performed by the Bubblesort algorithm to sort an array containing the numbers from 1 to n. The number of exchanges can be computed easily since it corresponds exactly to the number of inversions in the permutation. Given a permutation $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$ of the integers 1 through n, an *inversion* is a pair $i < j$ with $\pi_i > \pi_j$. If q_j is the number of $i < j$ with $\pi_i > \pi_j$ then $q = (q_1, q_2, \cdots, q_n)$ is called the *inversion table* of π. We use the notation $\sigma(\pi)$ to denote the number of inversions in the permutation, that is, the sum of the entries in the inversion table: $\sigma(\pi) = \sum_{j=1}^{n} q_j$. For example, the permutation $\pi = (5, 2, 3, 1, 4)$ corresponds to $q = (0, 1, 1, 3, 1)$ and $\sigma(\pi) = 6$.

The path $\mathcal{S}_k$ of a generic student k can be compared with the ideal path $\mathcal{I}$ by computing $\sigma(\mathcal{S}_k)$, $k = 1, \cdots, N$. After this preprocessing phase, we can assume that for each student our database contains at least the following information: the graduation time, `Time`, the final grade, `Vote`, and the `Bubblesort` distance, together with other personal information. We can proceed by applying a cluster algorithm, for example `K-means`[2] (see e.g, [3]). If the cluster algorithm splits the students into K well defined groups characterized by similar Bubblesort distance, we can infer important conclusions about students and the laurea degree. We observe explicitly that students who have taken the exams in the same order, that is, students with the same path, can have different final grade and graduation time. The idea is to understand if there exists a relation between the Bubblesort distance and the success of students. If the students having small distance achieve good performance, then we may conclude that the academic degree is well structured but if there exist many good students with large distances, then the organization should probably be modified. We can extend our analysis

[1] Since in the same semester there are many courses, the ideal path is not unique. In this paper we sort courses relative to the same semester according to the preference of students. A different solution consists in giving the same identifier to courses in the same semester; for example, $(1, 1, 2, 2, 2, 3, 3)$ would represent a sequence of 7 exams, two in the first and third semester and three in the second.

[2] We wish to point out that the Bubblesort distance is an attribute inserted in our database and that we use `K-means` with the Euclidean distance.

through the technique based on decision trees. To this purpose, we need to add to the database a new attribute `Bubblesort_class` which labels the students into K different ways, according to the ranges of values of Bubblesort distance in the K clusters previously found. This new attribute can be used to classify students, for example by using the `C4.5` algorithm (see e.g, [3]). The aim is to classify students as talented or not and find the attributes which most influence their career. We can also try to classify with respect to other attributes: for example, we can predict whether a student has a long (short) career or obtains a high (low) final grade by introducing a `Time_class` or a `Vote_class` attribute in the database. The greater are the database and the information in it, the more accurate will result the model based on this technique.

The database we analyze contains data of students in Computer Science at the University of Florence beginning their career during the years 2001-2003 and graduated up to now. This academic degree is structured in three years, each divided in two semesters. In the years under consideration, no constrains between exams were fixed, so students could take their exams almost in any order. In particular, we analyzed the careers of $N = 100$ students characterized by a sequence of $n = 25$ exams. We computed the ideal path through an important pre-processing phase, which allowed us to identify the semester in which courses were originally hold. Then, for each student we computed the `Bubblesort` distance and added this value to the database. To understand how the order of the exams affects the career of the students, we have performed several tests by using the `K-means` implementation of `WEKA` (see, e.g., [4]). We obtained significant result with $K = 2$ by selecting as clustering attributes `Time`, `Vote` and `Bubblesort` distance. In fact, with these parameters we can see that students are well divided into two groups: students who graduated relatively quickly and with high grades and students who obtained worse results. Luckily, we observed that students in the first group are characterized by *small* values of `Bubblesort` while students in the second group have *larger* values. This result confirms that the more students follow the order taken by the ideal path, the more they obtain good performance in terms of graduation time and final grade. For what concerns classification, we applied the `C4.5` implementation of `WEKA` with different choices of attributes and class. The most interesting tree we obtained classifies students with respect to *small* (≤ 100) and *large* (> 100) values of Bubblesort distance confirming the result of clustering and, moreover, highlights that the results obtained at the high school influence the performance of students.

3 The perspective of the course: delayed exams

Usually, "good" students try to pass early every exam, but "not so good" students prefer to postpone most exams, especially if they are considered too difficult or too technical. We are interested in studying the delay distribution of every exam in the hypothesis that it is a good parameter for classifying students and/or courses. In general, delays conform to some Poisson distribution, with average (and variance) λ and probability mass function $P_\lambda(k) = e^{-\lambda} \cdot \lambda^k/k!$ for

$k \geq 0$. The Poisson distribution is discrete and, in our case, k represents the delay of the exam from the end of the course, measured in full years. So, if N is the number of students, $P_\lambda(0) \cdot N$ is the number of those who passed the exam within the first year; $P_\lambda(1) \cdot N$ are the students who passed during the second year, and so on. Finally, the distribution is unimodal and attains its maximum value at $k \approx \lambda$. If we look at the actual distributions of students with respect to the delay with which they took their examinations, we observe that most of them are bimodal, with a sharp peak at $k = 0$ and a second and smoother peak at $k = 2$ or $k = 3$. The obvious interpretation is that there are two different distributions, the first one relative to "good" students and the second relative to "not so good" students, who delay their exams of about two years. The two distributions are superimposed and generate the two peaks. In other words, by examining the distributions for each exam, we can infer that students are divided into two classes: students who tend to take an exam as soon as a course is terminated, and students who delay difficult exams to the end of their career. In order to analyze this behavior in a more formal way, we need to find the two Poisson distributions. We consider n courses $c_1, c_2, \cdots, c_n$ taken by N students and a database containing, for each course c_i, the number of students $D_{c_i}(k)$ which take the exam with delay k, for $k = 0, \cdots, d_i$, where d_i is the maximum delay relative to course c_i. We then use the following algorithm to determine the average values λ_g and λ_{ng} characterizing the two Poisson distributions and the corresponding numbers $N(\lambda_g)$ and $N(\lambda_{ng})$ of students. We can make the hypothesis that the λ_g-distribution decreases very fast so that it reduces to $k = 0, 1$ as meaningful values. Our first step consists in separating the first two values from the rest and try to approximate the λ_{ng}-distribution. We iterate this approximation process until a fixed point is obtained. This process can modify the values for $k = 0$ and $k = 1$, so that we have to use these new values to approximate the λ_g-distribution. Again, we proceed until a fixed point is found. The algorithm stops here returning, for each course, the two desired approximations.

We applied the algorithm to $n = 15$ courses taken by $N = 152$ students in Computer Science at the University of Florence. The analysis confirmed that for each course c_i we have $D_{c_i}(k) \sim P_{\lambda_{g_i}}(k) \cdot N(\lambda_{g_i}) + P_{\lambda_{ng_i}}(k) \cdot N(\lambda_{ng_i})$, with a good approximation. In particular, we found that Computer Science exams are characterized by $N(\lambda_g)/N \sim 70\%$. Instead, Mathematics exams are delayed and often appear as the last exams taken before the final examination.

References

1. R. Campagni, D. Merlini, and R. Sprugnoli. Analyzing paths in a student database. In *The 5th International Conference on Educational Data Mining*, 208–209, 2012.
2. C. Romero and S. Ventura. Educational Data Mining: A Review of the State of the Art. *IEEE Transactions on systems, man and cybernetics*, 40(6):601–618, 2010.
3. P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2006.
4. I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition.* Morgan Kaufmann, 2011.

A Fast Active Learning Algorithm for Link Classification[⋆]

Nicolò Cesa-Bianchi[1], Claudio Gentile[2], Fabio Vitale[3], and Giovanni Zappella[4]

[1] Dipartimento di Informatica, Università degli Studi di Milano, Italy
nicolo.cesa-bianchi@unimi.it

[2] Dipartimento di Scienze Teoriche ed Applicate, Università dell'Insubria, Italy
claudio.gentile@uninsubria.it

[3] Dipartimento di Informatica, Università degli Studi di Milano, Italy
fabio.vitale@unimi.it

[4] Dipartimento di Matematica, Università degli Studi di Milano, Italy
giovanni.zappella@unimi.it

Abstract. We present a very efficient active learning algorithm for link classification in signed networks. Our algorithm is motivated by a stochastic model in which edge labels are obtained through perturbations of an initial sign assignment consistent with a two-clustering of the nodes. We provide a theoretical analysis within this model, showing that we can achieve an optimal (to within a constant factor) number of mistakes on any graph $G = (V, E)$ such that $|E| = \Omega(|V|^{3/2})$ by querying $\mathcal{O}(|V|^{3/2})$ edge labels. More generally, we show an algorithm that achieves optimality to within a factor of $\mathcal{O}(k)$ by querying at most order of $|V| + (|V|/k)^{3/2}$ edge labels. The running time of this algorithm is at most of order $|E| + |V| \log |V|$.

1 Introduction

A rapidly emerging theme in the analysis of networked data is the study of signed networks. From a formal viewpoint, signed networks are graphs whose edges host a sign encoding the positive or negative nature of the relationship between the incident nodes. E.g., in a protein network two proteins may interact in an excitatory or inhibitory fashion. The domain of social networks and e-commerce offers several examples of signed relationships: Slashdot users can tag other users as friends or foes, Epinions users can rate other users positively or negatively, Ebay users develop trust and distrust towards sellers in the network. More generally, two individuals that are related because they rate similar products in a recommendation website may agree or disagree in their ratings.

The availability of signed networks has stimulated the design of link classification algorithms, especially in the domain of social networks. Early studies of signed social networks are from the Fifties. E.g., [6] and [1] model dislike and distrust relationships among individuals as (signed) weighted edges in a graph. The conceptual underpinning is provided by the theory of *social balance*, formulated as a way to understand the structure of conflicts in a network of individuals

[⋆] This work was supported in part by Google Inc. through a Google Research Award, by the PASCAL2 Network of Excellence under EC grant 216886 and by "Dote ricerca: FSE, Regione Lombardia. This publication only reflects the authors views.

whose mutual relationships can be classified as friendship or hostility [7]. The advent of online social networks has revamped the interest in these theories, and spurred a significant amount of recent work —see, e.g., [5, 8, 10, 3, 2], and references therein.

Many heuristics for link classification in social networks are based on a form of social balance summarized by the motto "the enemy of my enemy is my friend". This is equivalent to saying that the signs on the edges of a social graph tend to be consistent with some two-clustering of the nodes. By consistency we mean the following: The nodes of the graph can be partitioned into two sets (the two clusters) in such a way that edges connecting nodes from the same set are positive, and edges connecting nodes from different sets are negative. Although two-clustering heuristics do not require strict consistency to work, this is admittely a rather strong inductive bias. Despite that, social network theorists and practitioners found this to be a reasonable bias in many social contexts, and recent experiments with online social networks reported a good predictive power for algorithms based on the two-clustering assumption [8–10, 3]. Finally, this assumption is also fairly convenient from the viewpoint of algorithmic design.

In the case of undirected signed graphs $G = (V, E)$, the best performing heuristics exploiting the two-clustering bias are based on spectral decompositions of the signed adiacency matrix. Noticeably, these heuristics run in time $\Omega\big(|V|^2\big)$, and often require a similar amount of memory storage even on sparse networks, which makes them impractical on large graphs.

In order to obtain scalable algorithms with formal performance guarantees, we focus on the active learning protocol, where training labels are obtained by querying a desired subset of edges. Since the allocation of queries can match the graph topology, a wide range of graph-theoretic techniques can be applied to the analysis of active learning algorithms. In the recent work [2], a simple stochastic model for generating edge labels by perturbing some unknown two-clustering of the graph nodes was introduced. For this model, the authors proved that querying the edges of a low-stretch spanning tree [4] of the input graph $G = (V, E)$ is sufficient to predict the remaining edge labels making a number of mistakes within a factor of order $(\log |V|)^2 \log\log |V|$ from the theoretical optimum. The overall running time is $O(|E| \ln |V|)$. This result leaves two main problems open: First, low-stretch trees are a powerful structure, but the algorithm to construct them is not easy to implement. Second, the tree-based analysis of [2] does not generalize to query budgets larger than $|V| - 1$ (the edge set size of a spanning tree). In this paper we introduce a different active learning approach for link classification that can accomodate a large spectrum of query budgets. We show that on *any* graph with $\Omega(|V|^{3/2})$ edges, a query budget of $\mathcal{O}(|V|^{3/2})$ is sufficient to predict the remaining edge labels within a *constant* factor from the optimum. More in general, we show that a budget of at most order of $|V| + \big(\frac{|V|}{k}\big)^{3/2}$ queries is sufficient to make a number of mistakes within a factor of $\mathcal{O}(k)$ from the optimum with a running time of order $|E| + (|V|/k)\log(|V|/k)$. Hence, a query budget of $\Theta(|V|)$, of the same order as the algorithm based on low-strech trees, achieves an optimality factor $\mathcal{O}(|V|^{1/3})$ with a running time of just $\mathcal{O}(|E|)$.

2 Results

We consider undirected and connected graphs $G = (V, E)$ with unknown edge labeling $Y_{i,j} \in \{-1, +1\}$ for each $(i, j) \in E$. Edge labels can collectively be represented by the associated *signed* adjacency matrix Y, where $Y_{i,j} = 0$ whenever $(i, j) \notin E$. We define a simple stochastic model for assigning binary labels Y to the edges of G. We assume that edge labels are obtained by perturbing an underlying labeling which is initially consistent with an arbitrary (and unknown) two-clustering. More formally, given an undirected and connected graph $G = (V, E)$, the labels $Y_{i,j} \in \{-1, +1\}$, for $(i, j) \in E$, are assigned as follows. First, the nodes in V are arbitrarily partitioned into two sets, and labels $Y_{i,j}$ are initially assigned consistently with this partition (within-cluster edges are positive and between-cluster edges are negative). Then, given a nonnegative constant $p < \frac{1}{2}$, labels are randomly flipped in such a way that $\mathbb{P}\bigl(Y_{i,j} \text{ is flipped}\bigr) \le p$ for each $(i, j) \in E$. We call this a p-stochastic assignment. Note that this model allows for correlations between flipped labels.

A learning algorithm in the link classification setting receives a training set of signed edges and, out of this information, builds a prediction model for the labels of the remaining edges.

Fact 1. *For any training set $E_0 \subset E$ of edges, and any learning algorithm that is given the labels of the edges in E_0, the number M of mistakes made by the algorithm on the remaining $E \setminus E_0$ edges satisfies $\mathbb{E}\, M \ge p\,\bigl|E \setminus E_0\bigr|$, where the expectation is with respect to a p-stochastic assignment of the labels Y.*

An active learner for link classification is a special learning algorithm that first constructs a query set E_0 of edges, and then receives the labels of all edges in the query set. Based on this training information, the learner builds a prediction model for the labels of the remaining edges $E \setminus E_0$. We assume that the only labels ever revealed to the learner are those in the query set, no labels being revealed during the prediction phase. It is clear from Fact 1 that any active learning algorithm that queries the labels of at most a constant fraction of the total number of edges will make on average $\Omega(p|E|)$ mistakes.

Theorem 1. *An active learning algorithm parameterized by interger $k \ge 2$ exists such that for any graph $G = (V, E)$ with $|E| \ge 2|V| - 2 + 2\bigl(\frac{|V|-1}{k} + 1\bigr)^{\frac{3}{2}}$, and diameter D_G, the number M of mistakes made by the algorithm on G satisfies $\mathbb{E}M = \mathcal{O}(\min\{k, D_G\})\, p|E|$, while the query set size is bounded by $|V| - 1 + \bigl(\frac{|V|-1}{k} + 1\bigr)^{\frac{3}{2}} \le \frac{|E|}{2}$.*

Hence, even if D_G is large, setting $k = |V|^{1/3}$ yields a $\mathcal{O}(|V|^{1/3})$ optimality factor just by querying $\mathcal{O}(|V|)$ edges. On the other hand, a truly constant optimality factor is obtained by querying as few as $\mathcal{O}(|V|^{3/2})$ edges (provided the graph has sufficiently many edges). As a direct consequence (and surprisingly enough), on graphs which are only moderately dense we need not observe too many edges in order to achieve a constant optimality factor. It is instructive to compare the bounds obtained by our algorithm to the ones we can achieve by using the CCCC algorithm of [2], or the low-stretch spanning trees [4].

Because CCCC operates within a harder adversarial setting, it is easy to show that Theorem 9 in [2] extends to the p-stochastic assignment model by replacing $\Delta_2(Y)$ with $p|E|$ therein.[5] The resulting optimality factor is of order $\left(\frac{1-\alpha}{\alpha}\right)^{\frac{3}{2}}\sqrt{|V|}$, where $\alpha \in (0,1]$ is the fraction of queried edges out of the total number of edges. A quick comparison to Theorem 1 reveals that our algorithm achieves a sharper mistake bound for any value of α. For instance, in order to obtain an optimality factor which is lower than $\sqrt{|V|}$, CCCC has to query in the worst case a fraction of edges that goes to one as $|V| \to \infty$.

A low-stretch spanning tree achieves a polylogarithmic optimality factor by querying $|V|-1$ edge labels. The results in [4] show that we cannot hope to get a better optimality factor using a single low-stretch spanning tree combined with the analysis in [2]. For a comparable amount $\Theta(|V|)$ of queried labels, Theorem 1 offers the larger optimality factor $|V|^{1/3}$. However, we can get a *constant* optimality factor by increasing the query set size to $\mathcal{O}(|V|^{3/2})$. It is not clear how multiple low-stretch trees could be combined to get a similar scaling.

Finally, besides being easy to implement, our algorithm is also very fast.

Theorem 2. *For any input graph $G = (V, E)$ which is dense enough to ensure that the query set size is no larger than the test set size, the total time taken by our algorithm for predicting all test labels is $\mathcal{O}\left(|E| + \frac{|V|}{k}\log\frac{|V|}{k}\right)$. In particular, whenever $k|E| = \Omega(|V|\log|V|)$ we have that our algorithm works in constant amortized time. The space required is always linear in the input graph size $|E|$, independent of k.*

References

1. Cartwright, D. and Harary, F. Structure balance: A generalization of Heider's theory. *Psychological review*, 63(5):277–293, 1956.
2. Cesa-Bianchi, N., Gentile, C., Vitale, F., Zappella, G. A correlation clustering approach to link classification in signed networks. In COLT 2012.
3. Chiang, K., Natarajan, N., Tewari, A., and Dhillon, I. Exploiting longer cycles for link prediction in signed networks. In *20th CIKM*, 2011.
4. Elkin, M., Emek, Y., Spielman, D.A., and Teng, S.-H. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2010.
5. Guha, R., Kumar, R., Raghavan, P., and Tomkins, A. Propagation of trust and distrust. In *13th WWW*, pp. 403–412. ACM, 2004.
6. Harary, F. On the notion of balance of a signed graph. *Michigan Mathematical Journal*, 2(2):143–146, 1953.
7. Heider, F. Attitude and cognitive organization. *J. Psychol*, 21:107–122, 1946.
8. Kunegis, J., Lommatzsch, A., and Bauckhage, C. The Slashdot Zoo: Mining a social network with negative edges. In *18th WWW*, 2009.
9. Leskovec, J., Huttenlocher, D., and Kleinberg, J. Signed networks in social media. In *28th ICHFCS*, 2010.
10. Leskovec, J., Huttenlocher, D., and Kleinberg, J. Predicting positive and negative links in online social networks. In *19th WWW*, 2010.

[5] This theoretical comparison is admittedly unfair, as CCCC has been designed to work in a harder setting than p-stochastic. Unfortunately, we are not aware of any other general active learning scheme for link classification to compare with.

Rate-Based Stochastic Fusion Calculus and Continuous Time Markov Chains[⋆]

Gabriel Ciobanu[1] and Angelo Troina[2]

[1] Romanian Academy, Institute of Computer Science and "A.I.Cuza" University
[2] Dipartimento di Informatica, Università di Torino

1 Introduction

This paper presents a stochastic fusion calculus suitable to describe systems involving general patterns of interactions. We start from fusion calculus [8] which is a symmetric generalisation of the π-calculus, and present a rate-based stochastic fusion calculus, providing a concise and compositional way to describe the behaviour of complex systems by using probability distributions.

We provide the semantics of stochastic fusion calculus by using rate-based transition systems [4] in the elegant and general variant proposed by De Nicola et al. [2]. The stochastic nature of the new transition systems is given by the fact that transition labels represent actions, and the transition result is a function associating a positive real value to each possible target process, expressing the stochastic rate of an exponential distribution modelling the duration of the transition. For two processes running in parallel, we define the distribution of their synchronisation using their apparent rates. Associativity of parallel composition is a particularly desirable property either in the context of network and distributed systems, either in the context of biological systems, where parallel composition is often used to model molecular populations. Following the approach proposed in [2], associativity of the parallel composition operator is guaranteed in the rate-based stochastic semantics of the fusion calculus (differently to what happens in the stochastic π-calculus [9] and in a previous formalisation of a stochastic fusion calculus [1]).

We extend the notion of hyperbisimulation to stochastic fusion calculus, and prove that the stochastic hyperequivalence is a congruence. The rate-based transition system resulting from a stochastic fusion process leads the expression of a continuous time Markov chain which preserves the notion of hyperequivalence.

The modelling power of the stochastic fusion calculus is suggested by an example where we formalise some of the one-to-many interactions occurring between a plant root and a particular kind of fungi in the arbuscular mycorrhizal symbiosis. A quantitative simulation is performed using the PRISM model checker on the continuous time Markov chain extracted from the rate-based transition system describing such interactions by means of stochastic fusion processes.

[⋆] This research has been partially funded by the BioBITs Project (*Converging Technologies* 2007, area: Biotechnology-ICT), Regione Piemonte.

2 Rate-Based Stochastic Fusion Calculus

The fusion calculus was introduced by Parrow and Victor as a symmetric generalisation of the π-calculus [8]. The π-calculus has two binding operators (prefix and restriction), the effects of communication are local, and input and output actions are asymmetric. Unlike the π-calculus, the fusion calculus has only one binding operator, and the effects of communication are both local and global. Fusion calculus makes input and output operations fully symmetric: a more appropriate terminology for them might be action and co-action. A fusion is a name equivalence which allows to use interchangeably all the names of an equivalence class in a term of the calculus. Computationally, a fusion is generated as the result of a synchronisation between two complementary actions, and it is propagated to processes running in parallel within the same scope of the fusion. In practice, the effect of a fusion could be seen as the update of a (global) shared state. Fusions are suitable to express general patterns of interactions, including one-to-many and many-to-many interactions. We remind to [8] for details about the syntax and semantics of the fusion calculus.

Many phenomena which take place in practice are described by non-exponential distributions, and stochastic fusion calculus could be defined by using general distributions. For the sake of simplicity, we use here the exponential distribution, inheriting some properties derived from the memoryless feature of this distribution: the time at which a state change occurs is independent of the time at which the last state change occurred. In this way we do not have to keep track of the past state transitions (e.g. in an implementation).

Following the variant of rate transition systems [4] introduced in [2], we define the semantics of SFC via a transition relation $P \xrightarrow{\delta} \rho$ associating to a given process P and a transition action label δ a *next state function* (NSF) $\rho : \mathcal{SFC} \to \mathbb{R}^{\geq 0}$.

3 Stochastic Hyperbisimulation

Several papers of the last two decades define Markovian bisimulations, we mention the seminal paper by Larsen and Skou [7]. The definition of stochastic hyperbisimulation is also related to the notion of lumpability for Markov chains [5] (also see the next section). Two processes P and Q are lumping equivalent, and we denote this by $P \sim Q$, if the total rate of moving to an equivalence class S under $\sim$ is identical for both processes. Lumping equivalence also preserves stochastic rewards while reducing the size of the underlying stochastic transition system.

Two processes P and Q are stochastic bisimilar, written $P \dot{\sim}_{SH} Q$, if they are related by a stochastic bisimulation. Stochastic bisimilarity is not a congruence in the fusion calculus. We therefore look for the largest congruence included in the stochastic bisimilarity. This is achieved by closing the definition of stochastic bisimulation under arbitrary substitutions.

Definition 1 (Stochastic Hyperbisimulation). *A stochastic hyperbisimulation is an equivalence relation $\mathcal{R}$ over $\mathcal{SFC}$ satisfying the following properties:*

i) *$\mathcal{R}$ is closed under any substitution σ, i.e., $P\mathcal{R}Q$ implies $P\sigma\mathcal{R}Q\sigma$ for any σ;*
ii) *for each pair $(P, Q) \in \mathcal{R}$, for all actions δ, and for all equivalence classes $S \in \mathcal{SFC}/\mathcal{R}$, we have $\gamma_\delta(P, S) = \gamma_\delta(Q, S)$.*

Two processes P and Q are stochastic hyperbisimulation equivalent (or stochastic hyperequivalent) if they are related by a stochastic hyperbisimulation. We write $P \sim_{SH} Q$.

The following holds.

Theorem 1. *(**Congruence**) Stochastic hyperequivalence is a congruence, i.e., for $P, Q \in \mathcal{SFC}$ and $C \in \mathcal{SFC}[\,]$, $P \sim_{SH} Q$ implies $C[P] \sim_{SH} C[Q]$.*

4 Stochastic Fusion Processes as CTMCs

We provide a mechanism to translate the rate-based transition system deriving from the stochastic semantics into a Continuous Time Markov Chain (CTMC), providing a wide set of means for automatic verification.

By construction, the following holds.

Theorem 2. *Stochastic hyperequivalence preserves strong Markovian bisimulation, i.e., for $P, Q \in \mathcal{SFC}$, $P \sim_{SH} Q$ implies that the related CTMCs are Markovian bisimilar.*

5 Modelling the Arbuscular Mycorrhizal Symbiosis

The arbuscular mycorrhizal (AM) symbiosis is an example of association with high compatibility formed between fungi belonging to the Glomeromycota phylum and the roots of most land plants [3]. AM fungi are obligate symbionts, in the absence of a host plant, spores of AM fungi germinate and produce a limited amount of mycelium. The recognition between the two symbionts is driven by the perception of diffusible signals and, once reached the root surface, the AM fungus enters in the root, overcomes the epidermal layer and it grows inter-and intracellularly all along the root in order to spread fungal structures. Once inside the inner layers of the cortical cells the differentiation of specialised, highly branched intracellular hyphae called arbuscules occur. Arbuscules are considered the major site for nutrients exchange between the two organisms. The fungus supply the host with essential nutrients such as phosphate, nitrate and other minerals from the soil. In return, AM fungi receive carbohydrates derived from photosynthesis in the host. AM symbiosis also confers resistance to the plant against pathogens and environmental stresses. The colonisation of the host plant requires the accomplishment of two main events: i) signalling and partner recognition, ii) the colonisation of root tissues and the development of intraradical fungal structures leading to a functional symbiosis.

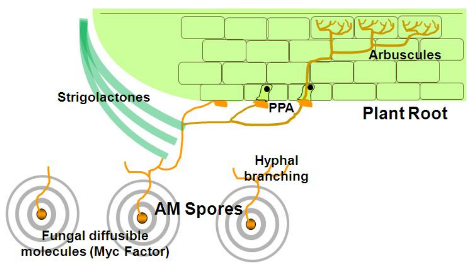


Fig. 1. The Arbuscular Mycorrhizal symbiosis

The interaction begins with a molecular dialogue between the plant and the fungus [3]. Host roots release signalling molecules characterised as *strigolactones*. Within just a few hours, strigolactones at subnanomolar concentrations induce alterations in fungal physiology, mitochondrial activity and extensive hyphal branching (leading the fungal spore to produce hyphae towards the plant root).

The external signal released by AM fungi (called *myc factor*) is perceived by a receptor on the plant plasma membrane and is transduced into the cell with the activation of a symbiotic signalling pathway that lead to the colonisation process (Pre Penetration Apparatus, PPA).

We model these initial communication between the plant root and AM fungi with SFC and simulated the resulting CTMC with the PRISM model checker.

An extended report about the work on this paper is available at: `http://www.di.unito.it/~troina/ictcs12/stocFusion.pdf`.

References

1. G. Ciobanu. From Gene Regulation to Stochastic Fusion. LNCS vol.5204, 51-63, 2008.
2. R. De Nicola, D. Latella, M. Loreti, M. Massink. Rate-based Transition Systems for Stochastic Process Calculi. Proc. ICALP, LNCS vol.5556, 435-446, 2009.
3. J. Harrison. Signaling in the Arbuscular Mycorrhizal Symbiosis. Annu. Rev. Microbiol., vol.59, 19–42, 2005.
4. B. Klin, V. Sassone. Structural Operational Semantics for Stochastic Process Calculi. Proc. FoSSaCS, LNCS vol.4962, 428-442, 2008.
5. J.G. Kemeny, J.L. Snell. *Finite Markov Chains*. Springer, 1976.
6. J. Krivine, R. Milner, A. Troina. Stochastic Bigraphs. Proc. MFPS'08, ENTCS, vol.218, 73-96, 2008.
7. K.G. Larsen, A. Skou. Bisimulation through Probabilistic Testing. Information and Computation, vol.94, 1-28, 1991.
8. J. Parrow, B. Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. Proc. LICS, IEEE Computer Society, 176-185, 1998.
9. C. Priami. Stochastic π-Calculus. Computer Journal, vol.38, 578-589, 1995.

Algebraic Characterization of the Class of Languages recognized by Measure Only Quantum Automata

Carlo Comin[1,2] and Maria Paola Bianchi[1]

[1]Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy
[2]ESTECO, AREA di Ricerca Science Park, Padriciano 99, Trieste, Italy
carlo.comin.86@gmail.com, bianchi@di.unimi.it

Abstract. We study a model of one-way quantum automaton where only measurement operations are allowed (MON-1QFA). We give an algebraic characterization of $\mathbf{LMO}(\Sigma)$, showing that the syntactic monoids of the languages in $\mathbf{LMO}(\Sigma)$ are exactly the literal pseudovariety of J-trivial literally idempotent monoids, where J is the Green's relation determined by two-sided ideals. We also prove that $\mathbf{LMO}(\Sigma)$ coincides with the literal variety of literally idempotent piecewise testable regular languages. This allows us to prove the existence of a polynomial time algorithm for deciding whether a regular language belongs to $\mathbf{LMO}(\Sigma)$.

Introduction and preliminaries. This paper gives a characterization of the class of languages recognized by a model of quantum automata, by using tools from algebraic theory, in particular, varieties of languages. Many models of one-way quantum finite automata are present in the literature: the oldest is the Measure-Once model [2, 5], characterized by unitary evolution operators and a single measurement performed at the end of the computation. On the contrary, in other models, evolutions and measurements alternate along the computation [1, 9]. The model we study is the Measure-Only Quantum Automaton (MON-1QFA), introduced in [4], in which we allow only measurement operations, not evolution. All these quantum models are generalized by Quantum Automata with Control Language [3].

A MON-1QFA over the alphabet Σ is a tuple of the form $A = \langle \Sigma \cup \{\#\}, (O_c)_{c \in \Sigma \cup \{\#\}}, \pi_0, F\rangle$. The complex m-dimensional vector $\pi_0 \in \mathbb{C}^{1\times m}$, with unitary norm $||\pi_0|| = 1$, is called the quantum initial state of A. For every $c \in \Sigma$, $O_c \in \mathbb{C}^{m\times m}$ is (the representative matrix of) an idempotent Hermitian operator and denotes an observable. The subset $F \subseteq V(O_\#)$ of the eigenvalues of $O_\#$ is called the spectrum of the quantum final accepting states of A.
The computation dynamics of automaton A is carried out in the following way: let $x = x_1 \dots x_n \in \Sigma^*$, suppose we start from π_0, then A measures the system with cascade observables $O_{x_1}, \dots, O_{x_n}$ (by applying the associated orthogonal projectors) and then performs the final measure with the end-word observable $O_\#$, that is the observable of the final accepting states F of A.
This last measure returns, as a result, an eigenvalue $r \in V(O_\#)$, if $r \in F$ then we

say that the automaton A accepts the word $x \in \Sigma^*$, otherwise that A does not accept it. What is remarkable in this computation dynamics, is the probabilistic behavior of the automaton A, that is the probability $p_A(x)$ that A accepts $x = x_1 \cdots x_n$. In the specific case of MON-1QFAs it turns out to be of some interest to express $p_A(x)$ using the well-known formalism of quantum density matrices. We say that a language L is recognized by A with isolated cut point λ iff for all $x \in \Sigma^*$ $p_A(x) > \lambda \Leftrightarrow x \in L$ and there exists a constant value $\delta > 0$ such that $|p_A(x) - \lambda| \geq \delta$.

We now recall some general definitions and results from the algebraic theory of automata and formal languages. For more details, we refer the reader to, e.g. [6, 10]. Let L be a regular language and let $\langle \Sigma, Q, \delta, q_0, F \rangle$ be the minimal deterministic automaton recognizing L. For a word $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$, we define its variation as $\mathrm{Var}_L(w) = \#\{0 \leq k < n \mid \delta(\sigma_1 \cdots \sigma_k) \neq \delta(\sigma_1 \cdots \sigma_{k+1})\}$. We say that L has finite variation iff $\sup_{x \in \Sigma^*} \mathrm{Var}_L(x) < \infty$. Using results and similar techniques as in [4], it is not difficult to show that the class $\mathbf{LMO}(\Sigma)$ of languages recognized by a MON-1QFA with isolated cut point is a boolean algebra of regular languages in Σ^* with finite variation. We say that a language $L \in \Sigma^*$ is literally idempotent iff for all $x, y \in \Sigma^*$ and $a \in \Sigma$, $xa^2y \in L \Leftrightarrow xay \in L$; we say that L is literally idempotent piecewise testable if and only if it lies in the boolean closure of the following class of languages: $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_k \Sigma^*$, for $a_1, a_2, \ldots, a_k \in \Sigma$ and $a_1 \neq \cdots \neq a_k$. We denote by liId the class of literally idempotent languages and by liId$\mathbf{PT}$ the class of literally idempotent piecewise testable languages. For any language L, we call $M(L)$ its syntactic monoid. We say that a class of finite monoids $\mathbf{A}$ is a (literal) pseudovariety if and only if it is closed under (literal) substructures, homomorphic images and finite direct products, [8]. Let $\mathbf{A}$ be a class of monoids and let Σ be an alphabet. We denote by $V_\Sigma(\mathbf{A})$ the class of regular languages on Σ having syntactic monoid in $\mathbf{A}$. Let L, R and J be the Green's relations determined by left, right and two-sided ideals, respectively. In this paper we denote by $\mathbf{R}$ the pseudovariety of R-trivial finite monoids and by $\mathbf{J}$ the pseudovariety of J-trivial finite monoids. We also define $\overline{\mathbf{J}}$ as the literal pseudovariety of J-trivial syntactic monoids $M(L)$ such that the associated morphism $\phi_L : \Sigma^* \to M(L)$ satisfies the literal idempotent condition $\phi_L(\sigma)\phi_L(\sigma) = \phi_L(\sigma)$, for every $\sigma \in \Sigma$. We say that a class of regular languages $V : \Sigma \to 2^{\Sigma^*}$ is a $*$-variety of *Eilenberg* if $V(\Sigma)$ is closed under boolean operations, right and left quotient, and inverse homomorphism. Replacing closure under inverse homomorphism by closure under inverse literal homomorphism, we get the notion of literal variety of languages. A fundamental result is due to Eilenberg, who showed that there exists a bijection V_Σ from the psuedovarieties of monoids and the $*$-varieties of Eilenberg of formal languages [10]. In [8], Klíma and Polák showed the following

Theorem 1. *Let $L \subseteq \Sigma^*$ be a formal language. The following propositions are equivalent: (i) $L \in$ liId$\boldsymbol{PT}$, (ii) $L \in V_\Sigma(\mathbf{J}) \cap$ liId(Σ), and (iii) $L \in V_\Sigma\left(\overline{\mathbf{J}}\right)$.*

Results. We give a direct proof that the class of finite variation regular languages is a $*$-variety of Eilenberg. Moreover, we observe that a regular language

L has finite variation if and only if its syntactic monoid is R-trivial. We proceed further on with our analysis by showing that the class of MON-1QFA over Σ is in fact a sub-class of Latvian Automata. This class of automata has been fully characterized algebraically in [1] as the class of automata recognizing exactly regular languages having syntactic monoids in the class **BG** of block groups.

Theorem 2. *Let A be a* MON-1QFA *on Σ and let L_A be a language recognized by A with cut-point λ isolated by δ. Then there exists a Latvian automaton A' recognizing $L_{A'} = L_A$ with cut-point $\lambda' = \frac{1}{2}$ isolated by $\delta' = \frac{\delta}{2\cdot max(\lambda,1-\lambda)}$.*

This directly implies that $\mathbf{LMO}(\Sigma) \subseteq V_\Sigma(\mathbf{BG})$.

Combining our analysis with the results of [1] on block groups syntactic monoids and the results of [4] on finite variation languages, we prove the following

Theorem 3. *Let $L \in \mathbf{LMO}(\Sigma)$ be a language recognized by some* MON-1QFA *with isolated cutpoint. Then its syntactic monoid $M(L)$ is an R-trivial block group, formally speaking $M(L) \in \mathbf{BG} \cap \mathbf{R}$.*

Since an R-trivial block group is also J-trivial, and since $\mathbf{LMO}(\Sigma)$ is a boolean algebra, we have $\mathbf{LMO}(\Sigma) \subseteq V_\Sigma(\mathbf{J})$. This, together with Theorem 1 and the fact that languages in $\mathbf{LMO}(\Sigma)$ are literally idempotent, leads to the following

Theorem 4. *$\mathbf{LMO}(\Sigma) \subseteq V_\Sigma(\overline{\mathbf{J}})$.*

We now show how languages in liId**PT** can be recognized by MON-1QFAs. Consider the language $L[a_1,\ldots,a_k] = \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_k \Sigma^*$, where $a_1,\ldots,a_k \in \Sigma$, $a_i \neq a_{i+1}$ for $1 \leq i < k$, and let $S = \{a_1,\ldots,a_k\}$. For every $\alpha \in S$, let $\#\alpha$ be the number of times that α appears as a letter in the word $a_1 a_2 \cdots a_k$. Let $j_1^{(\alpha)} < j_2^{(\alpha)} < \cdots < j_{\#\alpha}^{(\alpha)}$ be all the indexes such that $\alpha = a_{j_1^{(\alpha)}} = \ldots = a_{j_{\#\alpha}^{(\alpha)}}$. We define, for every $\alpha \in S$, two orthogonal projectors of dimension $(k+1)\times(k+1)$: the up operator $P_{\nearrow}^{(k)}(\alpha)$ and the down operator $P_{\searrow}^{(k)}(\alpha)$, such that

$$\left(P_{\nearrow}^{(k)}(\alpha)\right)_{rs} = \begin{cases} 1 & \text{if } r = s \text{ and } \forall\, 1 \leq i \leq \#\alpha \text{ it holds } r,s \notin \{j_i^\alpha, j_i^\alpha + 1\}, \\ \frac{1}{2} & \text{if } \exists\, 1 \leq i \leq \#\alpha \text{ such that } r,s \in \{j_i^\alpha, j_i^\alpha + 1\}, \\ 0 & \text{otherwise,} \end{cases}$$

$$\left(P_{\searrow}^{(k)}(\alpha)\right)_{rs} = \begin{cases} \frac{1}{2} & \text{if } r = s \text{ and } \exists\, 1 \leq i \leq \#\alpha \text{ such that } r,s \in \{j_i^\alpha, j_i^\alpha + 1\}, \\ -\frac{1}{2} & \text{if } r \neq s \text{ and } \exists\, 1 \leq i \leq \#\alpha \text{ such that } r,s \in \{j_i^\alpha, j_i^\alpha + 1\}, \\ 0 & \text{otherwise.} \end{cases}$$

By calling e_j the boolean row vector such that $(e_j)_i = 1 \Leftrightarrow i = j$, we define $A[a_1,\ldots,a_k] = \langle \Sigma \cup \{\#\}, \pi_0^{(k)}, \{O_\sigma^{(k)}\}_{\sigma\in\Sigma\cup\{\#\}}, F^{(k)} \rangle$ as the MON-1QFA where

- $\pi_0^{(k)} = e_1 \in \mathbb{C}^{1\times(k+1)}$,
- for $\alpha \in S$, the associated projectors of $O_\alpha^{(k)}$ are $P_{\nearrow}^{(k)}(\alpha)$ and $P_{\searrow}^{(k)}(\alpha)$,
- with each $O_\sigma^{(k)}$ such that $\sigma \in \Sigma \setminus S$, we associate the projector $I_{(k+1)\times(k+1)}$,
- the projector of the accepting result of $O_\#^{(k)}$ is $(e_{k+1})^T e_{k+1}$, i.e. the $(k+1)\times(k+1)$ boolean matrix having a 1 only in the bottom right entry.

A careful analysis of the behavior of $A[a_1, \ldots, a_k]$ leads to the following

Theorem 5. *The automaton $A[a_1, \ldots, a_k]$ recognizes $L[a_1, \ldots, a_k]$ with cutpoint $\lambda = \frac{1}{2^{2k+1}}$ isolated by $\delta = \frac{1}{2^{2(k+1)}}$.*

Since the class liId**PT** is the boolean closure of languages of the form $L[a_1, \ldots, a_k]$, and **LMO**(Σ) is a boolean algebra, Theorem 5 implies that all literally idempotent piecewise testable languages can be recognized by MON-1QFAs. The observations made up to this point imply our main result:

Theorem 6. ***LMO***$(\Sigma) = V_{\Sigma}(\overline{\mathbf{J}}) =$ *liId**PT***(Σ).

Theorem 6 allows us to prove the existence of a polynomial time algorithm for deciding **LMO**(Σ) membership:

Theorem 7. *Given a regular language $L \in \Sigma^*$, the problem of determining whether $L \in$ **LMO**(Σ) is decidable in time $O((|Q|+|\Sigma|)^2)$, where $|Q|$ is the size of the minimal deterministic automaton for L.*

This algorithm first constructs the minimal deterministic automaton A_L for L in time $O(|Q|\log(|Q|))$ as shown in [7]. Then, in time $O(|Q|)$, it checks whether L is literally idempotent by visiting all the vertices in the graph of A_L. Finally, it verifies whether L is piecewise testable in time $O((|Q|+|\Sigma|)^2)$ with the technique shown in [11]. The fact that **LMO**$(\Sigma) =$ liId**PT**(Σ) completes the proof.

Acknowledgements: The authors wish to thank Alberto Bertoni for the stimulating discussions which lead to the results of this paper.

References

1. A. Ambainis, M. Beaudry, M. Golovkins, A. Kikusts, M. Mercer, D. Thérien, *Algebraic Results on Quantum Automata*, Theory Comp. Syst., vol. 39(1), (2006), 165-188.
2. A. Bertoni, M. Carpentieri, *Regular Languages Accepted by Quantum Automata*, Inf. Comput., vol. 165(2), (2001), 174-182.
3. A. Bertoni, C. Mereghetti, B. Palano, *Quantum Computing: 1-Way Quantum Automata* Developments in Language Theory 2003: 1-20
4. A. Bertoni, C. Mereghetti, B. Palano, *Trace monoids with idempotent generators and measure-only quantum automata*, Natural Comp., vol. 9(2), (2010), 383-395.
5. A. Brodsky, N. Pippenger, *Characterizations of 1-Way Quantum Finite Automata*, SIAM J. Comput., vol. 31(5), (2002), 1456-1478.
6. S. Eilenberg, *Automata, languages, and machines*, vol. A,B, Academic Press, 1976.
7. J.E. Hopcroft, *An N Log N Algorithm for Minimizing States in a Finite Automaton*, Technical Report. Stanford University, Stanford, CA, USA, 1971.
8. O. Klíma, L. Polák, *On Varieties of Literally Idempotent Languages*, ITA 42(3), (2008), 583-598.
9. A. Kondacs, J. Watrous, *On the Power of Quantum Finite State Automata*, FOCS (1997) 66-75.
10. J. E. Pin, *Varieties of formal languages*, North Oxford, London and Plenum, New-York, 1986.
11. A.N. Trahtman, *Piecewise and Local Threshold Testability of DFA*, FCT (2001), 347-358.

Efficient algorithms for distributed shortest paths on power-law networks

Gianlorenzo D'Angelo[1], Mattia D'Emidio[2], Daniele Frigioni[2], and Daniele Romano[2]

[1] MASCOTTE Project INRIA/I3S(CNRS/UNSA) gianlorenzo.d_angelo@inria.fr
[2] Dip. di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila
{mattia.demidio,daniele.frigioni}@univaq.it, daniele.romano.vis@gmail.com

1 Introduction

The problem of computing and maintaining shortest paths in a distributed network whose topology dynamically changes is a core functionality of today's communication networks. The problem has been widely studied in the literature, and the solutions found can be classified as *distance-vector* and *link-state*.

Distance-vector algorithms, as for example the distributed Bellman-Ford method [6], require that a node knows the distance from each of its neighbors to every destination and stores them in a data structure called *routing table*; a node uses its own routing table to compute the distance and the via to each destination. The main drawbacks of distance-vector algorithms are the massive use of communication resources and the well-known *looping* phenomena.

Link-state algorithms, as e.g. the OSPF protocol used in Internet [7], require that a node knows the entire network topology to compute its distance to any destination, usually running Dijkstra's algorithm. They are free of looping, although each node needs to receive and store up-to-date information about the entire network topology after a change, thus requiring quadratic space per node.

In the last years, there has been a renewed interest in devising new efficient distance-vector solutions for Large-Scale Ethernet networks, where usually scalability and reliability are highly desirable properties or where the memory power of the nodes is limited (see, e.g., [2, 8, 9]). Notwithstanding this increasing interest, the most interesting distance-vector algorithm is still DUAL (Diffuse Update ALgorithm) [4], which is free of looping and is part of the CISCO's widely used EIGRP protocol, although it requires a high space per node.

In this paper, we present two contributions in this field. First, we describe LFR (*Loop Free Routing*) a new loop-free distance vector routing algorithm, recently proposed in [3], which: (i) from the theoretical point of view, is able to update the shortest paths of a distributed network in fully dynamic scenarios using the same message complexity and less space per node than DUAL; (ii) from the experimental point of view, has been shown to be the best choice, both in terms of messages sent and space requirements, in networks having a power-law node degree distribution, a highly important class of networks which includes many currently implemented communication infrastructures, like the Internet, the WWW, and so on [1]. Second, we introduce DP (*Distributed Pruning*), a new general and practical technique which is a generalization of DLP [2]. DP can be combined with every distance-vector algorithm based on shortest paths with the aim of overcoming some of their limitations (high number of messages sent, low scalability, poor convergence) in power-law networks. We give experimental evidence of the effectiveness of DP, showing that the combination of DP with DUAL and LFR provides a huge improvement in both the global number of messages sent and the space occupancy per node wrt DUAL and LFR, resp..

2 Description of LFR

In this section, we describe LFR, introduced in [3]. LFR stores, for each node v, the estimated distance $\texttt{D}[v, s]$ and the *feasible via*, $\texttt{VIA}[v, s]$, that is the node which provides the minimum distance to s and satisfies SNC (*Source Node Condition*), a sufficient condition for loop-freedom [4]. In addition, node v maintains for each $s \in V$, the following data structures: (i) $\texttt{ACTIVE}[v, s]$: the state of node v wrt a source s; v is in *active* state ($\texttt{ACTIVE}[v, s] = \mathit{true}$), if and only if v is trying to update $\texttt{VIA}[v, s]$ after a weight increase operation; (ii) $\texttt{UD}[v, s]$: the distance from v to s through $\texttt{VIA}[v, s]$; if v is active then $\texttt{UD}[v, s] \geq \texttt{D}[v, s]$, otherwise they coincide. In addition, in order to implement SNC, node v stores a temporary data structure $\texttt{tempD}_v$ which is allocated only when v is active wrt s, and it is deallocated when v turns passive wrt s. The entry $\texttt{tempD}_v[u, s]$ contains $\texttt{UD}[u, s]$, for each $u \in N(v)$.

The algorithm starts when the weight of an edge $\{x_i, y_i\}$ changes. As a consequence, x_i (y_i, resp.) sends to y_i (x_i, resp.) an *update* message carrying the value $\texttt{UD}[x_i, s]$ ($\texttt{UD}[y_i, s]$, resp.). If an arbitrary node v receives an *update* message from $u \in N(v)$, then it performs an update procedure in which, basically, v compares the received value $\texttt{UD}[u, s] + w(u, v)$ with $\texttt{D}[v, s]$ in order to determine whether v needs to update its estimated distance and $\texttt{VIA}[v, s]$. If node v is active, the processing of the message is postponed by enqueueing it into the FIFO queue associated to s. Otherwise, if $\texttt{D}[v, s] > \texttt{UD}[u, s] + w(u, v)$, then v sets $\texttt{D}[v, s] = \texttt{UD}[u, s] + w(u, v)$ and $\texttt{VIA}[v, s] = u$, while if $\texttt{D}[v, s] < \texttt{UD}[u, s] + w(u, v)$ and $\texttt{VIA}[v, s] = u$, node v performs a phase called Local-Computation in which it sends a *get.dist* to all its neighbor, except $\texttt{VIA}[v, s] = u$, in order to know the corresponding estimated distances to s. Each neighbor $u \in N(v)$ replies with its $\texttt{UD}[u, s]$. When v receives these values, it tries to compute a new $\texttt{VIA}[v, s]$, by comparing the received distances with $\texttt{D}[v, s]$. If this phase succeeds, node v updates its routing information and propagates the change, Otherwise, node v initiates a distributed phase, named Global-Computation. It sets $\texttt{UD}[v, s] = \texttt{UD}[\texttt{VIA}[v, s], s] + w(v, \texttt{VIA}[v, s])$ and sends to all its neighbors, except $\texttt{VIA}[v, s]$, a *get.feasible.dist* message, carrying $\texttt{UD}[v, s]$. A node $k \in N(v)$ that receives such a message first verifies whether $\texttt{VIA}[k, s] = v$ or not. In the first case, it replies to v with $\texttt{UD}[k, s]$. In the second case, it performs the Local-Computation and possibly the Global-Computation, in order to update its routing information and to reply to v. Note that this distributed phase can involve the whole network. Finally, if either $\mathtt{D}[v,s] = \mathtt{UD}[u,s] + w(u,v)$ or $\mathtt{D}[v,s] < \mathtt{UD}[u,s] + w(u,v)$ and $\mathtt{VIA}[v,s] \neq u$, the message is discarded and the procedure ends.

If we denote as Φ the total number of nodes *affected* by a set of updates on the edges of the network, as ϕ the maximum number of destinations for which a node is affected, and as Δ the maximum node degree of the network, then LFR requires $O(\Delta \cdot \Phi)$ messages and $O(n + \Delta \cdot \phi)$ space per node, while DUAL requires $O(\Delta \cdot \Phi)$ messages and $\Theta(\Delta \cdot n)$ space per node.

3 Distributed Pruning

In this section, we introduce DP (Distributed Pruning), a new technique that can be combined with every distance-vector algorithm, with the aim of reducing the messages sent and the space occupancy per node of that algorithm on power-law networks. The idea underlying DP mainly rely on two facts: (i) a power-law network with n nodes typically has average node degree much smaller than n and a number of nodes with low degree which is generally high (for example, the graphs of the *CAIDA IPv4 topology dataset* [5] have average degree approximately equal to $n/2000$, and a number of nodes with degree smaller than 3 approximately equal to $2n/3$); (ii) there are many topological situations in which nodes with degree smaller than 3 should neither perform nor be involved in the distributed computation of shortest paths, as they cannot provide any useful information. DP has been designed to improve the performances of a generic distance-vector algorithm, by exploiting these configurations.

In particular, when applied to a generic distance-vector algorithm, DP forces the distributed computation to be carried out only by those nodes of the network which has degree greater than two (*central* nodes). The non-*central* nodes receive updated routing information passively and do not start any kind of distributed computation. To implement this strategy, DP requires that a generic node stores and updates information about non-*central* paths. To this aim, each node v maintains a data structure, called *CHain Path*, which is an array containing one entry $\mathtt{CHP}_v[s]$, for each *central* node s, where the list of all edges, with the corresponding weight, belonging to the non-*central* paths containing s are stored. The space overhead induced by the *CHain Path* is clearly $O(n)$ per node. However, DP globally induces a reduction in the space occupancy per node, as the overhead required to store the *CHain Path* is counterbalanced by a reduction in the space occupancy per node of the original algorithm, which can avoid to store some information about non-*central* nodes.

In order to check the effectiveness of DP, we combined it with DUAL and LFR by obtaining two new algorithms named DUAL-DP and LFR-DP, resp.. Then, we implemented the four algorithms in OMNeT++[3] and performed a preliminary experimental study. As input to the algorithms, we considered instances similar to the ones used in [2]. We generated a set of different tests, each test consists of a CAIDA graph and a set of k edge updates, where $k \in \{5, 10, \ldots, 200\}$

[3] OMNeT++, Discrete event simulation environment: `http://www.omnetpp.org`.

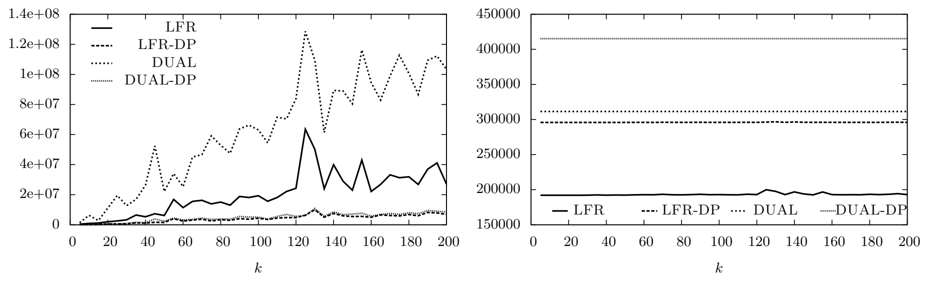


Fig. 1: Number of messages sent (left) and average space occupancy per node in Bytes (right) of LFR, LFR-DP, DUAL and DUAL-DP on a CAIDA graph with 8000 nodes subject to a set of k edge updates.

and each edge update consists of multiplying the weight of a randomly selected edge by a percentage value randomly chosen in $[50\%, 150\%]$.

Our experiments show that the combinations of DUAL and LFR with DP provide a huge improvement both in the global number of sent messages (Fig. 1(left)) and in the space occupancy per node wrt DUAL and LFR, resp.. The reduction in the space occupancy per node is significant both in the average and in the maximum case. We have noticed improvements also wrt the combinations of DUAL and LFR with DLP ([2]), thus allowing us to state that DP represents a step forward wrt the results presented in [2].

References

1. R. Albert and A.-L. Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
2. G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. A speed-up technique for distributed shortest paths computation. In *ICCSA 2011*, volume 6783 of *LNCS*, pages 578–593, 2011.
3. G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. Engineering a new loop-free shortest paths routing algorithm. In *SEA 2012*, volume 7276 of *LNCS*, pages 123–134, 2012.
4. J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. on Networking*, 1(1):130–141, 1993.
5. Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and K. Claffy. The CAIDA IPv4 routed/24 topology dataset. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.
6. J. McQuillan. Adaptive routing algorithms for distributed computer networks. Technical Report BBN Report 2831, Cambridge, MA, 1974.
7. J. T. Moy. *OSPF: Anatomy of an Internet routing protocol.* Addison-Wesley, 1998.
8. S. Ray, R. Guérin, K.-W. Kwong, and R. Sofia. Always acyclic distributed path computation. *IEEE/ACM Trans. on Networking*, 18(1):307–319, 2010.
9. C. Zhao, Y. Liu, and K. Liu. A more efficient diffusing update algorithm for loop-free routing. In *5th Int. Conf. on Wireless Communications, Networking and Mobile Computing (WiCom09)*, pages 1–4. IEEE Press, 2009.

A complete polynomial λ-calculus

Erika De Benedetti Simona Ronchi Della Rocca

Università degli Studi di Torino
Dipartimento di Informatica
Corso Svizzera 185, 10149 Torino
{debenede,ronchi}@di.unito.it

Abstract. We propose a system of stratified types, inspired by intersection types but without associativity, which is correct and complete for polynomial time computations, while typing all the strongly normalizing terms, so increasing the expressivity w.r.t. the previous proposals.

1 Introduction

This work is in the field of Implicit Computational Complexity. One of the aims of Implicit Computational Complexity is the design of programming languages with bounded computational complexity. In fact, guaranteeing and certifying a limited resources usage is of central importance for various aspects of computer science. One of the more promising approaches to this aim is based on the use of lambda-calculus as paradigmatic programming language, and on the design of type assignment systems for lambda-terms, where types guarantee, besides the functional correctness, also the desired complexity bound. In this spirit, some systems characterizing polynomial time complexity have been designed, inspired by the Light Logics. The problem of these characterizations is that, while the systems are functionally complete, their expressivity is very small, in the sense that few algorithms can be coded. In particular, a proper subset of strongly normalizing terms can be typed.

There are in the literature two characterizations of `PTIME` through type assignments for λ-calculus, `DLAL` ([2], [3]) based on Light Affine Logic [1], an affine version of Light Linear Logic [7], and `STA` [5, 6], based on the Soft Linear Logic of Lafont [8]. Both these characterizations are correct and complete with respect to `PTIME`, namely every typed term reduces to normal form in a number of steps which is polynomial in its size, and moreover all and only the polynomial functions can be coded by a typed term. However, the completeness in both systems is functional, not algorithmic, in the sense that for every polynomial function there is at least one algorithm that can be typed; even though the algorithmic completeness is undecidable, we would like to design more expressive systems.

2 The system `ISTA`

Here we propose a type assignment system for λ-calculus, whose types are *stratified* types, defined as follows.

Definition 1. *i) The set $\mathcal{T}$ of types is defined by the following syntax:*

$$\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A} \qquad \textit{(linear types)}$$

$$\sigma ::= \mathtt{A} \mid \{\underbrace{\sigma, ..., \sigma}_{n}\} \quad n > 0 \ \textit{(stratified types)}$$

ii) Let $\equiv$ denote the syntactical equality between (stratified) types. Types will be considered modulo the following equivalence relation:

$$\begin{array}{lll} \mathtt{A} \equiv \mathtt{B} \Rightarrow \mathtt{A} = \mathtt{B} & & \\ \sigma \multimap \mathtt{A} = \tau \multimap \mathtt{B} & \mathtt{iff}\ \sigma = \tau \quad \mathtt{and} \quad \mathtt{A} = \mathtt{B} & \\ \{\sigma_1, ..., \sigma_n\} = \{\tau_1, ..., \tau_m\} & \mathtt{iff}\ \forall \sigma_i.\exists \tau_j.\sigma_i = \tau_j \quad \mathtt{and} \quad \forall \tau_j.\exists \sigma_i.\sigma_i = \tau_j & \end{array}$$

i.e., a stratified type represents a set.

iii) The system ISTA *proves judgments of the kind $\Gamma \vdash \mathtt{M} : \sigma$, where Γ is a partial function associating to variables a linear type or a stratified type $\{\sigma_1, ..., \sigma_n\}$ for $n > 0$. $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$. The system is defined in Table 1.*

$$\frac{}{\mathtt{x} : \mathtt{A} \vdash \mathtt{x} : \mathtt{A}}\ (Ax)$$

$$\frac{\Gamma \vdash \mathtt{M} : \mathtt{B} \quad \mathtt{x} \notin \mathrm{dom}(\Gamma)}{\Gamma, \mathtt{x} : \mathtt{A} \vdash \mathtt{M} : \mathtt{B}}\ (w) \qquad \frac{\Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : \mathtt{B}}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap \mathtt{B}}\ (\multimap I)$$

$$\frac{\Gamma_1 \vdash \mathtt{M} : \sigma \multimap \mathtt{A} \quad \Gamma_2 \vdash \mathtt{N} : \sigma \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \vdash \mathtt{MN} : \mathtt{A}}\ (\multimap E)$$

$$\frac{\Gamma \vdash \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash \mathtt{M} : \forall \mathtt{a}.A}\ (\forall I) \qquad \frac{\Gamma \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Gamma \vdash \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Gamma, \mathtt{x}_1 : \mathtt{A}, ..., \mathtt{x}_n : \mathtt{A} \vdash \mathtt{M} : \tau}{\Gamma \mathtt{x} : \{\mathtt{A}\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau}\ (m_l) \qquad \frac{\Gamma, \mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_n : \sigma_n \vdash \mathtt{M} : \tau \quad \sigma_i \text{ not linear}}{\Gamma, \mathtt{x} : \{\sigma_1, ..., \sigma_n\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau}\ (m_s)$$

$$\frac{\Gamma_i \vdash \mathtt{M} : \sigma_i \quad n \geq 1 \quad \sigma_i \neq \sigma_j}{\cup_i \{\Gamma_i\} \vdash \mathtt{M} : \{\sigma_1, ..., \sigma_n\}}\ (sp)$$

Table 1. The ISTA Type Assignment system.

The system is inspired to STA (in Table 2), where the modality ! has been replaced by the stratification. So, while in STA the multiplexor can contract only premises with the same type, here the stratification allows to contract also different premises, in case of stratified types. In case of linear types the stratification behaves like the !, and this is essential for typing in an uniform way the data types, in particular binary numbers. Stratified types allow us to exploit some good properties of intersection types, for which the intersection is idempotent and commutative, but not associative.

$$\frac{}{\mathtt{x} : \mathtt{A} \vdash_{\mathtt{STA}} \mathtt{x} : \mathtt{A}}\ (Ax) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \quad \mathtt{x} \notin dom\Theta}{\Theta, \mathtt{x} : A \vdash_{\mathtt{STA}} \mathtt{M} : \mu}\ (w) \qquad \frac{\Theta, \mathtt{x} : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{A}}{\Theta \vdash_{\mathtt{STA}} \lambda \mathtt{x}.\mathtt{M} : \mu \multimap \mathtt{A}}\ (\multimap I)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \multimap A \quad \Xi \vdash_{\mathtt{STA}} \mathtt{N} : \mu \quad \Theta \# \Xi}{\Theta, \Xi \vdash_{\mathtt{STA}} \mathtt{MN} : \mathtt{A}}\ (\multimap E) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin FV(\Theta)}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{A}}\ (\forall I)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E) \qquad \frac{\Theta, \mathtt{x}_1 : \mu, \ldots, \mathtt{x}_n : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \nu}{\Theta, \mathtt{x} :! \mu \vdash_{\mathtt{STA}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \nu}\ (m) \qquad \frac{\Theta \vdash \mathtt{M} : \mu}{!\Theta \vdash_{\mathtt{STA}} \mathtt{M} :! \mu}\ (sp)$$

Table 2. The STA Type Assignment system.

3 Properties

The system ISTA enjoys the following properties.

Theorem 1. *i) Let $\pi : \Gamma \vdash \mathtt{M} : \sigma$. Then* M *reduces to normal form in a number of steps $\in O(|\mathtt{M}|^{3d})$, where $|\mathtt{M}|$ is the size of* M *and d is the depth of π, i.e., the number of nested applications of rule (sp).*
ii) The system ISTA *givets type to all and only the strongly normalizing terms.*

Observe that these two properties are not in contradiction! So ISTA is more powerful than STA, which can type a proper subset of strongly normalizing terms.

We represent binary numbers in Church style, so the number w is represented by $\underline{w} = \lambda \mathtt{s}_0 \mathtt{s}_1 \mathtt{x}.\mathtt{s}_{i_1}(...(\mathtt{s}_{i_{\lfloor \log w \rfloor + 1}} \mathtt{x})...)$ where $i_j \in \{0, 1\}$, for any $w \neq 0$, and $\underline{0} = \lambda \mathtt{s}_0 \mathtt{s}_1 \mathtt{x}.\mathtt{x}$.
In STA, binary numbers are typed uniformely by the type

$$\mathbf{W} = \forall a.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a}$$

and moreover, $\underline{w}$ can be given the type

$$\mathbf{W}_{n,m} = \forall a.!^n(\mathtt{a} \multimap \mathtt{a}) \multimap !^m(\mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap \mathtt{a} \text{ , for all } n, m \geq 1$$

Observe that the possibility of non uniform typings for binary numbers is essential to limit the expressivity of the language, in that it does not allow nesting iterations of functions. Similarly, in ISTA we define types for binary numbers to be

$$\mathbf{WI}_{n,m} = \forall a.\{^n \mathtt{a} \multimap \mathtt{a}\}^n \multimap \{^m \mathtt{a} \multimap \mathtt{a}\}^m \multimap \mathtt{a} \multimap \mathtt{a} \text{ , for all } n, m \geq 1$$

so all Church numerals have in particular the type

$$\mathbf{WI} = \forall a.\{\mathtt{a} \multimap \mathtt{a}\} \multimap \{\mathtt{a} \multimap \mathtt{a}\} \multimap \mathtt{a} \multimap \mathtt{a}$$

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p. Then the term M represents ϕ in an untyped setting if and only if $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$. In a typed setting, M needs to satisfy also typing constraints. Namely, in STA it needs to be typed as

$$\mathtt{x}_1 :!^{i_1} \mathbf{W}_{j_1,k_1}, ..., \mathtt{x}_p :!^{i_p} \mathbf{W} j_p k_p \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1 ... \mathtt{x}_p : \mathbf{W}_{j,k}$$

for some j, k, p, j_h, k_h $(1 \leq h \leq p)$.

In ISTA the corresponding typing must be

$$\mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_p : \sigma_p \vdash \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{WI}_{h,k}$$

such that, for all $i \in \{1, ..., p\}$, either $\sigma_i = \mathbf{WI}_{h_i,k_i}$ (so σ_i is linear), or $\bar{\sigma}_i = \{\mathbf{WI}_{h^i_1,k^i_1}, ..., \mathbf{WI}_{h^i_{q_i},k^i_{q_i}}\}$, for some $h, k, h_i, k_i, q_i, h^i_r, k^i_r$ $(1 \leq r \leq q_i)$.

Following an approach similar to the one in [4], we prove the following result:

Theorem 2. *The numerical functions definable in* ISTA *are all and only the numerical functions definable in* STA*.*

The consequences of this theorem are interesting and also quite surprising. The first one is that, as expected, ISTA is sound and complete with respect to PTIME. But also a further notion of completeness arises. In fact ISTA is polynomially complete with respect to strongly normalizing terms, i.e., all the numerical polynomial algorithms that can be expressed by a strongly normalizing term can be typed in it.

References

1. Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.
2. P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS 2004. IEEE Computer Society*, pages 266–275, 2004.
3. P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, 2009.
4. Antonio Bucciarelli, Adolfo Piperno, and Ivano Salvo. Intersection types and lambda-definability. *Mathematical Structures in Computer Science*, 13:15–53, 2003.
5. M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for λ-calculus. In *Computer Science Logic, 21st International Workshop, CSL 07, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.
6. Marco Gaboardi and Simona Ronchi Della Rocca. From light logics to type assignements: a case study. *Logic Journal of the IGPL, Special Issue on LSFA 2007*, 17:499 – 530, 2009.
7. J-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
8. Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.

A Formal Model of Asynchronous Broadcast Communication

Giorgio Delzanno and Riccardo Traverso

DIBRIS, Università di Genova, Italy
{Giorgio.Delzanno,Riccardo.Traverso}@unige.it

We present a mathematical model, called Asynchronous Broadcast Networks (ABN), of distributed computation based on topology-dependent and asynchronous communication. Our model combines three main features: a graph representation of a network configuration decoupled from the specification of individual process behavior, a topology-dependent semantics of synchronization, the use of local mailboxes to deliver messages to individual nodes. The resulting communication layer is similar to that of languages like AWN [9]. As in other protocol models like ω [16,17] and AHN [5], our main abstraction comes from considering protocols defined via a communicating finite-state automaton replicated on each node of the network.

Formally, we consider a finite set Σ of messages, and different disciplines for handling the mailbox (message buffer), e.g., unordered mailboxes that we represent as bags over Σ, and ordered mailboxes that we represent as words over Σ. The initial configuration is any graph in which all the nodes are in the initial control state and all local buffers are empty. Even if the set of control states is finite, there are infinitely many possible initial configurations. We next formalize the above intuition.

A mailbox structure is a tuple $\mathbb{M} = \langle \mathcal{M}, del?, add, del, [] \rangle$, where $\mathcal{M}$ is a denumerable set of elements denoting possible mailbox contents on some fixed finite alphabet Σ, and, for $a \in \Sigma$ and $m \in \mathcal{M}$: $add(a, m)$ denotes the mailbox obtained by adding a to m, $del?(a, m)$ is true if a can be removed from m; $del(a, m)$ denotes the mailbox obtained by removing a from m when possible, undefined otherwise. Finally, $[] \in \mathcal{M}$ denotes the empty mailbox. We call an element a of m *visible* when $del?(a, m) = true$. The semantics and corresponding properties change with the type of mailbox considered.

A protocol is defined by a process $\mathcal{P} = \langle Q, \Sigma, R, q_0 \rangle$, where Q is a finite set of control states, Σ is a finite message alphabet, $Act = \{\tau\} \cup \{!!a, ??a \mid a \in \Sigma\}$, $R \subseteq Q \times Act \times Q$ is the transition relation, $q_0 \in Q$ is an initial control state. The label τ represents the ability of performing an internal action, while $!!a$ [$??a$] represents the ability of broadcasting [receiving] a message $a \in \Sigma$. Configurations are undirected $Q \times \mathcal{M}$-graphs. A $Q \times \mathcal{M}$-graph γ is a tuple $\langle V, E, L \rangle$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges (self-loops are forbidden to model half-duplex communication), and $L : V \to Q \times \mathcal{M}$ is a labeling function.

We use the notation $u \sim_\gamma v$ and say that the vertices u and v are adjacent to one another in γ. We omit γ, and simply write $u \sim v$, when it is made clear by the context. We use $L(\gamma)$ to represent the set of labels in γ. The set of all possible configurations is denoted $\mathcal{C}$, while $\mathcal{C}_0 \subseteq \mathcal{C}$ is the set of all initial configurations, in which nodes always have the same label $\langle q_0, [] \rangle$.

Given the labeling L and the node v s.t. $L(v) = \langle q, m \rangle$, we define $L_s(v) = q$ (state component of $L(v)$) and $L_b(v) = m$ (buffer component of $L(v)$). Furthermore, for $\gamma = \langle V, E, L \rangle \in \mathcal{C}$, we use $L_s(\gamma)$ to denote the set $\{L_s(v) \mid v \in V\}$.

For $\mathbb{M} = \langle \mathcal{M}, del?, add, del, [] \rangle$, an Asynchronous Broadcast Network (ABN) associated to $\mathcal{P}$ is defined by its associated transition system $\mathcal{T}(\mathcal{P}, \mathbb{M}) = \langle \mathcal{C}, \Rightarrow_{\mathbb{M}}, \mathcal{C}_0 \rangle$, where $\Rightarrow_{\mathbb{M}} \subseteq \mathcal{C} \times \mathcal{C}$ is the transition relation defined next.

For $\gamma = \langle V, E, L \rangle$ and $\gamma' = \langle V, E, L' \rangle$, $\gamma \Rightarrow_{\mathbb{M}} \gamma'$ holds iff one of the following conditions on L and L' holds: *(local)* there exists $v \in V$ such that $(L_s(v), \tau, L'_s(v)) \in R$, $L_b(v) = L'_b(v)$, and $L(u) = L'(u)$ for each $u \in V \setminus \{v\}$; *(broadcast)* there exists $v \in V$ and $a \in \Sigma$ such that $(L_s(v), !!a, L'_s(v)) \in R$, $L_b(v) = L'_b(v)$ and for every $u \in V \setminus \{v\}$ if $u \sim v$ then $L'_b(u) = add(a, L_b(u))$ and $L_s(u) = L'_s(u)$, otherwise $L(u) = L'(u)$; *(receive)* there exists $v \in V$ and $a \in \Sigma$ such that $(L_s(v), ??a, L'_s(v)) \in R$, $del?(a, L_b(v))$ is satisfied, $L'_b(v) = del(a, L_b(v))$, and $L(u) = L'(u)$ for each $u \in V \setminus \{v\}$. A local transition only affects the state of the process that executes it, while a broadcast also adds the corresponding message to the mailboxes of all the neighbors of the sender. Notice that broadcast is never blocking for the sender. Receivers can read the message in different instants. This models asynchronous communication. A reception of a message a is blocking for the receiver whenever the buffer is empty or the visible elements are all different from a. If a is visible in the mailbox, the message is removed and the process moves to the next state.

An *execution* is a sequence $\gamma_0\gamma_1 \ldots$ such that γ_0 is an initial configuration, and $\gamma_i \Rightarrow_{\mathbb{M}} \gamma_{i+1}$ for $i \geq 0$. We use $\Rightarrow^*_{\mathbb{M}}$ to denote the reflexive and transitive closure of $\Rightarrow_{\mathbb{M}}$. Furthermore, we define the set of immediate predecessors of a set S of configurations as $pre(S) = \{\gamma \mid \gamma \Rightarrow_{\mathbb{M}} \gamma', \gamma' \in S\}$. We use pre^* to indicate the reflexive-transitive closure of pre.

Decision Problems The coverability problem parametric on the mailbox structure $\mathbb{M}$ is defined as follows. Given a protocol $\mathcal{P}$ with transition system $\mathcal{T}(\mathcal{P}, \mathbb{M}) = \langle \mathcal{C}, \Rightarrow_{\mathbb{M}}, \mathcal{C}_0 \rangle$ and a control state q, the coverability problem $COVER(\mathbb{M})$ states: are there two configurations $\gamma_0 \in \mathcal{C}_0$ and $\gamma_1 \in \mathcal{C}$ such that $\gamma_0 \Rightarrow^*_{\mathbb{M}} \gamma_1$ and $q \in L_s(\gamma_1)$?

Preliminary Results When local buffers are treated as bags of messages the coverability problem is decidable. For the proof, it is first possible to consider the restricted case of fully connected topologies. For fully connected topologies, we can then resort to the theory of well-structured transition systems (wsts) [1,10] and show that reachability of a given control state can be solved via a symbolic backward search algorithm. When mailboxes are ordered buffers, we obtain undecidability already in the case of fully connected topologies. Indeed, by using FIFO mailboxes, we give nodes the possibility of recognizing communication with multiple neighbors with the same role. We cannot use this feature to define discovery protocols as for the undecidability proof of synchronous broadcast given in [5], but we can simulate a counter machine by using FIFO mailboxes as circular queues for encoding counters and to block computations which may lead to incorrect results. The coverability problem becomes decidable when introducing non-deterministic message losses. We can exploit again the theory of wsts for this positive result. In an extended model in which a node can test if its mailbox is empty, we obtain undecidability with unordered bags and fully-connected topologies. We cannot rely on queues anymore to distinguish bad computations, but the emptiness test allows us to do it anyway. Detailed proofs of these results are available in the technical report [6].

Related Work Our analysis completes previous work on verification and expressiveness (w.r.t. coverability) of broadcast communication. More specifically, for synchronous broadcast communication, the coverability problem is decidable for fully connected graphs [8] and undecidable for arbitrary graphs in the AHN model of [5]. Broadcast in AHN is topology-dependent. Synchronous communication is used here to implement a discovery protocol that, by a careful control of interferences, allows individual nodes to infer precise information about their vicinity (e.g. the existence of one and only one neighbor with a certain role). The discovery protocol is a building block for more complex computations. In this paper we use similar ideas but reductions of different nature to obtain undecidability (e.g. we encode counters using mailboxes and not by using linked structures).

For variations of the synchronous semantics like those proposed in [11], intermittent nodes and non-atomic broadcast, coverability becomes decidable. The decidability results exploit however different proof techniques. Indeed, coverability with intermittent nodes can be decided by using a weaker model than Petri nets, whereas we need to resort to the theory of wsts with nested data structures (bags of tuples containing multisets) to show decidability for the unordered case. There seems to be no direct reduction from one model to the other. Furthermore, by either introducing ϵ-transitions or moving to the case of ordered mailboxes we obtain undecidability of the resulting model. Concerning other models of broadcast communication, we would like to mention the CBS process calculi by Prasad [14,15] for fully connected networks with synchronous broadcast communication, the ω-calculus by Singh et al. [16,17] for fully connected networks with synchronous broadcast communication, and the model with topology-dependent broadcast by Ene and Muntean [7]. More recently, a process algebra for different types of communication, including asynchronous broadcast, called AWN, has been proposed in [9]. Semantics that take into consideration interferences and conflicts during a transmission have been proposed in [13,12]. Verification of unreliable communicating FIFO systems have been studied in [2,3]. In [4] the authors consider different classes of topologies with mixed lossy and perfect channels [4]. Differently from all the previous works, we consider here coverability for parametric initial configurations for a distributed model with asynchronous broadcast. Furthermore, we also consider different policies to handle the message buffers (bags/queues) and as well as unreliability of the communication media.

Concerning possible refinement of the unordered case, we are currently considering an extension with identifiers where each node has a unique identifier that can be passed using broadcast messages and compared with equality. The introduction of the extended semantics with identifiers and value passing and the formal analysis of the coverability problem is left for an extended version of the work.

References

1. Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2. Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Inf. Comput.*, 130(1):71–90, 1996.
3. Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.
4. Pierre Chambart and Ph. Schnoebelen. Mixing lossy and perfect fifo channels. In *CONCUR*, pages 340–355, 2008.
5. Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, pages 313–327, 2010.
6. Giorgio Delzanno and Riccardo Traverso. On the coverability problem for asynchronous broadcast networks. Technical Report DISI-TR-12-05, Dip. Informatica e Scienze dell'Informazione, 2012.
7. Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *IPDPS*, page 149, 2001.
8. Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359, 1999.
9. Ansgar Fehnker, Rob J. van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks. In *ESOP*, pages 295–315, 2012.
10. Alain Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
11. Gianluigi Zavattaro Giorgio Delzanno, Arnaud Sangnier. Verification of ad hoc networks with node and communication failures. In *FORTE*, 2012.
12. Massimo Merro, Francesco Ballardin, and Eleonora Sibilio. A timed calculus for wireless systems. *Theor. Comput. Sci.*, 412(47):6585–6611, 2011.
13. Nicola Mezzetti and Davide Sangiorgi. Towards a calculus for wireless systems. *Electr. Notes Theor. Comput. Sci.*, 158:331–353, 2006.
14. K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.
15. K. V. S. Prasad. Broadcasting in time. In *COORDINATION*, pages 321–338, 1996.
16. Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. Query-based model checking of ad hoc network protocols. In *CONCUR*, pages 603–619, 2009.
17. Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.

h-quasi planar Drawings of Bounded Treewidth Graphs in Linear Area *

Emilio Di Giacomo[1], Walter Didimo[1], Giuseppe Liotta[1], Fabrizio Montecchiani[1]

Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo,didimo,liotta,montecchiani}@diei.unipg.it

Abstract. We study the problem of computing h-quasi planar drawings in linear area; in an h-quasi planar drawing the number of mutually crossing edges is bounded by a constant h. We prove that every n-vertex partial k-tree admits a straight-line h-quasi planar drawing in $O(n)$ area, where h depends on k but not on n. For specific sub-families of partial k-trees, we present ad-hoc algorithms that compute h-quasi planar drawings in linear area, such that h is significantly reduced with respect to the general result.

1 Introduction

Area requirement of graph layouts is a widely studied topic in Graph Drawing and Geometric Graph Theory. Many asymptotic bounds have been proven for a variety of graph families and drawing styles. One of the most fundamental results in this scenario establishes that every planar graph admits a planar straight-line grid drawing in $O(n^2)$ area and that this bound is worst-case optimal [5]. This has motivated a lot of work devoted to discover sub-families of planar graphs that admit planar straight-line drawings in $o(n^2)$ area. Unfortunately, sub-quadratic upper bounds are known only for trees [4] and outerplanar graphs [6], while super-linear lower bounds are known for series-parallel graphs [13].

Although planarity is one of the most desirable properties when drawing a graph, many real-world graphs are in fact non-planar. Furthermore, planarity often imposes severe limitations on the optimization of the drawing area, which may sometimes be overcome by allowing either "few" edge crossings or specific types of edge crossings that do not affect too much the drawing readability. So far, only a few papers have focused on computing non-planar layouts in sub-quadratic area. Wood proved that every k-colorable graph admits a non-planar straight-line grid drawing in linear area [15], which implies that planar graphs admit such a drawing. However, the technique by Wood does not provide any guarantee on the type and number of edge crossings. More recently, Angelini *et al.* provided techniques for constructing poly-line *large angle crossing drawings* (*LAC drawings*) of planar graphs in sub-quadratic area [1]. We recall that the study of drawings with large angle crossings started in [9].

In this paper we study the problem of computing linear area straight-line drawings of graphs with controlled *crossing complexity*, i.e., drawings where some types of edge

* The results presented in this extended abstract are described in a paper accepted for the 38^{th} International Workshop on Graph Theoretic Concepts in Computer Science.

crossings are forbidden. We study *h-quasi planar drawings*, i.e., drawings with at most $h-1$ mutually crossing edges; this measure of crossing complexity can be regarded as a sort of planarity relaxation. The combinatorial properties of h-quasi planar drawings have been widely investigated [12, 14]. The contributions of the paper are as follows: (i) We prove that every n-vertex partial k-tree (i.e., any graph with bounded treewidth) admits a straight-line h-quasi planar drawing in $O(n)$ area, where h depends on k but not on n (Section 3). (ii) For specific sub-families of partial k-trees (outerplanar graphs, flat series-parallel graphs, and proper simply-nested graphs), we can compute h-quasi planar drawings in $O(n)$ area with values of h significantly smaller than those obtained with the general technique (Section 4).

For reasons of space, all the proofs and technicalities are omitted.

2 Preliminaries

A *drawing Γ* of a graph G maps each vertex v of G to a point p_v on the plane, and each edge $e = (u, v)$ to a Jordan arc connecting p_u and p_v not passing through any other vertex; furthermore, any two edges have at most one point in common. If all edges are mapped to straight-line segments, Γ is called a *straight-line drawing* of G. If all vertices are mapped to points with integer coordinates, Γ is called a *grid drawing* of G. The *bounding box* of a straight-line grid drawing Γ is the minimum axis-aligned box containing the drawing. If the bounding box has side lengths $X-1$ and $Y-1$, then we say that Γ is a drawing with *area $X \times Y$*. A drawing Γ is *h-quasi planar* if it has less than h mutually crossing edges. A 3-quasi planar drawing is also called a *quasi planar drawing*.

For definitions about track layouts see Dujmović, Pór and Wood [11].

A *k-tree*, $k \in \mathbb{N}$, is defined as follows. The clique of size k is a k-tree; the graph obtained from a k-tree by adding a new vertex adjacent to each vertex of a clique of size k is also a k-tree. A *partial k-tree* is a subgraph of a k-tree. A graph has bounded *treewidth* if and only if it is a partial k-tree [3].

3 Compact h-quasi Planar Drawings of Partial k-trees

Lemma 1. *Let G be a graph with n vertices. If G admits a (c, t)-track layout, then G admits an h-quasi planar grid drawing in $O(t^3 n)$ area, where $h = c(t-1)+1$.*

Lemma 1 implies that every graph with constant track number admits an h-quasi planar grid drawing in linear area with h being a constant. Since it is known that partial k-trees have track number that is constant in n (although depending on k) [10], this implies that every partial k-tree admits an h-quasi planar grid drawing in linear area where the value of h does not depend on n. The current best upper bound on the track number of k-trees is given in [8]. Thus, every k-tree has an h_k-quasi planar drawing in $O(n)$ area with $h_k \in O(1)$. In order to improve the value of h_k, we exploit an algorithm that computes $(2, t)$-track layouts of k-trees, whose description is omitted.

Theorem 1. *Every partial k-tree with n vertices admits an h_k-quasi planar grid drawing in $O(t_k^3 n)$ area, where $h_k = 2t_k - 1$ and t_k is given by the following set of equations:*

$$\begin{aligned}
t_k &= (c_{k-1,k} + 1)t_{k-1} \\
c_{k,i} &= (c_{k-1,k} + 1)(c_{k-1,i} + \frac{c_{k-1,k}}{4} \sum_{j=1}^{i-1} c_{k-1,j} \cdot c_{k-1,i-j}) \quad (i = 1, \ldots, k+1) \\
c_{k,k+2} &= 0
\end{aligned} \qquad (1)$$

with $t_1 = 2$ and $c_{1,1} = 4$ and $c_{1,2} = 2$.

We can prove that the values of h_k given in Theorem 1 are smaller than those obtained by using the track number upper bound in [8]. In particular, every partial 2-tree (i.e., every series-parallel graph) admits an 11-quasi planar drawing in $O(n)$ area.

4 Improved Bounds for Specific Families of Planar Partial k-trees

It is known that outerplanar graphs are partial 2-trees [3]. We can prove that the value of h can be reduced from 11 to 3 for outerplanar graphs.

Theorem 2. *Every outerplanar graph with n vertices admits a quasi planar grid drawing in $O(n)$ area.*

A series-parallel graph, or SP-graph, is *flat* if it does not contain two nested parallel components. For an exact definition of flat SP-graphs and decomposition tree see [7]. We lower the value of h for flat SP-graphs from 11 to 5.

Theorem 3. *Every flat SP-graph with n vertices admits a 5-quasi planar grid drawing in $O(n)$ area.*

A *proper simply-nested graph* is a k-outerplanar graph such that the vertices of levels from 1 to k are chordless cycles [2]. It is known that k-outerplanar graphs have treewidth at most $3k - 1$ [3]. By using the technique of Section 3 we would obtain an h-quasi planar drawing in linear area with h that would be a function of the number of levels k. We show that h can be reduced to 3. We remark that proper simply-nested graphs may require quadratic area if we want a planar drawing.

Theorem 4. *Every proper simply-nested graph with n vertices admits a quasi planar grid drawing in $O(n)$ area.*

5 Concluding Remarks and Open Problems

In this paper we studied the problem of computing compact h-quasi planar drawings of partial k-trees. Indeed, our algorithms can be regarded as drawing techniques that produce drawings with optimal area and with bounded crossing complexity. This point of view is particularly interesting in the case of planar graphs. As recalled in the introduction, planar graphs can be drawn with either optimal crossing complexity (i.e., in a planar way), in which case they may require $\Omega(n^2)$ area [5], or with optimal $\Theta(n)$ area but without any guarantee on the crossing complexity [15]. These two extremal results naturally raise the following question: is it possible to compute an h-quasi planar drawing of a planar graph in $o(n^2)$ area and $h \in o(n)$? In Section 4 we showed that $O(n)$ area and $h \in O(1)$ can be simultaneously achieved for some families of planar graphs. In fact Lemma 1 combined with some known results can be used to give a positive answer to the above question even for general planar graphs.

Theorem 5. *Every planar graph with n vertices admits a $O(\log^{16} n)$-quasi planar grid drawing in $O(n \log^{48} n)$ area.*

The results in this paper give rise to several interesting open problems. Among them: (1) Reducing the value of h_k given by Equation 1 for other sub-families of partial k-trees. (2) Studying whether planar graphs admits h-quasi planar drawings in $O(n)$ area with $h \in o(n)$, possibly $h \in O(1)$. (3) Studying h-quasi planar drawings in linear area and aspect ratio $o(n)$.

References

1. P. Angelini, G. Di Battista, W. Didimo, F. Frati, S.-H. Hong, M. Kaufmann, G. Liotta, and A. Lubiw. RAC and LAC drawings of planar graphs in subquadratic area. In *ECG '11*, pages 125–128, 2011.
2. P. Angelini, G. Di Battista, M. Kaufmann, T. Mchedlidze, V. Roselli, and C. Squarcella. Small point sets for simply-nested planar graphs. In *Proc. of GD 2011*, volume 7034 of *LNCS*, pages 75–85. Springer, 2012.
3. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *TCS*, 209(1-2):1–45, 1998.
4. P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *CGTA*, 2:187–200, 1992.
5. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
6. G. Di Battista and F. Frati. Small area drawings of outerplanar graphs. *Algorithmica*, 54:25–53, 2009.
7. E. Di Giacomo. Drawing series-parallel graphs on restricted integer $3D$ grids. In *Proc. of GD 2003*, volume 2912 of *LNCS*, pages 238–246. Springer, 2004.
8. E. Di Giacomo, G. Liotta, and H. Meijer. Computing straight-line 3D grid drawings of graphs in linear volume. *CGTA*, 32(1):26 – 58, 2005.
9. W. Didimo, P. Eades, and G. Liotta. Drawing graphs with right angle crossings. *TCS*, 412(39):5156 – 5166, 2011.
10. V. Dujmović, P. Morin, and D. R. Wood. Layout of graphs with bounded tree-width. *SIAM J. on Comp.*, 34(3):553–579, 2005.
11. V. Dujmović, A. Pór, and D. R. Wood. Track layouts of graphs. *DMTCS*, 6(2):497–522, 2004.
12. J. Fox and J. Pach. Coloring k_k-free intersection graphs of geometric objects in the plane. In *Proc. of SCG '08*, pages 346–354. ACM, 2008.
13. F. Frati. Lower bounds on the area requirements of series-parallel graphs. *DMTCS*, 12(5):139–174, 2010.
14. A. Suk. k-quasi-planar graphs. In *Proc. of GD 2011*, volume 7034 of *LNCS*, pages 266–277. Springer, 2012.
15. D. R. Wood. Grid drawings of k-colourable graphs. *CGTA*, 30(1):25 – 28, 2005.

Graph Operations on Parity Games and Polynomial-Time Algorithms[⋆]

Christoph Dittmann, Stephan Kreutzer, Alexandru I. Tomescu[⋆⋆]

Chair for Logic and Semantics, Technical University Berlin
`christoph.dittmann@tu-berlin.de`, `stephan.kreutzer@tu-berlin.de`,
`alexandru.tomescu@mailbox.tu-berlin.de`

1 Introduction

Parity games (see below) are a type of 2-player games that are studied in the area of formal verification of systems by model checking. Deciding the winner in a parity game is polynomial time equivalent to the model checking problem of the modal μ-calculus (e.g., [3]). Another strong motivation lies in the fact that the exact complexity of solving parity games is a long-standing open problem, the currently best known algorithm being subexponential [5]. It is known that the problem is in the complexity class UP ∩ coUP [4].

In this paper we identify restricted classes of digraphs where the problem is solvable in polynomial time, following an approach from structural graph theory. We consider three standard graph operations: the join of two graphs, repeated pasting along vertices, and the addition of a vertex. Given a class $\mathcal{C}$ of digraphs on which we can solve parity games in polynomial time, we show that the same holds for the class obtained from $\mathcal{C}$ by applying once any of these three operations to its elements.

These results provide, in particular, polynomial time algorithms for parity games whose underlying graph is a tournament (i.e., an orientation of a complete graph), a complete bipartite graph, a block graph, or a block-cactus graph. These are classes where the problem was not known to be efficiently solvable.

Previous results concerning restricted classes of parity games which are solvable in polynomial time include classes of bounded tree-width [7], bounded DAG-width [1], and bounded clique-width [8].

Notation and Preliminaries. A *parity game* $P = (V, V_\bigcirc, V_\Box, E, \Omega)$ is a finite directed graph (V, E) with a partitioning of the nodes $V = V_\bigcirc \cup V_\Box$ equipped with a priority map $\Omega : V \to \mathbb{N}$. A play on P starts with a token placed on some vertex $v \in V$. If $v \in V_\bigcirc$, Player $\circ$ moves the token to a successor of v, otherwise $V_\Box$ moves it to a successor. If there is no successor, the respective player loses. If the play continues forever, Player $\circ$ wins the game if and only if the maximum priority that appears infinitely often is even.

[⋆] We refer the interested reader to the full version [2] of this extended abstract.

[⋆⋆] The third author gratefully acknowledges the support of the European Science Foundation, activity "Games for Design and Verification".

A *positional strategy* for Player $\circ$ is a map $\rho : V_{\bigcirc} \to V$ such that $\rho(v)$ is a successor of v for all v such that v has a successor. We only consider positional strategies in this paper. A play $v = v_0, v_1, v_2, \ldots$ *conforms* to ρ if $v_{i+1} = \rho(v_i)$ for all i such that $v_i \in V_{\bigcirc}$. A strategy ρ is a *winning strategy* for Player $\circ$ from vertex v if every play that starts at v and conforms to ρ is winning for Player $\circ$. We call the set of vertices $W_{\bigcirc}(P) \subseteq V$ from which Player $\circ$ has a positional winning strategy the *winning region* of Player $\circ$, similar for $W_{\Box}$ and Player $\Box$. We will write $W_{\bigcirc}$, $W_{\Box}$ if the game is clear from the context. Parity games are *positionally determined* in the sense that $W_{\bigcirc} \cup W_{\Box} = V$ and $W_{\bigcirc} \cap W_{\Box} = \emptyset$ [3].

Given $A \subseteq V$, we denote by $P \cap A$ the parity game restricted to the vertices in A, that is, $(V \cap A, V_{\bigcirc} \cap A, V_{\Box} \cap A, E \cap (A \times A), \Omega\restriction_A)$. Similarly, $P \setminus A$ stands for the game $P \cap (V \setminus A)$. Given a class of parity games $\mathcal{C}$, we say that $\mathcal{C}$ is *hereditary* if for all $P \in \mathcal{C}$ and all subsets A of vertices of P, we have $P \cap A \in \mathcal{C}$. If $i \in \{\circ, \Box\}$, we denote by $\bar{i}$ the element of $\{\circ, \Box\} \setminus \{i\}$.

2 Tournaments and Joins of Digraphs

We start by describing a polynomial-time algorithm for solving parity games on tournaments. In doing so, we observe that our algorithm can handle more general parity games. In particular, it can handle games with the sole requirement that between every vertex of Player $\circ$ and every vertex of Player $\Box$ there is an arc. This technique will then be generalized so that, as a very specific case, we obtain that parity games are solvable in polynomial-time on any biorientation of a complete bipartite graph. A *biorientation* of an undirected graph G is a directed graph G' with the same nodes as G such that for every edge $\{x, y\} \in E(G)$, the graph G' contains the arc (x, y), (y, x), or both.

We note that this result is not a special case of Obdržálek's polynomial time algorithm [8] for parity games of bounded directed clique-width because biorientations of complete graphs or complete bipartite graphs do not have bounded *directed* clique-width although their underlying undirected graphs have bounded clique-width.

We say that a digraph $D = (V, E)$, with a partition of its vertices $V = V_{\bigcirc} \cup V_{\Box}$, is a *weak tournament* if between every two vertices $v \in V_{\bigcirc}$, $w \in V_{\Box}$ we have that $(v, w) \in E$ or $(w, v) \in E$ (or both).

In Algorithm 1 on the next page, the function SOLVE-SINGLE-PLAYER-GAME solves single-player games in polynomial time (see [3]). We denote by $\mathrm{attr}_i(A)$ the set of vertices in V from which Player i has a strategy to enter A at least once and call it the *i-attractor set of A*. This notion is well-known [3] and stands at the basis of the exponential-time algorithms of McNaughton [6] and Zielonka [9].

Theorem 1. *Algorithm 1 correctly computes the winning regions of a parity game $P = (V, V_{\bigcirc}, V_{\Box}, E, \Omega)$ on a weak tournament and runs in time $O(|V|^4)$.*

Theorem 1 can be generalized to handle larger collections of digraphs, as long as the property that one of the winning regions induces a digraph on which we can efficiently solve parity games is maintained.

Algorithm 1: An algorithm for solving parity games on weak tournaments

```
SOLVE($P = (V, V_{\bigcirc}, V_{\Box}, E, \Omega)$)
    $(A_{\bigcirc}, A_{\Box}) \leftarrow$ SOLVE-SINGLE-PLAYER-GAME($P \cap V_{\bigcirc}$)
    if $A_{\bigcirc} \neq \emptyset$ then $(W_{\bigcirc}, W_{\Box}) \leftarrow$ SOLVE($P \setminus A_{\bigcirc}$); return $(W_{\bigcirc} \cup A_{\bigcirc}, W_{\Box})$
    $(B_{\bigcirc}, B_{\Box}) \leftarrow$ SOLVE-SINGLE-PLAYER-GAME($P \cap V_{\Box}$)
    if $B_{\Box} \neq \emptyset$ then $(W_{\bigcirc}, W_{\Box}) \leftarrow$ SOLVE($P \setminus B_{\Box}$); return $(W_{\bigcirc}, W_{\Box} \cup B_{\Box})$
    $d \leftarrow$ MAXIMUM-PRIORITY($\Omega$)
    $i \leftarrow \bigcirc$ if $d$ is even, $\Box$ otherwise
    $(C_{\bigcirc}, C_{\Box}) \leftarrow$ SOLVE($P \setminus \mathrm{attr}_i(\Omega^{-1}(d))$)
    if $C_{\bar{i}} \neq \emptyset$ then return $(W_i \leftarrow \emptyset, W_{\bar{i}} \leftarrow V)$
            else return $(W_i \leftarrow V, W_{\bar{i}} \leftarrow \emptyset)$
```

If $P = (V, V_{\bigcirc}, V_{\Box}, E, \Omega)$ and $P' = (V', V'_{\bigcirc}, V'_{\Box}, E', \Omega')$ are two parity games with $V \cap V' = \emptyset$, we say that parity game $P'' = (V'', V''_{\bigcirc}, V''_{\Box}, E'', \Omega'')$ is a *join* of P and P' (see Figure 1) if

- $V'' = V \cup V'$, $V''_{\bigcirc} = V_{\bigcirc} \cup V'_{\bigcirc}$, $V''_{\Box} = V_{\Box} \cup V'_{\Box}$,
- $E'' = E \cup E' \cup E^*$, where $E^* \subseteq (V \times V') \cup (V' \times V)$ contains at least one arc (x, y) or (y, x) for all $x \in V$, $y \in V'$,
- and the vertices of P'' have the same priorities as they have in P and P'.

Given two classes of parity games $\mathcal{C}$ and $\mathcal{C}'$, we denote by $\mathsf{Join}(\mathcal{C}, \mathcal{C}')$ the class $\mathsf{Join}(\mathcal{C}, \mathcal{C}') := \{P'' \mid P'' \text{ is a join of } P \in \mathcal{C} \text{ and } P' \in \mathcal{C}'\}$.

Theorem 2. *If $\mathcal{C}$ and $\mathcal{C}'$ are hereditary classes of parity games that we can solve in polynomial time, then there is an algorithm for solving parity games in polynomial time on all games $P'' \in \mathsf{Join}(\mathcal{C}, \mathcal{C}')$, assuming a decomposition of P'' as a join of $P \in \mathcal{C}$ and $P' \in \mathcal{C}'$ is given.*

3 Pasting of Parity Games and Adding a Single Vertex

Let P, P' be two parity games on disjoint vertex sets and let v and v' be vertices of P and P', respectively. Assume that v, v' have the same priority and belong to the same player. We say that a game P'' is obtained by *pasting* P, P' at v, v' if P'' is the disjoint copy of P and P' with v, v' identified (see Figure 1). Given a class of parity games $\mathcal{C}$, we denote by $P(\mathcal{C})$ the class of games obtained by repeated pasting of a finite number of games from $\mathcal{C}$.

Theorem 3. *If $\mathcal{C}$ is a hereditary class of parity games that can be solved in polynomial time, then there is a polynomial time algorithm for solving parity games in $P(\mathcal{C})$.*

As a corollary of Theorems 1 and 3, we can solve parity games in polynomial time on any orientation of a *block-cactus graph*, that is, a graph whose maximal 2-connected components are cliques or cycles.

Our last result states that if P is a parity game and v a vertex such that $P \setminus \{v\}$ can be solved in polynomial time, then we can solve P in polynomial

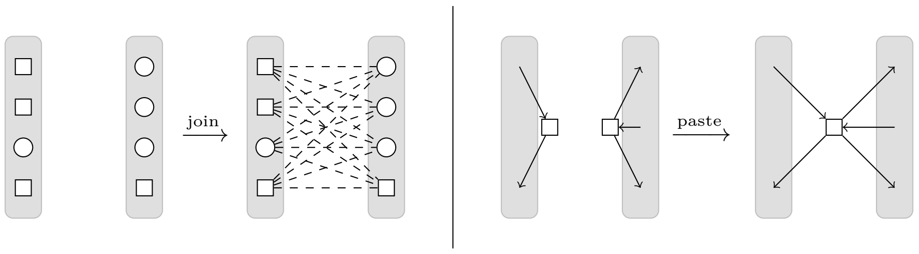


Fig. 1. The join and paste operations. The dashed lines represent necessary edges.

time. More formally, if $\mathcal{C}$ is a class of parity games, then $\mathcal{C}^+$ is the class obtained by adding a single vertex to every graph in $\mathcal{C}$ in any possible way.

Theorem 4. *If $\mathcal{C}$ is a hereditary class of games such that the decision problem (i.e., $P \in \mathcal{C}$?) is in polynomial time and games in $\mathcal{C}$ are solvable in polynomial time, then games in $\mathcal{C}^+$ are solvable in polynomial time.*

This theorem implies, for example, that if parity games can be solved in polynomial time on planar graphs, then they can also be solved in polynomial time on apex graphs, which are planar graphs with one additional vertex.

4 Conclusions

We have presented some graph operations that preserve the solvability of parity games in polynomial time. Generalizing this approach to more graph operations that generate larger classes of graphs is a possible line of future research.

References

1. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: DAG-Width and Parity Games. In: STACS. LNCS, vol. 3884, pp. 524–536. Springer (2006)
2. Dittmann, C., Kreutzer, S., Tomescu, A.I.: Graph operations on parity games and polynomial-time algorithms. pre-print arXiv:1208.1640 (2012)
3. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500. Springer (2002)
4. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Inf. Process. Lett. 68(3), 119–124 (1998)
5. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: SODA '06. pp. 117–123 (2006)
6. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Logic 65(2), 149–184 (1993)
7. Obdržálek, J.: Fast Mu-Calculus Model Checking when Tree-Width Is Bounded. In: CAV '03. LNCS, vol. 2725, pp. 80–92. Springer (2003)
8. Obdržálek, J.: Clique-width and parity games. In: Duparc, J., Henzinger, T.A. (eds.) CSL '07. LNCS, vol. 4646, pp. 54–68. Springer (2007)
9. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. 200(1-2), 135–183 (1998)

A Characterization of Bispecial Sturmian Words (extended abstract)

Gabriele Fici

I3S, CNRS & Université Nice Sophia Antipolis, France
`fici@i3s.unice.fr`

Abstract. We show that bispecial Sturmian words are exactly the maximal internal factors of Christoffel words. This result is an extension of the known relation between central words and primitive Christoffel words. Our characterization allows us to give an enumerative formula for bispecial Sturmian words.

1 Introduction

Sturmian words are non-periodic infinite words of minimal factor complexity. They are characterized by the property of having exactly $n+1$ distinct factors of length n for every $n \geq 0$ (and therefore are binary words) [5]. The set *St* of finite factors of Sturmian words coincides with the set of binary *balanced* words, i.e., binary words having the property that any two factors of the same length have the same number of occurrences of each letter up to one. If one considers extendibility within the set *St*, one can define *left special Sturmian words* (resp. *right special Sturmian words*) [4] as those words w over the alphabet $\Sigma = \{a, b\}$ such that aw and bw (resp. wa and wb) are both Sturmian words. The Sturmian words that are both left special and right special are called *bispecial Sturmian words*. They are of two kinds: *strictly bispecial Sturmian words*, that are the words w such that awa, awb, bwa and bwb are all Sturmian words, or *weakly bispecial Sturmian words* otherwise. Strictly bispecial Sturmian words have been deeply studied (see for example [2, 4]) because they play a central role in the theory of Sturmian words. They are also called *central words*. Weakly bispecial Sturmian words, instead, received less attention.

One important field in which Sturmian words arise naturally is discrete geometry. Indeed, Sturmian words can be viewed as digital approximations of straight lines in the Euclidean plane. It is known that given a point (p, q) in the discrete plane $\mathbb{Z} \times \mathbb{Z}$, with $p, q > 0$, there exists a unique path that approximates from below (resp. from above) the segment joining the origin $(0, 0)$ to the point (p, q). This path, represented as a concatenation of horizontal and vertical unitary segments, is called the *lower (resp. upper) Christoffel word* associated to the pair (p, q). If one encodes horizontal and vertical unitary segments with the letters a and b respectively, a lower (resp. upper) Christoffel word is always a word of the form awb (resp. bwa), for some $w \in \Sigma^*$. If (and only if) p and q are coprime, the associated Christoffel word is primitive (that is, it is not the power of a shorter word). It is known that a word w is a strictly bispecial Sturmian word if and only if awb is a primitive lower Christoffel word (or, equivalently, if and only if bwa is a primitive upper Christoffel word). As a main result of this paper, we show that this correspondence holds in general between bispecial Sturmian words and Christoffel words. That is, we prove (in Theorem 2) that w is a bispecial Sturmian word if and only if there exist letters x, y in $\{a, b\}$ such that xwy is a Christoffel word. This characterization allows us to prove an enumerative formula for bispecial Sturmian words (Corollary 1).

2 Sturmian words and Christoffel words

A finite word w over $\Sigma = \{a, b\}$ is Sturmian if and only if for any $u, v \in Fact(w)$ such that $|u| = |v|$ one has $||u|_a - |v|_a| \leq 1$. We let St denote the set of finite Sturmian words.

Let w be a finite Sturmian word. The following definitions are in [4].

Definition 1. *A word $w \in \Sigma^*$ is a left special (resp. right special) Sturmian word if $aw, bw \in St$ (resp. if $wa, wb \in St$). A bispecial Sturmian word is a Sturmian word that is both left special and right special. Moreover, a bispecial Sturmian word is strictly bispecial if awa, awb, bwa and bwb are all Sturmian word; otherwise it is non-strictly bispecial.*

We let *LS*, *RS*, *BS*, *SBS* and *NBS* denote, respectively, the sets of left special, right special, bispecial, strictly bispecial and non-strictly bispecial Sturmian words.

The following lemma is a reformulation of a result of de Luca [3].

Lemma 1. *Let w be a word over Σ. Then $w \in LS$ (resp. $w \in RS$) if and only if w is a prefix (resp. a suffix) of a word in SBS.*

Given a bispecial Sturmian word, the simplest criterion to determine if it is strictly or non-strictly bispecial is provided by the following nice characterization [4]:

Proposition 1. *A bispecial Sturmian word is strictly bispecial if and only if it is a palindrome.*

Using the results in [4], one can derive the following classification of Sturmian words with respect to their extendibility.

Proposition 2. *Let w be a Sturmian word. Then:*

- $|\Sigma w \Sigma \cap St| = 4$ *if and only if w is strictly bispecial;*
- $|\Sigma w \Sigma \cap St| = 3$ *if and only if w is non-strictly bispecial;*
- $|\Sigma w \Sigma \cap St| = 2$ *if and only if w is left special or right special but not bispecial;*
- $|\Sigma w \Sigma \cap St| = 1$ *if and only if w is neither left special nor right special.*

We now recall the definition of central word [4].

Definition 2. *A word over Σ is central if it has two coprime periods p and q and length equal to $p + q - 2$.*

We have the following remarkable result [4]:

Proposition 3. *A word over Σ is a strictly bispecial Sturmian word if and only if it is a central word.*

Another class of finite words, strictly related to the previous ones, is that of Christoffel words.

Definition 3. *Let $n > 1$ and $p, q > 0$ be integers such that $p+q = n$. The lower Christoffel word $w_{p,q}$ is the word defined for $1 \leq i \leq n$ by*

$$w_{p,q}[i] = \begin{cases} a \textit{ if } iq \textit{ mod}(n) > (i-1)q \textit{ mod}(n), \\ b \textit{ if } iq \textit{ mod}(n) < (i-1)q \textit{ mod}(n). \end{cases}$$

If one draws a word in the discrete grid $\mathbb{Z} \times \mathbb{Z}$ by encoding each a with a horizontal unitary segment and each b with a vertical unitary segment, the lower Christoffel word $w_{p,q}$ is in fact the best grid approximation from below of the segment joining $(0, 0)$ to (p, q), and has slope q/p, that is, $|w|_a = p$ and $|w|_b = q$.

Analogously, one can define the upper Christoffel word $w'_{p,q}$ by

$$w'_{p,q}[i] = \begin{cases} a \text{ if } ip \text{ mod}(n) < (i-1)p \text{ mod}(n), \\ b \text{ if } ip \text{ mod}(n) > (i-1)p \text{ mod}(n). \end{cases}$$

Of course, the upper Christoffel word $w'_{p,q}$ is the best grid approximation from above of the segment joining $(0, 0)$ to (p, q).

The next result follows from elementary geometrical considerations.

Lemma 2. *For every pair (p, q) the word $w'_{p,q}$ is the reversal of the word $w_{p,q}$.*

If (and only if) p and q are coprime, the Christoffel word $w_{p,q}$ intersects the segment joining $(0, 0)$ to (p, q) only at the end points, and is a primitive word. Moreover, one can prove that $w_{p,q} = aub$ and $w'_{p,q} = bua$ for a palindrome u. Since u is a bispecial Sturmian word and it is a palindrome, u is a strictly bispecial Sturmian word (by Proposition 1). Conversely, given a strictly bispecial Sturmian word u, u is a central word (by Proposition 3), and therefore has two coprime periods p, q and length equal to $p + q - 2$. Indeed, it can be proved that $aub = w_{p,q}$ and $bua = w'_{p,q}$. The previous properties can be summarized in the following theorem (cf. [1]):

Theorem 1. *$SBS = \{w \mid xwy$ is a primitive Christoffel word, $x, y \in \Sigma\}$.*

If instead p and q are not coprime, then there exist coprime integers p', q' such that $p = rp'$, $q = rq'$, for an integer $r > 1$. In this case, we have $w_{p,q} = (w_{p',q'})^r$, that is, $w_{p,q}$ is a power of a primitive Christoffel word. Hence, there exists a central Sturmian word u such that $w_{p,q} = (aub)^r$ and $w'_{p,q} = (bua)^r$. So, we have:

Lemma 3. *The word xwy, $x \neq y \in \Sigma$, is a Christoffel word if and only if $w = (uyx)^n u$, for an integer $n \geq 0$ and a central word u. Moreover, xwy is a primitive Christoffel word if and only if $n = 0$.*

Recall from [3] that the right (resp. left) palindromic closure of a word w is the (unique) shortest palindrome $w^{(+)}$ (resp. $w^{(-)}$) such that w is a prefix of $w^{(+)}$ (resp. a suffix of $w^{(-)}$).

Lemma 4. *Let xwy be a Christoffel word, $x, y \in \Sigma$. Then $w^{(+)}$ and $w^{(-)}$ are central words.*

Theorem 2. *$BS = \{w \mid xwy$ is a Christoffel word, $x, y \in \Sigma\}$.*

Proof. (Sketch) Let xwy be a Christoffel word, $x, y \in \Sigma$. Then, by Lemma 3, w is of the form $w = (uyx)^n u$, $n \geq 0$, for a central word u. By Lemma 4, w is a prefix of the central word $w^{(+)}$ and a suffix of the central word $w^{(-)}$, and therefore, by Lemma 1 and Proposition 3, w is a bispecial Sturmian word.

Conversely, let w be a bispecial Sturmian word. If w is strictly bispecial, then w is a central word by Proposition 3, and xwy is a (primitive) Christoffel word by Theorem 1. So suppose $w \in NBS$. By Lemma 3, it is enough to prove that w is of the form $w = (uyx)^n u$, $n \geq 1$, for a central word u and letters $x \neq y$. This can be proven by contradiction using the property of balanceness of Sturmian words. □

3 Enumeration of bispecial Sturmian words

It is known [4] that $SBS(n) = \phi(n+2)$. Therefore, in order to find an enumerative formula for bispecial Sturmian words, we only have to enumerate the non-strictly bispecial Sturmian words. We do this in the next proposition.

Proposition 4. *For every $n > 1$, one has*

$$NBS(n) = 2\,(n + 1 - \phi(n + 2))$$

Corollary 1. *For every $n \geq 0$, there are $2(n+1) - \phi(n+2)$ bispecial Sturmian words of length n.*

References

1. J. Berstel and A. de Luca. Sturmian Words, Lyndon Words and Trees. *Theoret. Comput. Sci.*, 178(1-2):171–203, 1997.
2. A. Carpi and A. de Luca. Central Sturmian Words: Recent Developments. In *Developments in Language Theory, 9th International Conference, DLT 2005*, volume 3572 of *Lecture Notes in Comput. Sci.*, pages 36–56. Springer, 2005.
3. A. de Luca. Sturmian Words: Structure, Combinatorics, and Their Arithmetics. *Theoret. Comput. Sci.*, 183(1):45–82, 1997.
4. A. de Luca and F. Mignosi. Some combinatorial properties of Sturmian words. *Theoret. Comput. Sci.*, 136(2):361–385, 1994.
5. M. Morse and G. A. Hedlund. Symbolic dynamics II: Sturmian Trajectories. *Amer. J. Math.*, 62:1–42, 1940.

Words with the Smallest Number of Closed Factors

Gabriele Fici[1] and Zsuzsanna Lipták[2]

[1] I3S, CNRS & Université Nice Sophia Antipolis, France, gabriele.fici@unice.fr
[2] Università di Verona, Italy, zsuzsanna.liptak@univr.it

Abstract. A word is closed if it contains a factor that occurs both as a prefix and as a suffix but does not have internal occurrences. We show that any word of length n contains at least $n+1$ closed factors (i.e., factors that are closed words). We investigate the language $\mathcal{L}$ of words over the alphabet $\{a, b\}$ containing exactly $n + 1$ closed factors. We show that a word belongs to $\mathcal{L}$ if and only if its closed factors and its palindromic factors coincide (and therefore the words in $\mathcal{L}$ are rich words). We also show that $\mathcal{L}$ coincides with the language of conjugates of words in a^*b^*.

Keywords: Closed word, closed factor, rich word, bitonic word.

1 Introduction

A *word* is a finite sequence of elements from a finite set Σ. We refer to the elements of Σ as *letters* and to Σ as the *alphabet*. The i-th letter of a word w is denoted by w_i. Given a word $w = w_1w_2 \cdots w_n$, with $w_i \in \Sigma$ for $1 \leq i \leq n$, the nonnegative integer n is the *length* of w, denoted by $|w|$. The empty word has length zero and is denoted by ε. The set of all words over Σ is denoted by Σ^*. Any subset of Σ^* is called a *language*.

A *prefix* (resp. a *suffix*) of a word w is any word u such that $w = uz$ (resp. $w = zu$) for some word z. A *factor* of w is a prefix of a suffix (or, equivalently, a suffix of a prefix) of w. The set of prefixes, suffixes and factors of the word w are denoted by *Pref*(w), *Suff*(w) and *Fact*(w) respectively. A *border* of a word w is any word in *Pref*(w) $\cap$ *Suff*(w) different from w. From the definitions, we have that ε occurs as a prefix, suffix and factor in any word. An *occurrence* of a factor u in a word w is a pair of positions (i, j) such that $w_i \dots w_j = u$. An occurrence is *internal* if $i > 1$ and $j < |w|$.

The word $\tilde{w} = w_nw_{n-1} \cdots w_1$ is called the *reversal* (or *mirror image*) of w. A *palindrome* is a word w such that $\tilde{w} = w$. In particular, the empty word is a palindrome. A *conjugate* of a word w is any word of the form vu such that $uv = w$, for some $u, v \in \Sigma^*$. A conjugate of a word w is also called a *rotation* of w.

Let w be a word. We denote by $PAL(w)$ the set of factors of w that are palindromes. Droubay, Justin and Pirillo showed [4] that for any word w of length n, one has $|PAL(w)| \leq n + 1$. Consequently, w is called *rich* [6] (or *full* [1]) if

$|PAL(w)| = n + 1$, that is, if it contains the largest number of palindromes a word of length n can contain.

A language L is called *factorial* if $L = Fact(L)$, i.e., if L contains all the factors of its words. A language L is *extendible* if for every word $w \in L$, there exist letters $a, b \in \Sigma$ such that $awb \in L$. The language of rich words over a fixed alphabet Σ is an example of factorial and extendible language.

We now recall the definition of closed word [5]:

Definition 1. *A word w is* closed *if it is empty or has a factor occurring exactly twice in w, as a prefix and as a suffix of w.*

The word aba is closed, since its factor a appears only as a prefix and as a suffix. The word $abaa$, instead, is not closed. Note that for any letter $a \in \Sigma$ and for any $n > 0$, the word a^n is closed, a^{n-1} being a factor occurring only as a prefix and as a suffix in it. More generally, any word w that is a power of a shorter word, i.e., $w = v^n$ for a non-empty v and $n > 1$, is closed.

There exist closed words that are not palindromes, for example the word $abab$. Conversely, there exist palindromes that are not closed, but it is worth noticing that a shortest palindrome over a two-letter alphabet that is not closed has length 14. An example is $aabbabaababbaa$.

Remark 1. The notion of closed word is closely related to the concept of *complete return* to a factor, as considered in [6]. A complete return to the factor u in a word w is any factor of w having exactly two occurrences of u, one as a prefix and one as a suffix. Hence w is closed if and only if it is a complete return to one of its factors; such a factor is clearly both the longest repeated prefix and the longest repeated suffix of w (that is, the longest border of w). The notion of closed word is also equivalent to that of *periodic-like* word [3]. A word w is periodic-like if its longest repeated prefix does not have two occurrences in w followed by different letters.

Observation 1 *Let w be a non-empty word over Σ. The following characterizations of closed words follow easily from the definition:*

1. *w has a factor occurring exactly twice in w, as a prefix and as a suffix of w;*
2. *the longest repeated prefix of w does not have internal occurrences in w, that is, occurs in w only as a prefix and as a suffix;*
3. *the longest repeated suffix of w does not have internal occurrences in w, that is, occurs in w only as a suffix and as a prefix;*
4. *the longest repeated prefix of w does not have two occurrences in w followed by different letters;*
5. *the longest repeated suffix of w does not have two occurrences in w preceded by different letters;*
6. *w has a border that does not have internal occurrences in w;*
7. *the longest border of w does not have internal occurrences in w;*
8. *w is the complete return to its longest prefix;*
9. *w is the complete return to its longest border.*

For more details on closed words and related results cf. [3, 2, 5].

2 Closed factors

Let w be a word. A factor of w that is a closed word is called a *closed factor* of w. The set of closed factors of the word w is denoted by $C(w)$.

Lemma 1. *For any non-empty word w of length n, one has $|C(w)| \geq n+1$.*

Lemma 2. *Let u, v be non-empty words. Then $|C(u)| + |C(v)| \leq |C(uv)| + 1$.*

Proposition 1. *Let w be a non-empty word of length n. If $C(w) \subseteq PAL(w)$, then $C(w) = PAL(w)$ and $|C(w)| = |PAL(w)| = n+1$. In particular, w is a rich word.*

Bucci et al. showed [2, Proposition 4.3] that a word w is rich if and only if every closed factor v of w has the property that the longest palindromic prefix (or suffix) of v is unrepeated in v. Moreover, if w is a palindrome, then it is rich if and only if $PAL(w) \subseteq C(w)$ [2, Corollary 5.2].

In Section 4, we shall prove that the condition $PAL(w) = C(w)$ characterizes the words having the smallest number of closed factors over a binary alphabet.

3 Words with the smallest number of closed factors

By Lemma 1, we have that $n+1$ is a lower bound on the number of closed factors of a word of length $n > 0$. We introduce the following definition:

Definition 2. *A word $w \in \Sigma^*$ is* C-poor *if $|C(w)| = |w| + 1$. We also set*

$$\mathcal{L}_\Sigma = \{w \in \Sigma^* : |C(w)| = |w| + 1\}$$

the language of C-poor words over the alphabet Σ.

Remark 2. If $|\Sigma| = 1$, then $\mathcal{L}_\Sigma = \Sigma^*$. So in what follows we will suppose $|\Sigma| \geq 2$.

Lemma 3. *The language $\mathcal{L}_\Sigma$ is closed under reversal.*

Lemma 4. *Let w be a C-poor word over the alphabet Σ and $x \in \Sigma$. The word wx (resp. xw) is C-poor if and only if it has a unique suffix (resp. prefix) that is closed and is not a factor of w.*

Proposition 2. *A word $w \in \Sigma^*$ belongs to $\mathcal{L}_\Sigma$ if and only if every factor of w belongs to $\mathcal{L}_\Sigma$. That is, $\mathcal{L}_\Sigma$ is a factorial language.*

4 Binary words

In this section, we fix the alphabet $\Sigma = \{a, b\}$. For simplicity of exposition, we will denote the language of C-poor words over $\{a, b\}$ by $\mathcal{L}$ rather than by $\mathcal{L}_{\{a,b\}}$. We first recall the definition of bitonic word.

Definition 3. *A word $w \in \{a,b\}^*$ is* bitonic *if it is a conjugate of a word in a^*b^*, i.e., if it is of the form $a^ib^ja^k$ or $b^ia^jb^k$ for integers $i,j,k \geq 0$.*

The following lemma, the proof of which is straightforward, relates bitonic words to closed factors.

Lemma 5. *If a word $w \in \{a,b\}^*$ does not contain any complete return to ab or ba as a factor, then it is bitonic.*

Lemma 6. *Let w be a bitonic word. Then $C(w) \subseteq PAL(w)$.*

Thus, by Proposition 1, any bitonic word w of length $n > 0$ contains exactly $n+1$ closed factors and so is a C-poor word. In the rest of the section we shall prove the converse, that is, we shall prove that if w is a C-poor word over $\{a,b\}$, then w is bitonic.

Consider the word $w = abab$. The word w does not belong to $\mathcal{L}$, since it has 6 closed factors, namely ε, a, b, aba, bab and $abab$. In fact, it has two suffixes (bab and $abab$) that are closed and do not appear before in w, and hence by Proposition 2 it cannot belong to $\mathcal{L}$. More generally, any word u such that u is the complete return to ab or ba does not belong to $\mathcal{L}$ for the same reason. So, using Proposition 2, we get:

Lemma 7. *If $w \in \mathcal{L}$, then w does not contain any complete return to ab or ba as a factor.*

We summarize the characterizations of $\mathcal{L}$ in the following theorem:

Theorem 1. *Let $w \in \{a,b\}^*$. The following are equivalent:*

1. *$w \in \mathcal{L}$;*
2. *$C(w) = PAL(w)$;*
3. *$C(w) \subseteq PAL(w)$;*
4. *w is a bitonic word;*
5. *w does not contain any complete return to ab or ba.*

Proof. 1) $\Rightarrow$ 5) by Lemma 7; 5) $\Rightarrow$ 4) by Lemma 5; 4) $\Rightarrow$ 3) by Lemma 6; finally, 3) $\Rightarrow$ 2) and 2) $\Rightarrow$ 1) by Proposition 1. □

So, by Theorem 1 and Proposition 1, every word in $\mathcal{L}$ is rich. Notice that there exist rich words that are not in $\mathcal{L}$, for example the word $w = abab$, which has 6 closed factors, namely ε, a, b, aba, bab and $abab$. Another consequence of Theorem 1 is that $\mathcal{L}$ is extendible, since the language of bitonic words is clearly extendible. Thus, the language $\mathcal{L}$ is a factorial and extendible subset of the language of (binary) rich words.

It further follows from Theorem 1 that $\mathcal{L}$ is a regular language, since the language of the conjugates of words of a regular language is regular [7]. In the following proposition we exhibit a closed enumerative formula for the language $\mathcal{L}$.

Proposition 3. *For every $n > 0$, there are exactly $n^2 - n + 2$ distinct words in $\mathcal{L}$.*

Proof. Each of the $n - 1$ words of length $n > 0$ in a^+b^+ has n distinct rotations, while for the words a^n and b^n all the rotations coincide. Thus, there are $n(n - 1) + 2$ bitonic words of length n, and the statement follows from Theorem 1.

5 Conclusion and open problems

In this paper we studied words with the smallest number of closed factors, which we referred to as C-poor words. We gave some interesting characterizations in the case of a binary alphabet. In particular, we showed that the language of binary C-poor words coincides with the language of bitonic words. A natural direction of further investigation is finding a characterization for C-poor words over alphabets larger than 2.

An enumerative formula for rich words is not known, not even in the binary case. A possible approach to this problem is to separate rich words in subclasses to be enumerated separately. Our enumerative formula for C-poor words given in Proposition 3 constitutes a step in this direction.

References

1. S. Brlek, S. Hamel, M. Nivat, and C. Reutenauer. On the palindromic complexity of infinite words. *Internat. J. Found. Comput. Sci.*, 15:293–306, 2004.
2. M. Bucci, A. de Luca, and A. De Luca. Rich and Periodic-Like Words. In *DLT 2009, 13th International Conference on Developments in Language Theory*, volume 5583 of *Lecture Notes in Comput. Sci.*, pages 145–155. Springer, 2009.
3. A. Carpi and A. de Luca. Periodic-like words, periodicity and boxes. *Acta Inform.*, 37:597–618, 2001.
4. X. Droubay, J. Justin, and G. Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theoret. Comput. Sci.*, 255(1-2):539–553, 2001.
5. G. Fici. A Classification of Trapezoidal Words. In *WORDS 2011, 8th International Conference on Words*, number 63 in Electronic Proceedings in Theoretical Computer Science, pages 129–137, 2011.
6. A. Glen, J. Justin, S. Widmer, and L. Q. Zamboni. Palindromic richness. *European J. Combin.*, 30:510–531, 2009.
7. J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001.

Time Modalities over Many-valued Logics

Nicholas Fiorentini[1], Achille Frigeri[1], Liliana Pasquale[2], and Paola Spoletini[3]

[1] Politecnico di Milano
fiorentini@studenti.polimi.it,achille.frigeri@polimi.it
[2] University of Limerick liliana.pasquale@lero.ie
[3] Università dell'Insubria paola.spoletini@uninsubria.it

1 Introduction

Model checking has been traditionally concerned on verifying a (critical) system against its specification, which is generally expressed in temporal logic. Despite this verification technique is mature, it becomes useless when the specification incorporates vagueness, especially for the temporal constraints. This is often the case when non-critical adaptive systems are considered. These systems may tolerate small violations or may need to be aware of the satisfaction degree of their specification for re-configuration purposes. We present FTL (Fuzzy-time Temporal Logic), an extension of LTL that relaxes the notion of time, and propose a verification technique to evaluate the truth degree of such vague temporal properties. Our verification technique has been implemented in a prototype and the experimental results are promising.

2 FTL: Fuzzy-Time Temporal Logic

FTL (Fuzzy-Time Temporal Logic) [1] is a language conceived to express specifications for system hampered by uncertainty and vagueness, extending LTL by adding a set of temporal operators to express vague temporal properties.

Syntax Let F be a numerable set of atomic (crisp or fuzzy) propositions, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ be the (fuzzy) logical connectives, and O and T the sets of unary and binary (fuzzy) temporal modalities. Then, φ belongs to the set Φ of *well-formed FTL formulae* (from now on, formulae), if it is defined as follows: $\varphi := p \mid \neg\varphi \mid \varphi \sim \varphi \mid \mathcal{O}\varphi \mid \varphi\mathcal{T}\varphi$, where $p \in F$, $\sim$ is a binary connective, $\mathcal{O} \in O$, and $\mathcal{T} \in T$. As unary operators we consider $\mathcal{X}$ (next), $\mathcal{S}oon$ (soon), $\mathcal{F}$ (eventually), $\mathcal{F}_t$ (eventually in the next t instants), $\mathcal{G}$ (always), $\mathcal{G}_t$ (always in the next t instants), $\mathcal{AG}$ (almost always), $\mathcal{AG}_t$ (almost always in the next t instants), $\mathcal{L}_t$ (lasts t instants), $\mathcal{W}_t$ (within t instants), where $t \in \mathbb{N}$; binary operators are $\mathcal{U}/\mathcal{U}_t$ (until/until with t instants), $\mathcal{AU}/\mathcal{AU}_t$ (almost until/almost until within t instants).

Semantics The semantics of a formula φ is defined w.r.t. a linear structure (S, s_0, π, L), where S is the set of states, s_0 is the initial state, π is an infinite path $\pi = s_0 s_1 \cdots \in S^\omega$, π^i is the suffix of π starting from the i-th position and

s^i is its first state. $L : S \to [0,1]^F$ is the *(fuzzy) labeling function* that assigns to each atomic proposition in F its corresponding evaluation at each state. Besides, we adopt an *avoiding function*, $\eta : \mathbb{Z} \to [0,1]$. We assume $\eta(i) = 1$ for all $i \leq 0$, η is strictly decreasing in $\{0, \ldots, n_\eta\}$ for some $n_\eta \in \mathbb{N}$, and $\eta(n') = 0$ for all $n' \geq n_\eta$. Since we are dealing with a multi-valued logic, we define the semantics of a formula via a *fuzzy satisfiability relation* $\models \subseteq S^\omega \times F \times [0,1]$, where $(\pi \models \varphi) = \nu \in [0,1]$ means that the truth degree of φ on π is ν. FTL is defined as a family of logics, where the semantics of connectives is given on a generic t-norm based logic [2]. We use the Gödel-Dummet interpretation for connectives, since this is t-norm based logic that can include the same interpretation of $\wedge$ and $\vee$ as in Zadeh logic. In particular, for an untimed sub-formula we have:

$$\begin{array}{ll}
(\pi^i \models p) = L(s^i)(p), & (\pi^i \models \varphi \wedge \psi) = \min\{(\pi^i \models \varphi), (\pi^i \models \psi)\} \\
(\pi^i \models \neg\varphi) = \begin{cases} 1, (\pi^i \models \varphi) = 0 \\ 0, (\pi^i \models \varphi) > 0 \end{cases} & (\pi^i \models \varphi \vee \psi) = \max\{(\pi^i \models \varphi), (\pi^i \models \psi)\} \\
 & (\pi^i \models \varphi \Rightarrow \psi) = \begin{cases} 1, (\pi^i \models \varphi) \leq (\pi^i \models \psi) \\ 0, (\pi^i \models \varphi) > (\pi^i \models \psi) \end{cases}
\end{array}$$

where $p \in F$, $i \in \mathbb{N}$. The semantics of temporal operators, except for $\mathcal{AG}_t$, $\mathcal{AG}$, $\mathcal{AU}_t$, and $\mathcal{AU}$, is summarized in Figure 1.

Φ	$\pi^i \models \Phi$	Φ	$\pi^i \models \Phi$
$\mathcal{X}\varphi$	$\pi^{i+1} \models \varphi$	$\mathcal{S}oon\,\varphi$	$\max\limits_{i<j\leq i+n_\eta} (\pi^j \models \varphi) \cdot \eta(j-i-1)$
$\mathcal{F}_t\varphi$	$\max\limits_{i\leq j\leq i+t} (\pi^j \models \varphi)$	$\mathcal{W}_t\varphi$	$\max\limits_{i\leq j<i+t+n_\eta} (\pi^j \models \varphi) \cdot \eta(j-t-i)$
$\mathcal{F}\varphi$	$\lim\limits_{t\to+\infty} (\pi^i \models \mathcal{F}_t\varphi)$	$\mathcal{L}_t\varphi$	$\max\limits_{0\leq j\leq \min\{n_\eta,t\}} (\pi^i \models \mathcal{G}_{t-j}\varphi) \cdot \eta(j)$
$\mathcal{G}_t\varphi$	$\min\limits_{i\leq j\leq i+t} (\pi^j \models \varphi)$	$\varphi\,\mathcal{U}_t\,\psi$	$\max\limits_{i\leq j\leq i+t} \{\min\{\pi^j \models \psi, \pi^i \models \mathcal{G}_{j-1}\varphi\}\}$
$\mathcal{G}\varphi$	$\lim\limits_{t\to+\infty} (\pi^i \models \mathcal{G}_t\varphi)$	$\varphi\,\mathcal{U}\,\psi$	$\lim\limits_{t\to+\infty} (\pi^i \models \varphi\,\mathcal{U}_t\,\psi)$

Fig. 1. Semantics for FTL temporal operators.

Informally, $\mathcal{X}$, $\mathcal{G}$ and $\mathcal{F}$ are interpreted as classical LTL operators, except in that the classical connectives used in their sub-formula are associated with their fuzzy interpretation. $\mathcal{G}_t$ and $\mathcal{F}_t$ are the bounded version of $\mathcal{G}$ and $\mathcal{F}$, respectively. $\mathcal{S}oon$ extends $\mathcal{X}$ by tolerating at most n_η time units delay. $\mathcal{W}_t$ means a property is supposed to hold in at least one of the next t time units or, possibly, in the next $t + n_\eta$ time units. An increasing penalization is applied in case a property holds after the t-th time unit. $\mathcal{L}_t$ expresses that a property should last for t consecutive time units. In case a property does not hold from a certain time unit $n \in [0,t]$, a penalization is given depending on the difference between n and t. $\mathcal{AG}$ evaluates a property by avoiding at most n_η worse cases (i.e., where a property is minimally satisfied). A penalization will be assigned according to the number of avoided worse cases (k). If more worse cases are avoided, penalization will be more severe. Hence, a tradeoff should be identified between the number of avoided worse cases and the assigned penalization. Formally, if $\mathcal{P}^k(\mathbb{N})$ is the set of subsets of $\mathbb{N}$ of cardinality k, then

$$(\pi^i \models \mathcal{AG}\,\varphi) = \max_{k\in\mathbb{N}} \max_{H\in\mathcal{P}^k(\mathbb{N})} \min_{\substack{j\geq i,\\ j\neq i+h,\\ h\in H}} (\pi^j \models \varphi\cdot\eta(k)).$$

$\mathcal{AG}_t$ is the bounded version of $\mathcal{AG}$, in which the index k is supposed to be in $\{0,\dots,\min n_\eta, t\}$. The semantics $\mathcal{AU}$ and $\mathcal{AU}_t$, is defined similarly to $\mathcal{U}$ and $\mathcal{U}_t$, by replacing the occurrence of $\mathcal{G}_{j-1}$ by $\mathcal{AG}_{j-1}$ (see Figure 1). Under the assumption that all events are crisp, FTL reduces to LTL, formally:

Theorem 1. *Let for all $p \in AP$ and $i \in \mathbb{N}$, $\pi^i \models p \in \{0,1\}$, and $\eta(1) = 0$. Then FTL reduces to LTL.*

We also provide an adequate set, i.e., a subset of its connectives and temporal operators that is sufficient to equivalently express any formula of the logic. Before, we need to introduce the extra operators $\odot^j$, for $1 \leq j < n_\eta$, whose semantics is defined by $(\pi^i \models \odot^j\varphi) = (\pi^i \models \varphi)\cdot\eta(j)$.

Theorem 2. *A set of adequate connectives is $\{\wedge, \Rightarrow, \mathcal{X}, \mathcal{U}, \mathcal{AU}, \odot^1, \dots, \odot^{n_\eta-1}\}$.*

3 Evaluating formulae

Describing systems with vague conditions FTL formulae are used to express properties of a system described via a Fuzzy discrete Timed Automata (FTA) [3]. FTA are an enriched version of timed automata (TA), introduced by Alur and Dill [4] to describe real-time systems. Analogously to TA, FTA have the capability of manipulating clocks, which evolve continuously and synchronously with absolute time. FTA extend classical finite automata to fuzzy events, which in many contexts are more suitable to describe realistic systems. The domain of our FTA is $\mathbb{N}$, since this is the same domain chosen for FTL. FTA are defined over a finite set of clocks, a finite set of crisp events, and a finite set of control variables. The range over which these variables vary represents the support for fuzzy events. The clock values change while the automaton stays in a location and can be reset when a transition is taken. Instead, control variables may change their value, within the limit of their range, both in a location or when a transition is taken. Each transition of this automaton is labeled with constraints on both clock values and fuzzy attributes, which may also be related to the value of control variables. Then, an FTA is a finite state automaton on infinite words in which locations are connected through Event-Condition-Action (ECA) rules. The condition of a ECA rule is the conjunction of a temporal and a fuzzy constraint (a fuzzy event is implicitly a fuzzy condition on the values of the attributes on a support). Starting from the source location, a transition is performed if the specified events occur and the conditions are satisfied. Once it is performed, the corresponding actions (i.e., reset of a subset of the variables in $T \cup F$ and the increasing or decreasing of a subset of the control variables in F) are performed as well, and the target location is reached. The values of control variables can also be changed in the location, defining the variation period. Analogously to TA, the semantics of FTA is described through a Timed Transition System (TTS), in which states represent a snapshot of the automaton during its evolution. Each state includes the current location, the value of the clocks in T, the set of events that just occurred, and the value of the control variables. The transitions between states can be of three types: discrete, purely timed or support timed. Discrete transitions represent the transition of the automaton, purely timed transitions describe the time passed within a location and support timed transitions represent the support variations within a location.

Evaluation algorithm An FTL formula is evaluated on a FTA not as true or false, but by associating to it a truth degree in $[0, 1]$.
The evaluation algorithm is composed of the following three phases:

1. Preprocessing of the FTL formula: the property is rewritten in terms of min and max and the parsing tree of the translated formula is annotated with the corresponding intervals under consideration;
2. Building a finite version of the TTS (FTTS) using temporal zones, as in [5];
3. Formula evaluation: the truth degree of the formula is computed on FTTS.

Step 1 is independent from step 2, but it needs to be performed before step 3. Steps 2–3 can be seen as the stages of a pipeline, since they can be executed in parallel and mutually synchronize on the inter-stage results.

4 Conclusions

We proposed an innovative approach for evaluating vague temporal formulae. Our technique computes the minimum and the maximum truth degree a given formula can have in an automata-based model (FTA) of the system. Each formula is expressed according to FTL, a fuzzy time temporal logic, and can assume any value in the interval $[0, 1]$. We provided a prototype that implements the evaluation technique, and performed a set of experiments on a simple case study. The preliminary results are encouraging and demonstrate the feasibility of the approach. As a future work we aim to further improve the performance of the checking technique and study its complexity bounds.

References

1. Frigeri, A., Pasquale, L., Spoletini, P.: Fuzzy Time in LTL. CoRR abs/1203.6278
2. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer (1998)
3. Fiorentini, N., Pasquale, L., Spoletini, P.: Evaluating vague temporal properties. In: Submitted to Formats 2012. (2012)
4. Alur, R., Dill, D.L.: A Theory of Timed Automata. TCS **126** (1994) 183–235
5. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. (2003) 87–124

A Reconstruction of a Type-and-Effect Analysis by Abstract Interpretation

Letterio Galletta

Dipartimento di Informatica - Università di Pisa
galletta@di.unipi.it

Abstract. Type-and-effect systems (TESs) have been widely exploited to specify static analyses of programs, for example to track computational side effects, exceptions and communications in concurrent programs. We adopt abstract interpretation techniques to reconstruct a TES developed to handle security problems of a multi-tier web language. In order to reconstruct this type-and-effect analysis we extended the Cousot's methodology used for type systems and the corresponding type inference algorithms by defining an abstract domain able to express types augmented by semantic annotations. This abstract domain is an extension of the Hindley's monotypes with a new kind of variables and constraints. We show that this abstract domain allows us to reconstruct different type-and-effect analyses by properly changing the set of monotypes and the shape of the constraints. In particular, we apply this approach to reconstruct the *Call-Tracking Analysis*. As usual with abstract interpretation, the analysis is correct by construction. The analyser has been implemented in OCaml.

1 Introduction

Type-and-effect systems (TESs) are a powerful extension of type systems which allows one to express general semantic properties and to statically reason about program execution. The underlying idea is to refine the type information so to express further intentional or extensional properties of the semantics of the program. In practice, TES compute the type of each program sentence and an approximate (but sound) description of its run-time behaviour.

Type systems (and the corresponding type inference algorithms) have been reconstructed as a hierarchy of abstract interpretations by Cousot [3]. In [5,6] we have extended the Cousot's methodology to reconstruct the TES used in [1] to handle security issues in the multi-tier web language LINKS [2]. We have shown that the original analysis was unsound. We have fixed the flaw and derived a correct analyser as an abstract semantics. In order to reconstruct this TES we have defined an abstract domain able to express types augmented by semantic annotations and concrete values. This abstract domain is an extension of Hindley's monotypes [7,4,3,9] and it could be easily implemented. Its definition is based on an approach described in [11] and exploits specific annotated types (*simple types*), where the annotations are replaced by a special kind of variables (*annotation variables*) whose values satisfy suitable constraints. An abstract value is indeed a pair (t_s, C) where t_s is a Hindley's monotype with annotation variables and C is a constraint whose solution represents the annotation.

We argue that this abstract domain is general enough to reconstruct different TESs by properly changing the set of monotypes and the shape of the constraints. To show the flexibility and the feasibility of our abstract domain we have reconstructed a quite simple analysis, *Call-Tracking Analysis (CTA)*, for which a TES was provided in [11]. This analysis allows one to determine, for each expression, the type of the computed value and the function applications which may occur during the evaluation. We defined this analysis, for a minimal ML core (TINYML). For example, the expression

```
let rec fact = fun[fact_point] n -> if (iszero n) then 1
                                    else  n * (fact (n - 1))
```

defines a recursive function which computes the factorial of a number and which is uniquely identified by the label `fact_point`. The CTA computes an abstract value expressing the annotated type $\mathtt{int} \xrightarrow{\{\mathtt{fact\_point}\}} \mathtt{int}$, i.e. a function from integers to integers which could apply the function denoted by the label `fact_point` during its evaluation.

To the best of our knowledge the only paper relating TESs and abstract interpretation is [12]. In this paper Vouillon and Jouvelot introduce a simple program time complexity estimator for a λ-calculus with recursion. They also define an abstract interpretation and a TES for this analysis. Their main result is that these two a priori distinct approaches are equivalent. Their abstract semantics computes for each expression the set of ground types compatible with the value given by the concrete evaluation. Hence, this abstract domain is more adapt to relate and prove equivalent two different approaches than to directly implement an analyser. Our abstract domain, instead, is designed to be easily implemented.

In this extended abstract we survey the ideas and the methodology underlying our abstract domain (Section 2) and we show some results obtained by the analyser, implemented in OCaml (Section 3). Page limitation prevent us from giving more details on the formal development and the implementation issues. A full presentation, including all the proof and the code can be found in [5].

2 Reconstruction of the Call-Tracking Analysis

In this section we describe the ideas of our reconstruction of the TES for CTA. TINYML is a minimal core of ML so its syntax is standard and we assume each λ-abstraction to have a unique label ($\mathtt{l} \in \mathtt{Point}$).

We define a denotational semantics as concrete semantics[3], that computes a pair for each expression: its value and the set of the function labels applied during the evaluation (the effects). To store the effects we use the *effects environment*. Since TINYML is an untyped λ-calculus, we define the semantic domain of values *Eval* as a recursive sum of cpos, where each element of the sum represents a suitable class of values.

As usual then we take the powerset of the concrete semantics as the collecting one. In a TES types can be annotated (e.g. to record latent effects), the definition of a suitable abstract domain requires particular care. In [6] we extend the Hindley's monotypes by introducing a new kind of variables (*annotation variable*) and constraints. In practice, an abstract value is a pair (t, C) where t is an Hindley's monotype with annotation variables and C is a set of constraints whose solution represent the annotation that the type can have. For the CTA the domain of abstract values is $TypeA = TypeS \times Constr$ where $TypeS$ is the set of Hindle's monotypes with annotation variables[1] (V_a) and $Constr = \mathcal{P}(V_a \times Point)$ is the set of constraints. A constraint is as a set of pairs $(annotation variable, label)$: $(\delta, \texttt{l})$ means that the label `l` is a member of the type annotation represented by the variable δ.

A Galois connection relates the abstract and the concrete domain. The connection is defined in [5] by using standard results, e.g. *representation function* [10].

The definition of the abstract semantics equations for CTA follows the same schema of [5,6], but in this case the computed effects concern the function applications encountered during the evaluation.

3 Examples

The abstract semantics of TinyML has been implemented in OCaml [8]. To illustrate the analyser, consider the expression

```
let a = fun[a_point] x -> true in
let b = fun[b_point] x -> false in
  (a 1) or (b 1)
```

defines two constant functions, `a` and `b`. The first function returns `true`, the second one `false`. We take the disjunction of applying both function to 1. The analyser computes

```
(type - : Boolean [(_annvar2_,a_point), (_annvar3_,b_point)] &
  {a_point, b_point})
```

that is the computed value is a boolean and during the evaluation we might apply both functions. Notice that this happens because during the abstraction process we loose precision. Since we do not know the values of the disjuncts, we have to evaluate them both. As a consequence, the resulting effect is not precise, yet safe and valid.

As second example consider the expression

```
(fun[x_point] x -> x) (fun[y_point] y -> y)
```

representing the application of the identity function with label `x_point` to the identity function with label `y_point`. The analyser computes

```
(type - : Function(_typevar1_, _annvar1_, _typevar1_)
     [(_annvar2_,x_point), (_annvar1_,y_point)]  & {x_point})
```

that is the computed type is a function and during the evaluation we might apply the function identified by the label `x_point`.

[1] Actually, for technical reasons *TypeS* is the lifting of Hindle's monotypes with idempotent substitutions and a new bottom, see [5,6]

4 Conclusions

We have shown that abstract interpretation can deal with TESs. We defined an abstract domain able to express types augmented by semantic annotations and at the same time simple enough to allow the implementation of an analyser in OCaml. Our abstract domain extends Hindley's monotypes with a new kind of variables and constraints. We have prove the expressive power of this domain by showing that it allows us to reconstruct different TESs by only changing the set of monotypes and the shape of constraints. As an example we have reconstructed the CTA.

References

1. Baltopoulos, I.G., Gordon, A.D.: Secure Compilation of a Multi-Tier Web Language. In: TLDI '09: Proceedings of the 4th international workshop on Types in language design and implementation. pp. 27–38. ACM, New York, NY, USA (2009)
2. Cooper, E., Lindley, S., Wadler, P., Yallop, J.: Links: Web programming without tiers. In: In 5th International Symposium on Formal Methods for Components and Objects (FMCO). Springer-Verlag (2006)
3. Cousot, P.: Types as abstract interpretations, invited paper. In: Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 316–331. ACM Press, New York, NY, Paris, France (Jan 1997)
4. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: POPL '82: Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 207–212. ACM, New York, NY, USA (1982)
5. Galletta, L.: Una semantica astratta per l'inferenza dei tipi ed effetti in un linguaggio multi-tier. Master's thesis, Università di Pisa (2010), available at `http://www.cli.di.unipi.it/~galletta/tesi.html`
6. Galletta, L., Levi, G.: An abstract semantics for inference of types and effects in a multi-tier web language. In: Proceedings of the 7th International Workshop on Automated Specification and Verification of Web Systems (2011)
7. Hindley, R.: The principal type-scheme of an object in combinatory logic. Transactions of the American Mathematical Society 146, 29–60 (1969)
8. INRIA: The Caml Language, `http://caml.inria.fr`, wWW publication
9. Monsuez, B.: Polymorphic typing by abstract interpretation. In: Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 217–228. Springer-Verlag, London, UK (1992)
10. Nielson, F., Nielson, H.R.: Type and Effect Systems. In: Olderog, E.R., Steffen, B. (eds.) Correct System Design, pp. 114–136. No. 1710 in Lecture Notes in Computer Science, Springer (1999), `http://www2.imm.dtu.dk/~nielson/Papers/NiNi99tes.pdf`
11. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, 1st ed. 1999. corr. 2nd printing, 1999 edn. (2005)
12. Vouillon, J., Jouvelot, P.: Type and effect systems via abstract interpretation (1995), `http://www.cri.ensmp.fr/classement/doc/A-273.pdf`

Leveraging dynamic typing through static typing

Paola Giannini[⋆2] and Daniele Mantovani[1] and Albert Shaqiri[12]

[1] AlgorithMedia, Alessandria, Italy
[2] CSI, DISIT, Univ. del Piemonte Orientale, Italy

Introduction Implementing more than a trivial application in JavaScript (or any other dynamically typed language) can cause problems due to the absence of type checking. Such problems can lead to unexpected application behaviour followed by onerous debugging. Although dynamic type checking and automatic type casting shorten the programming time, they introduce serious difficulties in the maintenance of medium to large applications. This is the reason why dynamically typed languages are rarely used for more than just prototyping and quick scripting.
We propose to deal with these problems using dynamically typed languages as "assembly languages" to which we translate the source code from `F#` which is statically typed. In this way, we take advantage of the `F#` type checker and type inference system, as well as other `F#` constructs and paradigms such as pattern matching, classes, discriminated unions, namespaces, etc. There are also the advantages of using an IDE such as Microsoft Visual Studio (code organization, debugging tools, IntelliSense, etc.).
To provide translation to different target languages we introduce an intermediate language. This is useful, for instance, for translating to Python that does not have complete support for functions as first class concept, or for translating to JavaScript, using or not libraries such as jQuery.
The paper is organized as follows. We first introduce the syntax of the core of the intermediate language. Then, we present the translation from `F#` to this intermediate language, and from the intermediate language to both JavaScript and Python. We do this via some examples that highlight the features of the intermediate language and the differences between the two target languages. Then, we briefly discuss correctness, and implementation. Finally, we compare our approach with related work, and discuss plans for future work.

Intermediate language The intermediate language is higher level than most intermediate languages. The syntax of the language is presented in Fig.1. There are three syntactic categories: *expressions*, e, *statements*, st, and *sequence statements*, s, which are sequences of statements returning a value. A *program* is a sequence statement. Although in `F#` everything is an expression, we introduce a distinction between expressions and statements as many target languages do. This facilitates the translation process and prevents some errors while building the intermediate abstract syntax tree, see [3] for a similar choice. The construct of the language which is most useful in the translation is `stm2exp`, which is a sequence statement, s, whose free mutable variables are a subset of $\{u_1, \ldots, u_n\}$;

⋆ The work of this author was partly funded by the project MIUR DISCO - Distribution, Interaction, Specification, Composition for Object Systems.

`stm2exp` is a value (like the lambda abstraction), and therefore, may be passed around. Its type, $\langle u_1{:}t_1, \ldots, u_n{:}t_n\rangle t$ says that in an environment in which the mutable variables u_i have type t_i $(1 \leq i \leq n)$, then s has type t. The construct `exc` e evaluates the expression e, which is supposed to be a `stm2exp` in the current store, dynamically binding its free mutable variables in the execution environment. The use of the construct will be explained when presenting the translation.

s	$::=$ `return` $e \mid st; s$	sequence statements
st	$::= u{:=}e \mid$ `let` $x{:}t{=}e \mid$ `let!` $u{:}t{=}e \mid$ `return` $e \mid$	
	$::=$ `if` e `then` s_1 `else` s_2	statements
e	$::= x \mid n \mid$ `tr` $\mid$ `fls` $\mid e_1{+}e_2 \mid$ `fun` $x{:}t$`->`$s \mid e_1\ e_2 \mid$ `exc` $e \mid$	
	$::=$ `stm2exp`$(s, \{u_1{:}t_1, \ldots, u_n{:}t_n\}) \mid$ `(int)`$e \mid$ `(bool)`e	expressions
t	$::=$ `int` $\mid$ `bool` $\mid t_1 \rightarrow t_2 \mid \langle u_1{:}t_1, \ldots, u_n{:}t_n\rangle t$	types
v	$::= n \mid$ `tr` $\mid$ `fls` $\mid$ `fun` $x{:}t$`->`$s \mid$ `stm2exp`$(s, \{u_1{:}t_1, \ldots, u_n{:}t_n\})$	values

Fig. 1. Syntax of core intermediate language

Translation by examples Many `F#` constructs can be directly mapped to JavaScript (or Python), but when this is not the case we obtain a semantically equivalent behaviour by using the primitives offered by the target language. E.g., in `F#` a sequence of expressions is itself an expression, while in JavaScript and Python it is a statement. Suppose we want to translate a piece of code that calculates a fibonacci number, binds the result to a name and also stores the information if the result is even or odd. On the left of Fig. 2 is the `F#` code. As we can see, on the right-hand-side side of `let x=` we have a sequence of

```
let mutable even = false
let x =
    let rec fib x =
        if x < 3 then 1
        else fib(x - 1) + fib(x - 2)
    let temp = fib 7
    even <- (temp % 2 = 0)
    temp
x
```

```
let y = stm2exp(
    let fib = fun x:int ->
          if x < 3 then return 1
          else return (fib (x-1)
                     + fib (x-2));
    let temp = fib 7;
    even := temp % 2 = 0;
    return temp;,
  {even:bool});
let! even = false;
let x = exc y
return x;
```

Fig. 2. Translation of `F#` sequence of expressions in the intermediate language

expressions: the definition of the function `fib` followed by the definition of `temp`, etc. This sequence is, in `F#`, an expression. On the right side of Fig. 2 is the translation into the intermediate code. The sequence of statements is translated in a `stm2exp` expression whose first component is the sequence of statements, and the second the set of free mutable variables occurring in such statements with their type: in this case the variable `even` of type `bool`, and bound to the variable `y`. The variable `x` is then bound to the `exc` expression applied to `y` (to obtain the result that we would have by evaluating the sequence of statements in the current environment). Assume that, the F# code was mapped to JavaScript literally, we would obtain the program on the left side of Fig. 3. This program is

```
var even = false;
var x =
    var fib = function (x) {
        if (x < 3)
            return 1;
        else
            return fib(x - 1) + fib(x - 2);
    };
    var temp = fib(7);
    even = (temp % 2) == 0;
    temp;
return x;
```

```
(function() {
  var even = false;
  var x = (function () {
      var fib = function (x) {
          if (x < 3)
              return 1;
          else
              return fib(x - 1)
                   + fib(x - 2);
      };
      var temp = fib(7);
      even = (temp % 2) == 0;
      return temp;
  })();
  return x;
})();
```

Fig. 3. Wrong and Correct JavaScript translations

syntactically wrong, since on the right-hand-side of an assignment we must have an expression, while a sequence of expressions is, in JavaScript, a statement. To transform a sequence of statements in an expression, in JavaScript, we wrap the sequence into a function, and to execute it we call the function, i.e., we use a JavaScript closures and application. Also, the whole program is wrapped into an entry point function. In this way, the code on the right side of Fig. 3 is correct. Unfortunately, the same cannot be done in Python as its support for closures is partial. So we have to define a temporary function, say `temp1`, in the global scope and to execute it we have to call `temp1` in the place where the original sequence should be. However, variables such as `even` will be out of the scope of their definition, and this would make the translation wrong. To obtain a behaviour semantically equivalent, we have to pass to `temp1` the variable `even`, by reference, since it may be modified in the body of `temp`. Note that, this problem is not present in JavaScript where the closure is defined and called in the scope of `even`. Another problem in Python is related to lambdas, whose body must be an expression (not a sequence). So we define the function `temp2` whose body contains the statements that should be placed where an expression is expected. In Fig. 4 we can see the translation of the F# code into Python. The class `ByRef` is used to wrap the mutable variable `even` to obtain a parameter called by reference. The Python code generator inserts the needed wrapping and unwrapping before and after the call of `temp1`, and in the body of `temp1`. Going back to our intermediate language, we use the construct `stm2exp` to provide the information needed to produce both translations, recording the information on the free mutable variables, needed for any language not supporting closures. We do not record free immutable variables, as they can be substituted with their values.

```
def temp1(even):
    def temp2(even, fib, x):
        if (x < 3):
            return 1
        else:
            return fib(x - 1) + fib(x - 2)

    fib = lambda x: temp2(even, fib, x)
    temp = fib(7)
    even.value = ((temp % 2) == 0)
    return temp
```

```
def __main__():
  even = false;
  wrapper1 = ByRef(even)
  x = temp1(wrapper1)
  even = wrapper1.value
  return x

__main__();
```

Fig. 4. Translation in Python

Dynamic Type checking JavaScript, and many dynamically typed languages, lack a rigorous type system. On the contrary, in `F#` if we write a function that adds two integers, see left side of Fig. 5, even though we do not specify type information, the interpreter infers the type shown after the function definition. Therefore, there is no way of calling `add` with arguments that are not of type integer. However, if our translation in the intermediate code would produce a function whose body was simply `x+y`, which in turn could be translated in the corresponding expression in both JavaScript and Python, the target JavaScript function could be called, e.g., `add("foo")(1)` and obtain the string `"foo1"` which is not what we wanted. In Python the situation would be better, in the sense that we cannot call `add` on a string and an integer, however, due to overloading we can call it on 2 floating points obtaining a floating point. To prevent

```
let add x y = x + y

val add : int -> int -> int
```

```
let add = fun x:int ->
    return fun y:int ->
        return (int)x+(int)y;
```

Fig. 5. `F#` code and the corresponding intermediate representation with type casting

this, the translation in the intermediate language, see right side of Fig. 5, insert type casting on the occurrences of function parameters. This is translated into dynamic type checking in JavaScript and Python as is shown in Fig. 6, where the function `toInt` tries to convert the argument we pass it to an integer, and if it fails, raises an exception. Our intermediate language supports other features

```
var add = function (x) {
    return function(y) {
      return toInt(x) + toInt(y);
    }
}
```

```
def temp1(x, y):
    return (int(x) + int(y))

def add(x):
    return lambda y: temp1(x, y)
```

Fig. 6. JavaScript and Python version with type casting

such as namespacing, classes, pattern matching, discriminated unions, etc. Some of this features have poor or no support at all in JavaScript or Python although semantically equivalent behaviour can be achieved through other language constructs.

Full abstraction of the translations We have defined an operational semantics for the intermediate language, IL, and a type system enforcing the property that well typed programs evaluated in a store that agrees with their definition environment do not get stuck. Moreover, before the conference we plan to prove the full abstraction of the translations. That is, (1) formalize a fragment of F#, $\texttt{FS}^c$, a core Javascript, $\texttt{JS}^c$, and a core Python, $\texttt{PY}^c$, as we have done for the intermediate language; (2) define the translations from $\texttt{FS}^c$ to IL, and from IL to $\texttt{JS}^c$ and $\texttt{PY}^c$; (3) prove that the translations preserve the operational semantics (and for the one from $\texttt{FS}^c$ to IL also the typing) of the relative languages.

Implementation The compiler is implemented in F# and is based on two metaprogramming features offered by the .net platform: *quotations* and *reflection.* These mechanisms allow one to extract code and type information during runtime, reason about it and, in our case, are used to build an intermediate language abstract syntax tree from which the target code is generated.

Comparisons and future work Similar projects exist and are based on similar translation techniques, although, as far as we know, we are the first to introduce an intermediate language allowing to translate to many target languages. Pit, see [4], is an open source F# to JavaScript compiler. It supports many F# features (at the time of this writing it is at version 0.2) and is very well documented. It supports only translation to JavaScript. Websharper, see [5], is a professional web and mobile development framework. As of version 2.4 an open source license is available. It is a very rich framework offering extensions for ExtJs, jQuery, Google Maps, WebGL and many more. Again it supports only JavaScript. F# Web Tools is an open source tool whose main objective is not the translation to JavaScript, instead, it is trying to solve the difficulties of web programming: "the heterogeneous nature of execution, the discontinuity between client and server parts of execution and the lack of type-checked execution on the client side", see [8]. It does so by using meta-programming and monadic syntax. One of it features is translation to JavaScript. Finally, a translation between Ocaml byte code and JavaScript is provided by Ocsigen, and described in [9].
On the theoretical side, a framework integrating of statically and dynamically typed (functional) languages is presented in [6]. In [10] a cast construct wrapping dynamic code is introduced, and it is showed how it can be used to prove the source of run time type errors. Support for dynamic languages is provided with ad hoc constructs in Scala, see [7]. Finally, a construct similar to `stm2exp`, is studied in [2], where it is shown how to use it to realize dynamic binding and meta-programming, an issue we are planning to address.
Our future work will be on the practical side to use the intermediate language to integrate F# code and JavaScript or Python native code. On the theoretical side, we plan to finish the proof of full abstraction of the translation from F# to the intermediate language, and from this to the target languages. Moreover, we would like to explore meta-programming on the line of [2]. We also plan to explore

the extension to polymorphic types of the type system for the intermediate language, which is, as shown in [1] non trivial.

References

1. Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. Blame for all. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA*, ACM, pages 201–214, 2011.
2. Davide Ancona and Paola Giannini Elena Zucca. Reconciling positional and nominal binding. In *ITRS 2012 (accepted for presentation)*, 2012.
3. Andrew W. Appel. *Modern Compiler Implementation in ML.* Cambridge University Press, 1998.
4. Mohamed Suhaib Fahad. Pit - F Sharp to JS compiler. `http://pitfw.org/`, May 2012.
5. Intellifactory. Websharper 2010 platform. `http://websharper.com/`, May 2012.
6. Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *ACM Trans. Program. Lang. Syst.*, 31(3), 2009.
7. Adriaan Moors, Tiark Rompf, Philipp Haller, and Martin Odersky. Scala-virtualized. In Oleg Kiselyov and Simon Thompson, editors, *Proceedings of the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation, PEPM 2012, Philadelphia, Pennsylvania, USA*, ACM, pages 117–120, 2012.
8. Tomáš Petříček and Don Syme. AFAX: Rich client/server web applications in `F#`. `http://www.scribd.com/doc/54421045/Web-Apps-in-F-Sharp`, May 2012.
9. Jerome Vouillon and Vincent Balat. From bytecode to javascript: the js of ocaml compiler. `http://www.pps.univ-paris-diderot.fr/~balat/publi.php`.
10. Philip Wadler and Robert Bruce Findler. Well-typed programs can't be blamed. In *ESOP 2009*, volume 5502 of *LNCS*, pages 1–16, 2009.

Lock Analysis for an Asynchronous Object Calculus

Elena Giachino and Tudor A. Lascu

FOCUS Research Team INRIA / Dipartimento di Scienze dell'Informazione, Università di Bologna
[giachino | lascu]@cs.unibo.it

1 Introduction

The model. The problem we are interested in is discovering deadlocks and livelocks in an object-oriented setting, where method calls are asynchronous, meaning that after a method is invoked the caller does not wait for the returned value, instead it continues its activity until the result is strictly necessary. In this model [4, 3] objects have multiple tasks in execution, spawned by method invocations, and there is at most one active task per object at each point in time. The active task may explicitly return the control in order to let another task of the same object progress. The decoupling of method invocation and returned value is realized by means of *future variables*, which are pointers to values that may be not available yet. Clearly, the access to values of future variables may require waiting for the value to be returned. In order to program in this model we introduce a language called *Featherweight Java with futures* (`FJf`) [1], inspired by `FJ` [2], that features two primitives for dealing with *future*s and control release. The `get` operation is used to retrieve a return value, keeping object's lock while waiting for it. The `await` operation, instead, is used to wait for the availability of a computation's result. If the result is not ready, the task suspends, by first releasing the lock of the object. If the result is ready, `await` is non-blocking but in order to retrieve the actual value it waited for, the task still needs to perform a `get`.

Deadlocks. In the considered model a typical deadlock occurs when one or more tasks are waiting for each other's termination to return a value. Let us consider a simple scenario with two objects `a` and `b` as in Fig. 1. Object `a` contains a task related to some method `m` which in turn invokes a method `n` on object `b` and explicitly blocks waiting for the result. This invocation triggers a task at `b` responsible of executing `n`'s body, which again performs an invocation on `a` waiting for the result. This last task inside `a` is created but it never obtains the lock (held by `m`) for executing its own code. The computation is deadlocked. Black arrows between objects correspond to *object dependencies*,

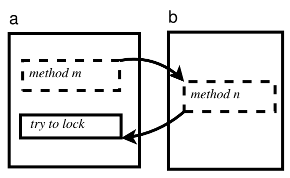


Fig. 1. A simple deadlock.

```
class C {
    C m() { return new C() ;}
    C r(C x) { return x!m().get ;}
}
class D extends C {
    Fut(C) q(D y) {
      return (y!r(this); this!r(y));
    }
}
```

Fig. 2. Simple classes in `FJf`

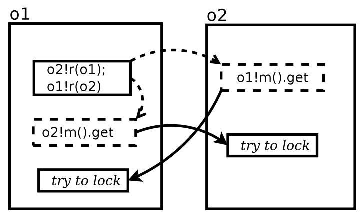


Fig. 3. Deadlock in `FJf`

introduced by `get` operations. A circular dependency means that a deadlock is encountered.

Livelocks. Let us consider a slightly different example, where method `n` at `b` performs an `await`. Thus, while waiting it releases the lock instead of keeping it. The semantics of `await` requires the task to compete again for the lock with other tasks in the same object, and then try again for the result. If it is available the computation proceeds, otherwise the lock is released again and so on. Releasing the lock in `b` does not change the fact that tasks in `a` are blocked. The circular dependency still holds, however the system is not completely blocked but the task of method `n` is caught in an infinite loop of getting and releasing the lock (a *livelock*).

A non-trivial dangerous pattern. We now discuss a program that can manifest a subtle deadlock. In `FJf` a program is a collection of class definitions plus an expression to evaluate, just as in `FJ`. A simple definition in `FJf` is the class `C` in Fig. 2 that defines two methods: `m` to build a new object and `r` that we discuss below. Class `D` extends `C` with a method, `q`, which returns a value of type `Fut(C)`, the type of a method invocation returning an object of type `C`. Let's consider the expression `new D()!q(new D())`, i.e. an asynchronous call of method `q`. Two objects, o_1 and o_2, of class `D` are created and then the method `q` is invoked on o_1 passing o_2 as a parameter. The body of `q` contains a sequence of two invocations of `r`. Each invocation spawns a new task and returns immediately the control to the caller. These two tasks are hence spawned, inside o_1 and o_2, respectively (see Fig. 3 where the method invocations are indicated by the dotted lines). The task executing `q` terminates correctly. The program, however, continues its execution with the spawned tasks. Among the possible interleavings there is one that gives a deadlock. In particular the dotted task in o_2 invokes `m` on o_1, where a new task is created, and waits for the result returned by a task inside o_1, keeping the lock of o_2 (since it's a `get` operation). Analogously, the dotted task in o_1 gets to run and it hangs waiting for a method in o_2 to return, holding o_1's lock. The two tasks shown with a plain line will never be able to execute since the lock of each object is kept by the dotted tasks. Moreover the two objects o_1 and o_2 are both indefinitely blocked and hence the program is stuck. Notice that this deadlock depends on the scheduler: had the task in o_1, on which the dotted task in o_2 depends, gotten into execution before the dotted task in o_1 no problem would have arisen.

Contracts. In order to capture circular dependencies and therefore statically detect dangerous configurations as the ones described above, we need to trace all the object dependencies, in the form of pairs of object names (graphically represented by the arrows in Fig. 1). Since we want to do this statically, we need a way to identify and track every object in the program. To this aim, we define a technique that associates to each `new` operation (i.e. the one that is responsible of creating a new object) a fresh object name, picked from a countable set of object names $o_1, o_2, \ldots$. Then a type system computes the object name associated to each expression of the program. For instance, to a method invocation is assigned (a reference to) the object name returned by the method, to a field selection the object stored in the field, and so on. The type entity conceived to represent this kind of information is the *future record.* Moreover, as we said, we are interested only in detecting object pairs, which are the result of `get` and `await` operations, and we collect those pairs by following all the chains of method invocations. Therefore other local computation terms different from `get`, `await`, and method invocations are not relevant to the analysis. The type system, besides computing object identities of expressions, is responsible of extracting from the program abstract descriptions, called *contracts*, containing only relevant information. More precisely, the typing judgments have the form $\Gamma \vdash_a \mathtt{e} : (\mathtt{T}, \mathbb{r})\,,\ \mathbb{c}$, where Γ is the environment, a is the name of the object `this`, `e` is a `FJf` expression, `T` is its (class or future) type, $\mathbb{r}$ is a future record, and the $\mathbb{c}$ is the contract. A future record of the form $a\mathtt{[}\bar{\mathtt{f}} : \bar{b}\mathtt{]}$, when associated to an expression `e`, means that a is the object associated to `e`, and the fields of a are assigned the objects $\bar{b}$. A future record of the form $a \rightsquigarrow \mathbb{r}$ corresponds to a future reference to the object $\mathbb{r}$, being produced by a method invoked on an object a. (In order to retrieve the actual value, a `get` operation must be perfomed on the expression assigned this record.) Future records can also be not fully specified, as $c\mathtt{[f} : X\mathtt{]}$, allowing us to avoid infinite future records. The syntax of contracts associated to expressions is the following:

$$\begin{array}{l} \mathbb{c} ::= \mathtt{0} \ \mid\ \mathtt{C.m}\, \mathbb{r}(\bar{\mathbb{r}}) \to \mathbb{r}' \ \mid\ \mathtt{C.m}\, \mathbb{r}(\bar{\mathbb{r}}) \to \mathbb{r}' \centerdot (a, a') \ \mid\ \mathtt{C.m}\, \mathbb{r}(\bar{\mathbb{r}}) \to \mathbb{r}' \centerdot (a, a')^{\mathtt{a}} \ \mid \\ \qquad (a, a') \ \mid\ (a, a')^{\mathtt{a}} \ \mid\ \mathbb{c} \fatsemi \mathbb{c} \end{array}$$

where $\mathtt{C.m}\ \mathbb{r}(\bar{\mathbb{r}}) \to \mathbb{r}'$ corresponds to the invocation of `m` in class `C` on a object $\mathbb{r}$, passing objects $\bar{\mathbb{r}}$ as parameters, and object $\mathbb{r}'$ is the returned value. Object pairs (a, a') and $(a, a')^{\mathtt{a}}$ are introduced by `get` and `await` operation, respectively. They can be isolated, meaning the operation has been performed on a method parameter or field, or they can be associated to a method invocation. Finally, $\mathbb{c} \fatsemi \mathbb{c}$ is used for sequential composition.
The behavior of a method is described by the contract of its body, which is wrapped around by an interface specifying the binders, i.e. the receiver's and parameters' names, for the names occurring inside the contract, and the returned value. All free occurrences correspond to new objects, and are therefore fresh names in the system. This whole description, called *method contract*, has the form $\mathbb{r}(\bar{\mathbb{r}})\{\mathbb{c}\}\ \mathbb{s}$. The contract of a method call is built by instantiating the formal object names, contained in the method contract, with the actual ones. For example, the contract of the method `m` of Fig. 2 is derived using the rule

$$\frac{\varGamma + \texttt{this} : (\texttt{C}, a\texttt{[]}) \vdash_a \texttt{new C()} \ : \ (\texttt{C}, b\texttt{[]})\,,\ \texttt{0}}{\varGamma \vdash \ \texttt{C m ()\{return new C();\}} : a\texttt{[]()\{0\}}\ b\texttt{[]} \ \ \textsc{in}\ \texttt{C}}$$

where in $a\texttt{[]()\{0\}}\ b\texttt{[]}$ we have $a\texttt{[]}$ as the receiver object (a is the object name and `[]` means an empty sequence of fields), no parameters, and $b\texttt{[]}$ as the returned object. The contract is empty ($=$ `0`) in this case, meaning that `m`'s body contains no method invocations. Let's consider method `r`'s contract: $a\texttt{[]}(c\texttt{[]})\{\,\texttt{C.m}\ \ c\texttt{[]()} \ \rightarrow\ b\texttt{[]}.(a, c)\,\}\ b\texttt{[]}$, where the contract body gives an abstract account of `r`'s behavior: it invokes method `m` of the same class on the object $c\texttt{[]}$, passed to it as a parameter, and finally returns a third object $b\texttt{[]}$. The `get` operation specifies that the task inside a has to wait for another task inside c to return: this information is thus added to contract $\mathbb{c}$ with the pair (a, c). The general rules for `get` and `await` expressions are:

$$\frac{\varGamma \vdash_a \texttt{e} \ : \ (\texttt{Fut(T)}, a' \rightsquigarrow \mathbb{s})\,,\ \mathbb{c}}{\varGamma \vdash_a \texttt{e.get} \ : \ (\texttt{T}, \mathbb{s})\,,\ \mathbb{c} \between (a, a')} \qquad \frac{\varGamma \vdash_a \texttt{e} \ : \ (\texttt{Fut(T)}, a' \rightsquigarrow \mathbb{s})\,,\ \mathbb{c}}{\varGamma \vdash_a \texttt{e.await} \ : \ (\texttt{Fut(T)}, a' \rightsquigarrow \mathbb{s})\,,\ \mathbb{c} \between (a, a')^{\texttt{a}}}$$

where the $\between$ operator can be read as "add pair (a, a') to $\mathbb{c}$". As expected, the `get` operation, retrieving the result, on an expression of type `Fut(T)` returns an object of type `T` and the returned future record $\mathbb{s}$. The `await` operation instead, dealing only with the availability of the result, leaves the type and record part unchanged. They both affect the contract $\mathbb{c}$ by documenting that in the former (resp. the last) case, the continuation (resp. correct termination) of the execution of object a's activity is bound to the termination of a task inside an object a'.

The analysis. Once contracts have been inferred we transform them into automata and by composition we obtain a finite model of an `FJf` program. The states of this automaton contain dependencies and the transitions model how dependencies are activated or discarded during program's execution. A potential misbehavior is signaled by the presence of a circularity in some state of it. Our analysis is based on over-approximations and as such: if the analysis certifies a program to be lock-free it certainly is, otherwise it means that a locked configuration *might* be reached at run-time (as is the case of deadlock depending on the scheduler's choices). The framework we have just described can be found in [1]. In the following section we discuss some extensions and their impact on the analysis.

2 Extensions

Field updates. `FJf`, just as `FJ`, is a functional language as fields are initialized by the constructor and are immutable. In this contribution we introduce a notion of state by enabling the private field updates, namely by adding `this.f = e` to the syntax of expressions. To see how this enhancement is reflected upon our framework let us consider a sequence of two method invocations `x.m(); x.n()`, both called on the same object and both modifying the same field. Say `m` does `this.f = e1` and `n` does `this.f = e2`. Due to the asynchronous nature of method invocation, there is no way to know the order in which the updates will take place at run-time. Working statically, we have to keep track of all the different possibilities, thus the type system must assume for an expression a set of possible objects it can reduce to. The updated syntax of future records is the following: $\mathbb{r} ::= X \mid \mathcal{A}[\bar{\mathtt{f}} : \bar{\mathbb{r}}] \mid \mathcal{A} \rightsquigarrow \mathbb{r}$, where $\mathcal{A}$ is a set of object names. For instance, let us consider a field `f` containing an object with a field `g`. If two different updates of `f` occur in the program, with two expressions of future record $\mathbb{r} = b[\mathtt{g} : \mathbb{r}']$ and $\mathbb{s} = c[\mathtt{g} : \mathbb{s}']$, respectively. The future record of `f` must then take into account both updates and therefore it will be $\mathbb{r} \bigvee \mathbb{s} = \{b, c\}[\mathtt{g} : \mathbb{r}' \bigvee \mathbb{s}']$. A typing rule for the new construct is introduced:

$$\frac{\Gamma \vdash_a \mathtt{this} : (\mathtt{C}, a[\bar{\mathtt{f}} : \bar{\mathbb{r}}', \mathtt{f} : \mathbb{r} \bigvee \mathbb{s}) \,,\, 0 \qquad \Gamma \vdash_a \mathtt{e} : (\mathtt{C}', \mathbb{r}) \,,\, \mathbb{c} \qquad \mathtt{C}' <: \mathtt{C}}{\Gamma \vdash_a \mathtt{this.f = e} : (\mathtt{C}', \mathbb{r}) \,,\, \mathbb{c}}$$

As for the analysis, the transformation of contracts into automata must be adapted to treat sets of names instead of single object names. While the analysis is less precise, since it predicts a set of dependencies for each actual one, it is still correct: if a program is recognized to be lock-free, its execution will proceed without encountering a locked configuration.

Task Dependencies. By extending `FJf` with field update we introduce the possibility of having *pure livelocks*: configurations made up of only `await`-generated dependencies in which there is a circularity of task dependencies. Fig. 5 depicts such a configuration in which `await`-pairs are shown with a dashed arc. Without field updates, it is not possible to write a program that leads to such a configuration. Therefore, the analysis could safely ignore an `await`-circularity between objects (as in [1]). Consider for example the situation of Fig. 4.

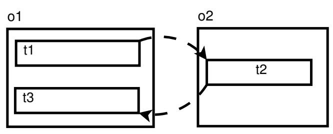


Fig. 4. Non-problematic `await`-circularity

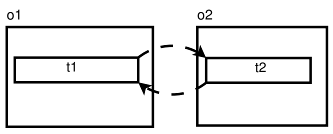


Fig. 5. Pure livelock

The (object) circularity due to $(o1, o2)^{\mathsf{a}}$ and $(o2, o1)^{\mathsf{a}}$ does not lead to a livelock since task `t3` can acquire $o1$'s lock and get into execution. `t2` and `t1` can therefore terminate. As soon as pure livelocks become expressible, we have to refine the analysis to work at a finer granularity by also taking into consideration task pairs. In facts, when the analysis finds a circularity of `await`-pairs, it needs additional information on task pairs in order to discriminate pure livelocks from non-dangerous configurations. (This additional discriminating power only concerns `await`-circularities, as a circularity with one or more `get` is always a dangerous configuration.) The typing judgments must then include information about the task in which a given expression is to be typed. They thus take the form: $\Gamma \vdash_a^{\mathtt{t}} \mathtt{e} : (\mathtt{T}, \mathbb{r}) \,,\, \mathbb{c}$.

We adopt a technique similar to the one for object names described in Sec. 1. Namely, we introduce a fresh task name (picked from a countable set of task names `t1`, `t2`,...) for every method invocation much in the same way as we introduced a fresh object name for every `new` construct in the program. That is, we employ task names for tagging a method invocation with the task responsible for its computation: $\mathcal{A} \rightsquigarrow_{\mathtt{t}} \mathbb{S}$. The rule for `await` becomes:

$$\frac{\Gamma \vdash^{\mathtt{t}}_{a} \mathtt{e} : (\mathtt{Fut(T)}, \mathcal{A} \rightsquigarrow_{\mathtt{t'}} \mathbb{S}) \;,\; \mathbb{C}}{\Gamma \vdash^{\mathtt{t}}_{a} \mathtt{e.await} : (\mathtt{Fut(T)}, \mathcal{A} \rightsquigarrow_{\mathtt{t'}} \mathbb{S}) \;,\; \mathbb{C} \between (\mathtt{t}, \mathtt{t'}) \between (a, a_i)^{\mathsf{a}} \, \forall a_i \in \mathcal{A} \rightsquigarrow_{\mathtt{t'}} \mathbb{S}}$$

adding both a task pair $(\mathtt{t}, \mathtt{t'})$ and a set of object pairs $(a, a_i)^{\mathsf{a}}$, one for each a_i in the set $\mathcal{A}$.

References

1. E. Giachino, C. Laneve, and T. A. Lascu. Deadlock and livelock analysis in concurrent objects with futures. `www.cs.unibo.it/~laneve/publications.html`, 2011. Submitted.
2. A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.*, 23:396–450, 2001.
3. E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In *Proc. of FMCO 2010*, volume 6957 of *LNCS*, pages 142–164. Springer-Verlag, 2011.
4. E. B. Johnsen and O. Owe. An asynchronous communication model for distributed concurrent objects. *Software and System Modeling*, 6(1):39–58, 2007.

Input/Output Types for Dynamic Web Data (Extended Abstract)

Svetlana Jakšić

Faculty of Technical Sciences
University of Novi Sad
sjaksic@uns.ac.rs

As information networks become more open and dynamic, the need for protecting security and privacy of data is increasingly important in many fields of human activities. Systems must be able to exchange data and processes while preserving security. In case we are given a target security policy for a distributed system containing XML data, how can we check weather the system behaves according to the policy? One solution is to suitably annotate the security relevant events, to classify them according to a type system and to verify security properties by typing. In [3] we introduced ®X$d\pi$-calculus, a calculus for role-based access control of dynamic web data, and a type system for it and we verify the security properties by typing. Here, a subtyping relation which extends the type system for ®X$d\pi$ is proposed.

First, an example which illustrates the proposed approach and advantages of the extended type system will be given and then a brief presentation of the subtyping relation and a discussion of the related work. The full presentation of ®X$d\pi$-calculus and the corresponding type system, which can be found [3], will be omitted here.

An example

Let us consider a simple distributed system consisting of code running on behalf of four principals: an online discussion forum, a guest, a member and the moderator. Let the forum, written in XML notation, have the shape:

$$\begin{array}{l} <\texttt{forum}> \\ \quad <\texttt{general}> \\ \qquad \mathit{forum\ rules} \\ \quad </\texttt{general}> \\ \quad <\texttt{music}> \\ \qquad \mathit{lyrics} \\ \quad </\texttt{music}> \\ </\texttt{forum}> \end{array}$$

The forum contains two main topics: the `general` and the `music`. In order to describe the behaviour of the guest, the member and the moderator we will use process calculus notation. In the syntax of ®X$d\pi$-calculus we consider four kinds of processes: π-calculus processes [7], for modelling local communication; go command, for modelling process migration between locations, as in $D\pi$ calculus [6]; `run`, `read` and `change` commands and for modelling interaction of processes with local data in place of the update command of [5] and commands enable and disable for changing permissions to access data. We write $\texttt{read}_{\texttt{forum/general}}(\chi)$ for a guest wishing to read forum rules, where χ is a data pattern he is looking for, and $\texttt{read}_{\texttt{forum/music}}(\chi)$ for a member wishing to read the lyrics. The process $!b(y).\texttt{change}_{\texttt{forum}}(x, x|y)$ represents the moderator of the forum who has the ability to add a new topic. The moderator receives the new topic, updates the forum with it and waits to receive the next topic. In [3] we have investigated a system in which different participants can have different rights. We have achieved diversity and control of the rights by introducing role-based access control. More precisely, each tag is assigned a set of roles that a process is required to have in order to access it. The forum of this example decorated with sets of roles is:

$$
\begin{array}{l}
< \texttt{forum role} = \texttt{guest} > \\
\quad < \texttt{general role} = \texttt{guest} > \\
\quad\quad \mathit{forum\ rules} \\
\quad < /\texttt{general} > \\
\quad < \texttt{music role} = \texttt{member} > \\
\quad\quad \mathit{lyrics} \\
\quad < /\texttt{music} > \\
< /\texttt{forum} > .
\end{array}
$$

The same forum written in the syntax of our calculus is:

$$\texttt{forum}^{\{\texttt{guest}\}}[\texttt{general}^{\{\texttt{guest}\}}[\mathit{forum\ rules}]|[\texttt{music}^{\{\texttt{member}\}}[\mathit{lyrics}]].$$

Let the roles guest, member and moderator belong to a countable set of roles which is a lattice for a partial order $\sqsubseteq$. We consider $\texttt{guest} \sqsubseteq \texttt{member} \sqsubseteq \texttt{moderator}$. As expected, we assign the role guest to the unregistered guest of the forum, the role member to the registered user and the role moderator to the moderator of the forum. We say that the tag (or the edge when we use tree representation of XML documents) forum is accessible to the process with role guest or higher. The path forum/general is accessible to the process with the role guest since both tags are accessible to it, while the path forum/music is not. The forum we have described here is a "wiki" forum that allows guests and members to add content to the parts they have access to, as on an Internet forum, but also allows them to edit the content. All processes belonging to the same role have the same rights. Locations contain the processes and data. The behaviour of all the principals in the system is controlled with location policies and type system introduced in [3]. The policy of a location regulates changes of access rights. For example, if the forums' location policy is $(\{\texttt{guest}\}, \{(\{\texttt{moderator}\}, \texttt{guest})\}, \{(\{\texttt{moderator}\}, \texttt{member})\})$ then the processes with a role lower then guest can not access the forum at all and that the moderator may allow guest to access more topics or ban members to access the some topics in the forum. The dynamic change of access rights to data is done by adding or removing roles from the sets of roles on the data tree edges. The type system checks if a data tree and a process conform to a given location policy.

We propose an extension of the type system of [3] with subtyping relation in order to describe richer behaviour in our model. In the forum example, the process

$$
\begin{array}{l}
\bar{b}\langle \texttt{new}^{\{\texttt{guest}\}}[\cdots]\rangle^{\neg\{\texttt{guest}\}} \mid \bar{b}\langle \texttt{new}^{\{\texttt{member}\}}[\cdots]\rangle^{\neg\{\texttt{member}\}} \\
\quad \mid\ !b(y).\texttt{change}_{\texttt{forum}^{\{\texttt{moderator}\}}}(x, x|y)^{\neg\{\texttt{moderator}\}}
\end{array}
$$

which represents a guest and a member, both wishing to send a new topic to the moderator for approval, is rejected by the type system of [3]. However, with the proposed subtyping relation, this process is typable.

Subtyping relation

We assume a countable set of roles $\mathcal{R}$, and use r to range over elements of $\mathcal{R}$. Let $(\mathcal{R}, \sqsubseteq)$ be a lattice and let $\bot, \top \in \mathcal{R}$ be its bottom and top element, respectively. By α, ρ, σ we denote non-empty sets of roles and by τ, ζ sets of roles containing the $\top$ element. We introduce a pre-order relation on value types in order to expand a domain of values that channels can communicate. With this aim, we have enriched the set of value types from [3] with the type of channels emitting values and with the type of channels receiving values as in [8, 9]. The types are presented in Table 1. Tv ranges over *value types* where as a value we consider either a channel name, a script, a location name, a path or a tree.

Table 1. The Syntax of ®X$d\pi$ Types

$Ch(Tv)$	type of channels communicating values of type Tv
$Ch^!(Tv)$	type of channels emitting values of type Tv
$Ch^?(Tv)$	type of channels receiving values of type Tv
$Loc(\mathcal{P})$	type of locations with the policy $\mathcal{P}$
$Script(\mathcal{P})$	type of scripts which can be activated at locations with the policy $\mathcal{P}$
$Path(\alpha)$	type of paths having the last edge with the set of roles α
$Pointer(\alpha)$	type of pointers whose path is typed by $Path(\alpha)$
$Tree(\mathcal{P}, \tau, \zeta)$	type of trees, which can stay at locations with the policy $\mathcal{P}$, with initial branches asking τ and which can be completely accessed by processes with at least one role of ζ
$Proc(\mathcal{P}, \rho)$	type of pure processes, which can stay at locations with the policy $\mathcal{P}$ and which can be assigned roles ρ
$ProcRole(\mathcal{P})$	type of processes with roles which can stay at locations with the policy $\mathcal{P}$

The subtyping rule

$$\frac{\Gamma \vdash v : Tv_1 \quad Tv_1 \prec Tv_2}{\Gamma \vdash v : Tv_2}$$

states that if Tv_1 is subtype of Tv_2, then a value of type Tv_1 is also of type Tv_2. The subtyping relation is such that if a *channel* can communicate values of type Tv then it can do both, emit and receive the values. Any channel that is receiving values of some type can be regarded as a channel receiving higher values. Any channel that is emitting values of some type can be regarded as a channel emitting lower values. The type of a *location* with the policy $\mathcal{P}_1$ is lower then the type of a location with policy $\mathcal{P}_2$ if $\mathcal{P}_2$ is less restrictive then $\mathcal{P}_1$. If a *script* can be activated at a location then it can also be activated at any bigger location. A *path* having the last edge with a set of roles α_1 can be regraded as having at last edge any set of bigger or equal roles to those from α_1. Suppose that we are given a *tree* that can stay at a location of some policy $\mathcal{P}$, with initial branches asking τ_1 and that can be completely accessed by a process with at least one role from ζ_1. We can say that its initial branches are also asking set of roles that are greater than or equal to ζ_1 and it can be completely accessed by processes with roles greater or equal to ζ_1.

By extending the proof from [3], we can prove that the system satisfies the subject reduction and other relevant properties of well behaved processes. We proved in [3] that processes can communicate only values with at least one characteristic role lower than equal to a role of the process. The subtyping relation implies that channels emitting a value of type can also emit all the values that are of smaller type with respect to the relation. The channels receiving values of a type can also receive values that have bigger types.

Conclusions and related work

In this paper a notion of subtyping is added to the type system of the ®X$d\pi$-calculus and it is demonstrated that subtyping increases the flexibility of types. The type systems and calculi discussed here strongly relies on [3] and is most related to [4, 1] and [2]. Input and output types are those from [8, 9]

Acknowledgment. This work has been supported by the Serbian Ministry of Education and Science (projects ON174026 and III44006) and Provincial Secretariat for Science and Technological Development of Province of Vojvodina. ICTCS reviewers have provided useful comments.

References

1. Chiara Braghin, Daniele Gorla, and Vladimiro Sassone. Role-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155, 2006.
2. Adriana B. Compagnoni, Elsa L. Gunter, and Philippe Bidinger. Role-based access control for boxed ambients. *Theoretical Computer Science*, 398(1-3):203–216, 2008.
3. Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jakšic, and Jovanka Pantovic. Types for Role-Based Access Control of Dynamic Web Data. In *WFLP'10*, volume 6559 of *LNCS*, pages 1–29. Springer, 2011.
4. Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Jovanka Pantovic, and Daniele Varacca. Security types for dynamic web data. *Theoretical Computer Science*, 402(2-3):156–171, 2008.
5. Philippa Gardner and Sergio Maffeis. Modelling dynamic web data. *Theoretical Computer Science*, 342(1):104–131, 2005.
6. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
7. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
8. Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
9. Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

Converting Nondeterministic Automata and Context-Free Grammars into Parikh Equivalent Deterministic Automata[*]

(Extended Abstract)

Giovanna J. Lavado[1], Giovanni Pighizzini[1], and Shinnosuke Seki[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano
via Comelico 39, I-20135, Milano, Italy
giovanna.lavado@unimi.it
giovanni.pighizzini@unimi.it
[2] Department of Information and Computer Science, Aalto University,
P.O. Box 15400, FI-00076, Aalto, Finland
shinnosuke.seki@aalto.fi

Abstract. We investigate the conversion of nondeterministic finite automata and context-free grammars into Parikh equivalent deterministic finite automata, from a descriptional complexity point of view.
We prove that for each nondeterministic automaton with n states there exists a Parikh equivalent deterministic automaton with $e^{O(\sqrt{n \cdot \ln n})}$ states. Furthermore, this cost is tight. In contrast, if all the strings accepted by the given automaton contain at least two different letters, then a Parikh equivalent deterministic automaton with a polynomial number of states can be found.
Concerning context-free grammars, we prove that for each grammar in Chomsky normal form with n variables there exists a Parikh equivalent deterministic automaton with $2^{O(n^2)}$ states. Even this bound is tight.

1 Introduction

It is well-known that the state cost of the conversion of nondeterministic finite automata (NFAs) into equivalent deterministic finite automata (DFAs) is exponential: using the classical subset construction [10], from each n-state NFA we can build an equivalent DFA with 2^n states. Furthermore, this cost cannot be reduced.

In all examples witnessing such a state gap (e.g., [5–7]), input alphabets with at least two letters and proof arguments strongly relying on the structure of strings are used. As a matter of fact, for the unary case, namely the case of the one letter input alphabet, the cost reduces to $e^{\Theta(\sqrt{n \cdot \ln n})}$, as shown by Chrobak [1].

[*] Paper accepted at the *16th International Conference on Developments in Language Theory*. In H.–C. Yen and O.H. Ibarra (Eds.): DLT 2012, LNCS 7410, pp. 284–295. Springer (2012)

What happens if we do not care of the order of symbols in the strings, i.e., if we are interested only in obtaining a DFA accepting a set of strings which are equal, after permuting the symbols, to the strings accepted by the given NFA?

This question is related to the well-known notions of Parikh image and Parikh equivalence [8]. Two strings over a same alphabet Σ are Parikh equivalent if and only if they are equal up to a permutation of their symbols or, equivalently, for each letter $a \in \Sigma$ the number of occurrences of a in the two strings is the same. This notion extends in a natural way to languages (two languages L_1 and L_2 are Parikh equivalent when for each string in L_1 there is a Parikh equivalent string in L_2 and vice versa) and to formal systems which are used to specify languages as, for instance, grammars and automata. Notice that in the unary case Parikh equivalence is just the standard equivalence. So, in the unary case, the answer to our previous question is given by the above mentioned result by Chrobak.

Our first contribution in this paper is an answer to that question in the general case. In particular, we prove that the state cost of the conversion of n-state NFAs into Parikh equivalent DFAs is the same as in the unary case, i.e., it is $e^{\Theta(\sqrt{n \cdot \ln n})}$. More surprisingly, we prove that this is due to the unary parts of languages. In fact, we show that if the given NFA accepts only nonunary strings, i.e., each accepted string contains at least two different letters, then we can obtain a Parikh equivalent DFA with a polynomial number of states in n. Hence, while in standard determinization the most difficult part (with respect to the state complexity) is the nonunary one, in the "Parikh determinization" this part becomes easy and the most complex part is the unary one.

In the second part of the paper we consider context-free grammars (CFGs). Parikh Theorem [8] states that each context-free language is Parikh equivalent to a regular language. We study this equivalence from a descriptional complexity point of view. Recently, Esparza, Ganty, Kiefer, and Luttenberger proved that each context-free grammar in Chomsky normal form (CNFG) with h variables can be converted into a Parikh equivalent NFA with $O(4^h)$ states [2]. In [4] it was proven that if G generates a bounded language then we can obtain a DFA with $2^{h^{O(1)}}$ states, i.e., a number exponential in a polynomial of the number of variables. In this paper, we are able to extend such a result by removing the restriction to bounded languages. We also reduce the upper bound to $2^{O(h^2)}$. A milestone for obtaining such a result is the conversion of NFAs to Parikh equivalent DFAs presented in the first part of the paper. By suitably combining that result (in particular the polynomial conversion in the case of NFAs accepting nonunary strings) with the above mentioned result from [2] and with a result by Pighizzini, Shallit, and Wang [9] concerning the unary case, we prove that each context-free grammar in Chomsky normal form with h variables can be converted into a Parikh equivalent DFA with $2^{O(h^2)}$ states. From the results concerning the unary case, it follows that this bound is tight.

Even for this simulation, as for that of NFAs by Parikh equivalent DFAs, the main contribution to the state complexity of the resulting automaton is given by the unary part.

2 From NFAs to Parikh equivalent DFAs

In this section we present our first main contribution. From each n-state NFA A we derive a Parikh equivalent DFA A' with $e^{O(\sqrt{n \cdot \ln n})}$ states. Furthermore, we prove that this cost is tight.

Actually, as a preliminary step we obtain a result which is interesting *per se* (Theorem 1): if each string accepted by the given NFA A contains at least two different symbols, i.e., it is nonunary, then the Parikh equivalent DFA A' can be obtained with polynomially many states. Hence, the superpolynomial blowup is due to the unary part of the accepted language.

The proof of Theorem 1 gives a construction which uses a normal form for the Parikh image of the languages accepted by NFAs. Such a form is a refinement of a form presented in [3, 11].

Theorem 1. *For each n-state* NFA *accepting a language none of whose words are unary, there exists a Parikh equivalent* DFA *with a number of states polynomial in n.*

For the unary part, the following result proved by Chrobak in 1986 is useful.

Theorem 2 ([1]). *The state cost of the conversion of n-state unary* NFA*s into equivalent* DFA*s is $e^{\Theta(\sqrt{n \cdot \ln n})}$.*

Theorem 1 and Theorem 2 are useful to study the general case. From a given n-state NFA A with input alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$, for each $i = 1, \ldots, m$, we first build an n-state NFA A_i accepting the unary language $L(A) \cap a_i^*$. Using Theorem 2, we convert A_i into an equivalent DFA A'_i with $e^{O(\sqrt{n \cdot \ln n})}$ states. We can also build an $O(n)$-state NFA A_0 accepting all the nonunary strings belonging to $L(A)$. The NFA A_0 can be converted into a Parikh equivalent DFA A_n with a number of states polynomial in n. Using standard constructions, we combine DFAs $A'_1, \ldots, A'_m$ and A_n to finally obtain a DFA accepting a language Parikh equivalent to the language accepted by the original NFA A and with a number of states polynomial in n.

From this argument and from the optimality of the upper bound for the unary case (Theorem 2) we obtain the following result.

Theorem 3. *For each n-state* NFA*, there exists a Parikh equivalent* DFA *with $e^{O(\sqrt{n \cdot \ln n})}$ states. Furthermore, this cost is tight.*

3 From CFGs to Parikh Equivalent DFAs

In this section we extend the results of Section 2 to the conversion of CFGs in Chomsky normal form to Parikh equivalent DFAs. Actually, Theorem 1 will play an important role in order to obtain the main result of this section.

Even in this case the proof is given by splitting the unary and the nonunary parts of the language under consideration, converting the corresponding grammars into Parikh equivalent DFAs and, finally, recombining the DFAs so obtained into a DFA.

For the unary part, the conversion is done by using a result from [9] stating that for any CNFG with h variables that generates a unary language, there exists an equivalent DFA with less than 2^{h^2} states.

For the nonunary part, we first use a result from [2] stating that for a CNFG with h variables there exists a Parikh equivalent NFA with $O(4^h)$ variables. Then, we apply the construction used to prove Theorem 1 to the resulting NFA.

Theorem 4. *For any* CNFG *with h variables, there exists a Parikh equivalent* DFA *with at most $2^{O(h^2)}$ states.*

We finally observe that in [9] it was proven that there is a constant $c > 0$ such that for infinitely many $h > 0$ there exists a CNFG with h variables generating a unary language such that each equivalent DFA requires at least 2^{ch^2} states. This implies that the upper bound given in Theorem 4 cannot be improved.

References

1. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science 47, 149–158 (1986), corrigendum, ibid. 302 (2003) 497-498
2. Esparza, J., Ganty, P., Kiefer, S., Luttenberger, M.: Parikh's theorem: A simple and direct automaton construction. Information Processing Letters 111(12), 614–619 (2011)
3. Kopczyński, E., To, A.W.: Parikh images of grammars: Complexity and applications. In: Symposium on Login in Computer Science. pp. 80–89 (2010)
4. Lavado, G.J., Pighizzini, G.: Parikh's theorem and descriptional complexity. In: Proceedings of SOFSEM 2012. LNCS, vol. 7147, pp. 361–372. Springer (2012)
5. Lupanov, O.: A comparison of two types of finite automata. Problemy Kibernet 9, 321–326 (1963), (in Russian). German translation: Über den Vergleich zweier Typen endlicher Quellen, Probleme der Kybernetik 6, 329–335 (1966)
6. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: FOCS. pp. 188–191. IEEE (1971)
7. Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Transactions on Computers C-20(10), 1211–1214 (1971)
8. Parikh, R.J.: On context-free languages. Journal of the ACM 13(4), 570–581 (1966)
9. Pighizzini, G., Shallit, J., Wang, M.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. Journal of Computer and System Sciences 65(2), 393–414 (2002)
10. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. 3, 114–125 (1959)
11. To, A.W.: Parikh images of regular languages: Complexity and applications (February 2010), arXiv:1002.1464v2

Hoare Logic for Multiprocessing (work in progress)

Marina Lenisa and Daniel Pellarini

UNIUD, Italy

marina.lenisa@uniud.it, daniel.pellarini@gmail.coms

In the traditional approach of *Hoare logic*, the operational semantics of *concurrent programs* is explained via the *interleaving transition rule*. This reflects the execution on a single processor, where atomic actions of parallel components are interleaved and sequentially executed, and it does not directly account for multiprocessing systems.

Here we depart from the traditional approach, and we introduce a new *parallel operator* whose operational semantics directly reflects the execution on a *multiprocessing system*, where *disjoint* atomic actions of program components are executed in parallel.

Then, we develop a technique for verifying concurrent programs in this setting, inspired by the traditional Owicki-Gries method, see *e.g.* [1]. In the multiprocessing setting, the above technique substantially simplifies, by only requiring *local interference freedom*, in place of *global interference freedom* between the proof outlines of parallel components. We plan to implement a tool for the verification of the local interference freedom; this could then be used in combination with a tool for verifying sequential components, such as *Why3* [2], for verifying concurrent programs.

Coalgebraically (or game-theoretically), the new parallel operator can be interpreted as a kind of *conjunctive sum*, where each action consists of concurrent actions in more components. More in general, it would be interesting to explore what kind of parallelism corresponds to notions of sums alternative to the interleaving sum, which arise in the theory of coalgebras and games, [3–5].

The Language.
We focus on the language for parallel programs with synchronization $\mathcal{L}_{par}$, whose syntax is defined as follows:

Definition 1 (Syntax).
$(\mathcal{L}_{par} \ni) S ::= skip \mid x := t \mid S_1;\ S_2 \mid await\ B\ then\ S\ end \mid if\ B\ then\ S_1\ else\ S_2\ fi \mid$
$while\ B\ do\ S_1\ od \mid [S_1 \mid\mid\mid \ldots \mid\mid\mid S_n]$
where
- $x, y, \ldots$ *are variables;*
- t *is an expression built over a standard language for integer and boolean expressions;*
- B *is a boolean expression;*
- *programs not containing the* $\mid\mid\mid$ *operator are called* (sequential) components*;*
- *the program* S *in the* conditional atomic section $await\ B\ then\ S\ end$ *contains neither the* $\mid\mid\mid$ *operator nor* $while$ *subprograms;*
- *the components* $S_1, \ldots, S_n$ *in the parallel composition* $[S_1\ \mid\mid\mid \ldots \mid\mid\mid\ S_n]$ *do not contain the* $\mid\mid\mid$ *operator.*

We assume an abstract level of granularity of our language, so as skip, assignment, and the evaluation of an if/while-guard are considered as *atomic actions*, as well as *conditional atomic sections*. The latter is executed in *mutual exclusion*, *i.e.* the guard B of a command $await\, B\, then\, S\, end$ is evaluated in the current state and, if it is true, then S is executed atomically with the evaluation of B, and in mutual exclusion.

Formally, the transition system of the language $\mathcal{L}_{par}$ is defined by a set of rules for deriving judgements of the shape $\langle S, \sigma\rangle \to \langle S', \sigma'\rangle$, where $\sigma \in \Sigma$ is a *state*, *i.e.* a function from variables to values. The transition system that we consider includes the usual transition rules for the components (see [1], Chapter 9), but it differs from the standard one because the rule for *interleaving* is replaced by the rule for *parallel execution of atomic actions*: at each step, in a parallel program $[S_1 \;|||\ldots|||\; S_n]$, a maximal set of *disjoint atomic actions* in the components is executed. Informally, two components, S_1 and S_2, execute *disjoint atomic actions* if the variables *modified* in the next atomic command/guard of each component are *not* used in the next atomic command/guard of the other. That is no written variable can be shared in the actions executed by the two components; conditional atomic regions cannot be executed in parallel with any other component.

In the following, for states σ, τ and X set of variables, we denote by $\sigma = \tau\ mod\ X$ the fact that the states σ and τ coincide on all variables but those in X.

Definition 2 (Parallel Transition Rule).
(i) Let $\tau_1 = \sigma\ mod\ X$, $\tau_2 = \sigma\ mod\ Y$, for $X \cap Y = \emptyset$. We define the state $\tau_1 \uplus \tau_2$ by

$$(\tau_1 \uplus \tau_2)(x) = \begin{cases} \sigma(x) & \textit{if}\, x \notin X \cup Y \\ \tau_1(x) & \textit{if}\, x \in X \\ \tau_2(x) & \textit{if}\, x \in Y \end{cases}$$

(ii) Parallel transition rule:

$$\frac{\{< S_i,\ \sigma > \longrightarrow < S'_i,\ \tau_i >\}_{i\in I}}{< [S_1 \;|||\cdots\;|||\; S_n],\ \sigma > \longrightarrow < [T_1 \;|||\cdots\;|||\; T_n],\ \biguplus_{i=1}^{n} \tau_i >}$$

where $\{S_i\}_{i\in I}$ is a maximal set of components executing disjoint atomic actions, and $T_i = \begin{cases} S_i & \textit{if}\, i \notin I \\ S'_i & \textit{if}\, i \in I\ . \end{cases}$

With the above parallel transition rule we make two implicit assumptions. First, we assume that the number of processors available is not bounded, or at least not less than the maximum number of components which can execute disjointly. Possibly, one can impose a limitation on the number of components which can execute in parallel, according to the number of processors, but this will not change the theory, and hence, for simplicity, we work without this assumption. The second implicit assumption is that all atomic actions, being executed in parallel on different processors, have the same "cost" in terms of execution time.

Notice that, the final states generated by computations arising with the interleaving rule are in general more than those induced by computations arising with the parallel transition rule. Namely, with the latter, *maximal* sets of disjoint atomic actions are forced to be executed at each step, and hence not all interleaving executions are compatible. This will be exploited in order to simplify the verification of parallel programs.

Verification of Concurrent Programs.

The standard technique due to Owicki-Gries for the verification of concurrent programs is based on the construction of *standard proof outlines* for the components, *i.e.* proof outlines where each atomic command or atomic region is preceded by exactly one assertion, and on the control of *interference-freedom* between these proof outlines. Intuitively, this latter step substantially simplifies in our multiprocessing setting, because of the parallelism constraints. In the following, we propose a technique for verifying concurrent programs in our setting. After having built standard proof outlines for components in the usual way, we proceed as follows:

1. We build the graph of *abstract computations* of the parallel program; by an *abstract computation* we mean a computation where we forget about the states, and we only account for the sequence of programs that we reach by executing the original program, and for the atomic actions/conditional section executed at each step (see Definition 3 below). Each node in the graph of abstract computations represents a point in the parallel program reached during its execution, and it is labeled by the sequence of corresponding assertions annotating the proof outlines of the components. The arcs in the graph will be labeled by the sequence of atomic commands/guards or by the conditional atomic action executed at that step. Notice that the graph has a finite number of nodes.
2. Once the graph of abstract computations has been built, the proof of *interference freedom* between the proof outlines of the components reduces to a *local* check of non interference between the assertions labeling a node and the sequence of atomic actions or the atomic section labeling the outgoing arcs.

The graph of abstract computations.

Definition 3 (Abstract Transition System and Computation).

(i) The abstract transition system *consists of rules for deriving judgements $S \xrightarrow{l} S'$, where l is a label representing (a sequence of) atomic commands/guards/sections or the empty action ε (representing a computation that doesn't perform any action in that specific computation step),* i.e.*:*

$$l \ ::= \ skip \mid x := t \mid B \mid \varepsilon \mid await\ B\ then\ S\ end \mid \langle l_1, \ldots, l_n \rangle \ .$$

The abstract transition rules are the following:

$$\frac{}{skip \xrightarrow{skip} E} \qquad \frac{}{x := t \xrightarrow{x:=t} E} \qquad \frac{}{S \xrightarrow{\varepsilon} S}$$

$$\frac{}{await\, B\, then\, S\, end \xrightarrow{await\, B\, then\, S\, end} E} \qquad \frac{S_1 \xrightarrow{l_1} S_1'}{S_1; S_2 \xrightarrow{l_1} S_1'; S_2}$$

$$\frac{}{if\, B\, then\, S_1\, else\, S_2\, fi \xrightarrow{B} S_1} \qquad \frac{}{if\, B\, then\, S_1\, else\, S_2\, fi \xrightarrow{\neg B} S_2}$$

$$\frac{}{while\, B\, do\, S\, od \xrightarrow{\neg B} E} \qquad \frac{}{while\, B\, do\, S\, od \xrightarrow{B} S; while\, B\, do\, S\, od}$$

3

$$\frac{\{S_i \xrightarrow{l_i} S_i'\}_{i \in I}}{[S_1' \mid\mid\mid \ldots \mid\mid\mid S_n] \xrightarrow{<l_1', \ldots, l_n'>} [S_1' \mid\mid\mid \ldots \mid\mid\mid S_n']}$$

where $\{S_i\}_{i \in I}$ *is a maximal set of components executing disjoint atomic actions and* $l_i' = \begin{cases} l_i & \textit{if } i \in I \\ \varepsilon & \textit{if } i \notin I \ . \end{cases}$

(ii) An abstract computation *is a (finite or infinite) sequence* $S \xrightarrow{l_1} S_1 \xrightarrow{l_2} \ldots \xrightarrow{l_n} S_n \ldots$

Notice that in all abstract computations, even the infinite ones, only finitely many different programs can appear.

Now we sketch how to define the (finite rooted) graph representing the abstract computations generating from a program $S = [S_1 \mid\mid\mid \ldots \mid\mid\mid S_n]$. Each node n represents a point in the computation of S, and it is labeled by the n-tuple of assertions $\langle p^1_{j_1}, \ldots, p^n_{j_n} \rangle$ appearing in the proof outlines at that point. The construction of the graph starts from the root n, which is labeled with the initial assertions, and proceeds by analyzing, for each created node n', the abstract transitions arising from the corresponding program $[S_1' \mid\mid\mid \ldots \mid\mid\mid S_n']$: for each transition $[S_1' \mid\mid\mid \ldots \mid\mid\mid S_n'] \xrightarrow{<l_1, \ldots, l_n>} [S_1'' \mid\mid\mid \ldots \mid\mid\mid S_n'']$, a new node n'' is built, if it does not already exists, corresponding to $[S_1'' \mid\mid\mid \ldots \mid\mid\mid S_n'']$, and an arc is drawn from n' to n'', labeled by $\langle l_1, \ldots, l_n \rangle$. Some optimizations can be performed during the graph construction, by avoiding to represent transitions corresponding to the evaluation of a guard which is not compatible with the current assertions.

Local interference freedom. Once the graph of abstract computations has been built, the non-interference checks can be performed at a local level. That is, for any node n, for any outgoing arc and any atomic action appearing in its label and in the proof outline of a component, it is sufficient to check that this atomic action does not interfere with the assertions of the node n, which appear in the proof outlines of other components.

Coalgebraically speaking, this new parallel operator can be interpreted as a kind of *conjunctive sum*, where each action consists of concurrent actions in more components. This is currently being studied as a natural continuation of this work.

References

1. K. Apt, F. de Boer, E. Olderog. *Verification of Sequential and Concurrent Programs*, Springer, 2009.
2. F. Bobot, J-C. Filliâtre, C. Marché. G. Melquiond, A. Paskevich, *The Why3 Platform*, Version 0.72, May 2012, available at http://why3.lri.fr/#documentation.
3. J.H. Conway. On Numbers and Games, A K Peters Ltd, 2001.
4. F. Honsell, M. Lenisa. Conway Games, algebraically and coalgebraically, *Logical Methods in Computer Science* **7(3)**, 2011.
5. F. Honsell, M. Lenisa, R. Redamalla. Equivalences and Congruences on Infinite Conway Games, *Theoretical Informatics and Applications* **46(2)**, 231–259, 2012.

Size constrained clustering problems in fixed dimension

Jianyi Lin

Dipartimento di Informatica, Università degli Studi di Milano, Italy
jianyi.lin@unimi.it

Extended Abstract

Clustering or cluster analysis [1] is a classical method in unsupervised learning and one of the most used techniques in statistical data analysis. Clustering has a wide range of applications in many areas like pattern recognition, medical diagnostics, data mining, biology, market research and image analysis among others. A cluster is a set of data points that in some sense are similar to each other, and clustering is a process of partitioning a data set into disjoint clusters. In *distance clustering*, the similarity among data points is obtained by means of a *distance* function.

Fixed a norm $\| \|_p$ ($p \geq 1$), the *clustering problem* consists in finding for a finite point set $X \subset \mathbb{Q}^d$ and an integer k, a k-partition $\{A_1, ..., A_k\}$ of X that minimizes the cost function

$$W(A_1, ..., A_k) = \sum_{i=1}^{k} \sum_{x \in A_i} \|x - C_{A_i}\|_p^p \tag{1}$$

where C_{A_i} is the *p-centroid* of A_i, i.e.

$$C_{A_i} = \arg\min_{\mu} \sum_{x \in A_i} \|x - \mu\|_p^p$$

Distance clustering is a difficult problem. For an arbitrary dimension d, assuming the Euclidean norm ($p = 2$), the problem is NP-hard even if the number k of clusters equals 2 [2]; the same occurs if $d = 2$ and k is arbitrary [3,4]. For the Euclidean distance, a well-known heuristic is Lloyd's algorithm [5,6], also known as the k-Means Algorithm; however there is no guarantee that the solution yielded by this procedure approximates the global optimum. This algorithm is usually very fast, but it can require exponential time in the worst case [7].

In real-world problems, often people have some information on the clusters: incorporating this information into traditional clustering algorithms can increase the clustering performance. Problems that include background information are called *constrained clustering* problems and are divided in two classes.

On the one hand, clustering problems with instance-based constraints typically comprise a set of must-link constraints or cannot-link constraints [8], defining pairs of elements that must be included, respectively, in the same cluster or in different clusters.

On the other hand, clustering problems with cluster-based constraints [9,10] incorporate constraints concerning the size of the possible clusters. Recently, in [11] cluster size constraints are used for improving clustering accuracy; this approach, for instance, allows one to avoid extremely small or large clusters in standard cluster analysis.

Here we study a constrained clustering problem where the size of clusters is included in the instance. This problem, called *Size Constrained Clustering Problem* (SCC), is formally defined as follows: given a set $X \subset \mathbb{Q}^d$ of n points and k many positive integers $m_1, ..., m_k$ such that $\sum_1^k m_i = n$, find a k-partition $\{A_1, ..., A_k\}$ of X that minimizes the cost function $W(A_1, ..., A_k)$ such that $|A_i| = m_i$ for each $i = 1, ..., k$. This problem was studied in [12,13] and it is known to be a difficult problem. More precisely, the following results hold [13]:

1) For every norm $\| \|_p$ with $p > 1$, SCC with fixed clustering size k is NP-hard, even in the case $k = 2$ and $m_1 = m_2 = \frac{n}{2}$.
2) For every norm $\| \|_p$ with $p \geq 1$, SCC with fixed dimension d is NP-hard, even in the case $d = 1$.

As a consequence, we can't expect to obtain a polynomial-time algorithm for solving the general SCC problem.

In this paper we investigate SCC in the plane ($d = 2$) with a fixed clustering size $k = 2$. In particular, we consider the following two problems:
• 2-SCC in the Plane:
Given a point set $X = \{x_1, ..., x_n\} \subset \mathbb{Q}^2$ and a positive integer $m \leq \frac{n}{2}$, find a 2-partition $\{A, \bar{A}\}$ of X with $|A| = m, |\bar{A}| = n - m$, that minimizes

$$W(A, \bar{A}) = \sum_{x \in A} \|x - C_A\|_2^2 + \sum_{x \in \bar{A}} \|x - C_{\bar{A}}\|_2^2$$

where C_A and $C_{\bar{A}}$ are the centroid of A and $\bar{A}$ respectively.
• Full 2-SCC in the Plane:
Given a point set $X = \{x_1, ..., x_n\} \subset \mathbb{Q}^2$, for all integers m, $1 \leq m \leq \frac{n}{2}$, find the optimal 2-partition $\pi_m = \{A_m, \bar{A}_m\}$, with $|A_m| = m$.

The main results we obtain are the following:

1) There is an algorithm for solving Full 2-SCC in the Plane in time $O(n^2 \cdot \log n)$.
2) There is an algorithm for solving 2-SCC in the Plane in time $O(n \sqrt[3]{m} \cdot \log^2 n)$.

It should be observed that, the algorithm for solving 2-SCC in the plane requires the application of methods for enumerating the k-sets of a collection of points in the plane, which is a challenging problem [14] in combinatorial geometry.

Here we also study the problem 2-SCC in fixed dimension d. First, we use a separation result [13] stating that if $\{A, \bar{A}\}$ is an optimal solution of an instance of the 2-SCC problem, then A and $\bar{A}$ are separated by an hypersurface of the form

$$\|x - \alpha\|_p^p - \|x - \beta\|_p^p = c$$

for some constant parameters $\alpha, \beta \in \mathbb{R}^d$, $c \in \mathbb{R}$. By applying a suitable method for decomposing the parameter space $\mathbb{R}^{2d+1}$, one can compute a set of optimal 2-partitions $\pi_m = \{A_m, \bar{A}_m\}$ such that $|A_m| = m$, for $m = 1, ..., \lfloor \frac{n}{2} \rfloor$. This allows us to design an algorithm for the Full 2-SCC problem in fixed dimension d that works in polynomial time both in n and p. To obtain this result we make use of concepts and methods of real algebraic geometry, and in particular we apply the cylindrical algebraic decomposition [15].

In this work we also study another variant of the clustering problem, called *Relaxed Constraints Clustering Problem* (RCC), which is defined as follows: given a point set $X = \{x_1, ..., x_n\} \subset \mathbb{Q}^d$, an integer $k > 1$ and a finite set $\mathcal{M}$ of positive integers, find a k-partition $\{A_1, ..., A_k\}$ of X with

$$|A_i| \in \mathcal{M} \quad \text{for all } i = 1, ..., k$$

that minimizes the cost function

$$W(A_1, ..., A_k) = \sum_{i=1}^{k} \sum_{x \in A_i} \|x - C_{A_i}\|_p^p.$$

We prove that for the euclidean norm $\| \ \|_2$, the decision version of RCC in dimension $d = 2$ is NP-complete even in the case $\mathcal{M} = \{2, 3\}$. On the contrary, RCC in dimension 1 is known to be solvable in polynomial time by a dynamic programming technique [12].

Acknowledgements

The author wishes to acknowledge helpful discussions with Alberto Bertoni and Massimiliano Goldwurm.

References

1. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd edn. Springer-Verlag (2009)
2. Aloise, D., Deshpande, A., Hansen, P., Popat, P.: NP-hardness of Euclidean sum-of-squares clustering. Machine Learning **75** (2009) 245–249
3. Mahajan, M., Nimbhorkar, P., Varadarajan, K.: The Planar k-Means Problem is NP-Hard. In Das, S., Uehara, R., eds.: WALCOM: Algorithms and Computation. Volume 5431 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg (2009) 274–285
4. Vattani, A.: The hardness of k-means clustering in the plane. manuscript (2009)

5. Lloyd, S.: Least squares quantization in PCM. IEEE Transactions on Information Theory **28**(2) (1982) 129–137
6. MacQueen, J.B.: Some method for the classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Structures. (1967) 281–297
7. Vattani, A.: K-means requires exponentially many iterations even in the plane. In: Proceedings of the 25th Symposium on Computational Geometry (SoCG). (2009)
8. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proc. of the 17th Intl. Conf. on Machine Learning. (2000) 1103–1110
9. Bradley, P.S., Bennett, K.P., Demiriz, A.: Constrained K-Means Clustering. Technical Report MSR-TR-2000-65, Miscrosoft Research Publication (May 2000)
10. Tung, A., Han, J., Lakshmanan, L., Ng, R.: Constraint-Based Clustering in Large Databases. In Van den Bussche, J., Vianu, V., eds.: Database Theory ICDT 2001. Volume 1973 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg (2001) 405–419
11. Zhu, S., Wang, D., Li., T.: Data clustering with size constraints. Knowledge-Based Systems **23**(8) (2010) 883–889
12. Saccà, F.: Problemi di Clustering con Vincoli: Algorithmi e Complessità. PhD thesis, University of Milan, Milan (2010)
13. Bertoni, A., Goldwurm, M., Lin, J., Saccà, F.: Size Constrained Distance Clustering: Separation Properties and Some Complexity Results. Fundamenta Informaticae **115**(1) (2012) 125–139
14. Erdős, P., Lovász, L., Simmons, A., Straus, E.G.: Dissection graphs of planar point sets. In: A survey of combinatorial theory (Proc. Internat. Sympos., Colorado State Univ., Fort Collins, Colo., 1971). North-Holland, Amsterdam (1973) 139–149
15. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In Barkhage, E., ed.: Proc. 2nd GI Conf. on Automata Theory and Formal Lang. Volume 33 of LNCS., Berlin, Springer (1975) 134–183

Automata and Logic for Floyd Languages

Violetta Lonati[1], Dino Mandrioli[2], Matteo Pradella[2]

[1] DSI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@dsi.unimi.it
[2] DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
{dino.mandrioli, matteo.pradella}@polimi.it

Floyd languages (FL), as we renamed Operator Precedence Languages after their inventor, were originally introduced to support deterministic parsing of programming and other artificial languages [1]; then, interest in them decayed for several decades, probably due to the advent of more expressive grammars, such as LR ones [2] which also allow for efficient deterministic parsing.

In another context Visual Pushdown Languages (VPL) have been introduced and investigated [3] with the main motivation to extend to them the same or similar automatic analysis techniques -noticeably, model checking- that have been so successful for regular languages. Recently we discovered that VPL are a proper subclass of FL, which in turn enjoy the same properties that make regular and VP languages amenable to extend to them typical model checking techniques; in fact, to the best of our knowledge, FL are the largest family closed w.r.t. Boolean operation, concatenation, Kleene * and other classical operations [4]. Another relevant feature of FL is 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.

Originally, Floyd languages were defined in terms of grammars. In this work we present an appropriate automata family that recognizes exactly FL [5], together with a complete characterization of FL in terms of a suitable Monadic Second-Order (MSO) logic [6]. In this way, as well as with regular and VP languages, one can, for instance, state a language property by means of a MSO formula, then automatically verify whether a given FA accepts a language that enjoys that property.

Operator precedence alphabet and chains

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The empty string is denoted ϵ. We use a special symbol # not in Σ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the look-back and look-ahead of one character to determine the next parsing action [2].

An *operator precedence matrix* (OPM) M over an alphabet Σ is a partial function $(\Sigma \cup \{\#\})^2 \to \{\lessdot, \doteq, \gtrdot\}$, that with each ordered pair (a, b) associates the OP relation $M_{a,b}$ holding between a and b. We call the pair (Σ, M) an *operator precedence alphabet* (OP). Relations $\lessdot, \doteq, \gtrdot$, are named yields precedence, equal in precedence, takes precedence, respectively. By convention, the initial # can only yield precedence, and other symbols can only take precedence on the ending #.

If $M_{a,b} = \circ$, where $\circ \in \{\lessdot, \doteq, \gtrdot\}$, 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 *complete* if $M_{a,b}$ is defined for every a and b in Σ. Moreover in the following we assume that M is $\doteq$*-acyclic*, which means that $c_1 \doteq c_2 \doteq \ldots \doteq c_k \doteq c_1$ does not hold for any $c_1, c_2, \ldots c_k \in \Sigma, k \geq 1$. See [7,4,5] for a discussion on this hypothesis.

Given an OP alphabet, the OPM M assigns a structure to strings in Σ^*, i.e., a string can be uniquely associated with a tree.

A *simple chain* is a string $c_0 c_1 c_2 \ldots c_\ell c_{\ell+1}$, written as ${}^{c_0}[c_1 c_2 \ldots c_\ell]^{c_{\ell+1}}$, such that: $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$, $c_i \in \Sigma$ for every $i = 1, 2, \ldots \ell$, and $c_0 \lessdot c_1 \doteq c_2 \ldots c_{\ell-1} \doteq c_\ell \gtrdot c_{\ell+1}$. A *composed chain* is a string $c_0 s_0 c_1 s_1 c_2 \ldots c_\ell s_\ell c_{\ell+1}$, where ${}^{c_0}[c_1 c_2 \ldots c_\ell]^{c_{\ell+1}}$ is a simple chain, and $s_i \in \Sigma^*$ is the empty string or is such that ${}^{c_i}[s_i]^{c_{i+1}}$ is a chain (simple or composed), for every $i = 0, 1, \ldots, \ell$. Such a composed chain will be written as ${}^{c_0}[s_0 c_1 s_1 c_2 \ldots c_\ell s_\ell]^{c_{\ell+1}}$. A string $s \in \Sigma^*$ is *compatible* with the OPM M if ${}^{\#}[s]^{\#}$ is a chain.

Floyd automata

Floyd automata are stack-based automata perfectly carved on the generation mechanism of the traditional Floyd grammars [1]. Not surprisingly they inherit some features of VPA (mainly a clear separation between push and pop operations) and maintain some typical behavior of shift-reduce parsing algorithms [2]; however, they also exhibit some distinguishing features.

A nondeterministic *Floyd automaton* (FA) is a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ where: (Σ, M) is a precedence alphabet, Q is a set of states (disjoint from Σ), $I, F \subseteq Q$ are sets of initial and final states, respectively, $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is the transition function. The transition function is the union of two disjoint functions: $\delta_{\text{push}} : Q \times \Sigma \to 2^Q$ and $\delta_{\text{flush}} : Q \times Q \to 2^Q$.

To define the semantics of the automaton, we introduce some notations. We use letters $p, q, p_i, q_i, \ldots$ for states in Q and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked* symbols. Let $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in Γ as $[a\ q]$, $[a' q]$, or $[\#\ q]$, respectively. We set $smb([a\ q]) = smb([a' q]) = a$, $smb([\#\ q]) = \#$, and $st([a\ q]) = st([a' q]) = st([\#\ q]) = q$.

A *configuration* of a FA is any pair $C = \langle B_1 B_2 \ldots B_n,\ a_1 a_2 \ldots a_m \rangle$, where $B_i \in \Gamma$ and $a_i \in \Sigma \cup \{\#\}$. The first component represents the contents of the stack, while the second component is the part of input still to be read.

A computation is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $smb(B_n)$ and a_1:

(push) if $smb(B_n) \doteq a_1$ then $C_1 = \langle B_1 \ldots B_n[a_1\ q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
(mark) if $smb(B_n) \lessdot a_1$ then $C_1 = \langle B_1 \ldots B_n[a_1{}' q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
(flush) if $smb(B_n) \gtrdot a_1$ then let i be the greatest index such that $smb(B_i) \in \Sigma'$ and $C_1 = \langle B_1 \ldots B_{i-2}[smb(B_{i-1})\ q],\ a_1 a_2 \ldots a_m \rangle$, with $q \in \delta_{flush}(st(B_n), st(B_{i-1}))$.

Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push}; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that delimit the portion of the stack to be removed; notice that in this move the input symbol is not consumed and it remains available for the following move.

Finally, we say that a configuration $[\# \, q_I]$ is *starting* if $q_I \in I$ and a configuration $[\# \, q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\# \, q_I], \; x\# \rangle \overset{*}{\vdash} \langle [\# \, q_F], \; \# \rangle, q_I \in I, q_F \in F \right\}.$$

The chains fully determine the structure of the parsing of any automaton over (Σ, M). Indeed, if the automaton performs the computation $\langle [a \, q_0], \; sb \rangle \overset{*}{\vdash} \langle [a \, q], \; b \rangle$, then ${}^a[s]^b$ is necessarily a chain over (Σ, M) and the first move in the above computation is a mark from state q_0, whereas the last one is a flush towards state q labelled by q_0. Such a computation corresponds to the parsing by the automaton of the string $s_0 c_1 \ldots c_\ell s_\ell$ within the context a,b; this context contains all information needed to build the subtree whose frontier is that string. This is a distinguishing feature of FL, not shared by other deterministic languages: we call it the *locality principle* of Floyd languages.

In other terms, given an OP alphabet, the OPM M assigns a structure to any string in Σ^* compatible with M; a FA defined on the OP alphabet selects an appropriate subset within such a “universe”. In some sense this property is yet another variation of the fundamental Chomsky-Shützenberger theorem.

Logic characterization of Floyd languages

Our characterization of FL in terms of a suitable Monadic Second Order (MSO) logic follows the approach originally proposed bu Büchi for regular languages and subsequently extended by Alur and Madhusudan for VPL. The essence of the approach consists in defining language properties in terms of relations between the positions of characters in the strings: first order variables are used to denote positions whereas second order ones denote subsets of positions; then, suitable constructions build an automaton from a given formula and conversely, in such a way that formula and corresponding automaton define the same language. The extension designed by [3] introduced a new basic binary predicate $\leadsto$ in the syntax of the MSO logic, $x \leadsto y$ representing the fact that in positions x and y two matching parentheses –named call and return, respectively in their terminology– are located. In the case of FL, however, we have to face new problems.

Both finite state automata and VPA are real-time machines, i.e., they read one input character at every move; this is not the case with more general machines such as FA, which do not advance the input head when performing flush transitions, and may also apply many flush transitions before the next push or mark which are the transitions that consume input. As a consequence, whereas in the logic characterization of regular and VP languages any first order variable can belong to only one second order variable representing an automaton state, in this case –when the automaton performs a flush– the same position may correspond to different states and therefore belong to different second-order variables.

In VPL the $\leadsto$ relation is one-to-one, since any call matches with only one return, if any, and conversely (with the exception of unmatched calls and returns, where many call positions can be in relation with +infinite and symmetrically). In FL, instead the same position y can be "paired" with different positions x in correspondence of many flush transitions with no push/mark in between, as it happens for instance when parsing a derivation such as $A \overset{*}{\Rightarrow} \alpha^k A$, consisting of k immediate derivations $A \Rightarrow \alpha A$; symmetrically the same position x can be paired with many positions y.

Consider an OP alphabet (Σ, M). We introduce a relation over positions of characters in any word $s \in \Sigma^*$. For $0 \le x < y \le |s| + 1$, we say that (x, y) is a *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.

Let us define a countable infinite set of first-order variables $x, y, \ldots$ and a countable infinite set of monadic second-order (set) variables $X, Y, \ldots$. The $\mathrm{MSO}_{\Sigma,M}$ (*monadic second-order logic* over (Σ, M)) is defined by the following syntax:

$$\varphi := a(x) \mid x \in X \mid x \le y \mid x \curvearrowright y \mid x = y + 1 \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $a \in \Sigma$, x, y are first-order variables and X is a set variable.

$\mathrm{MSO}_{\Sigma,M}$ formulae are interpreted over (Σ, M) strings and the positions of their characters in the following natural way: first-order variables are interpreted over positions of the string; second-order variables are interpreted over sets of positions; $a(x)$ is true iff the character in position x is a; $x \curvearrowright y$ is true iff (x, y) is a chain boundary; the other logical symbols have the usual meaning.

A sentence is a formula without free variables. The language of all strings $s \in \Sigma^*$ such that $\#s\# \models \varphi$ is denoted by $L(\varphi) = \{s \in \Sigma^* \mid \#s\# \models \varphi\}$, where $\models$ is the standard satisfaction relation.

This characterization completes a research path that began more than four decades ago and was resumed only recently with new -and old- goals. FL enjoy most of the nice properties that made regular languages highly appreciated and applied to achieve decidability and, therefore, automatic analysis techniques.

References

1. Floyd, R.W.: Syntactic analysis and operator precedence. Journ. ACM **10** (1963) 316–333
2. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM **56** (2009)
4. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. Journal of Computer and System Science (2012) to appear.
5. Lonati, V., Mandrioli, D., Pradella, M.: Precedence automata and languages. In Kulikov, A.S., Vereshchagin, N.K., eds.: CSR. Volume 6651 of Lecture Notes in Computer Science., Springer (2011) 291–304
6. Lonati, V., Mandrioli, D., Pradella, M.: Logic characterization of Floyd languages. CoRR-arXiv **1204.4639** (2012) http://arxiv.org/abs/1204.4639.
7. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Information and Control **37** (1978) 115–133

The algebra and geometry of networks

Davide Maglia, N. Sabadini, Filippo Schiavio, and R.F.C. Walters

University of Insubria, Como, Italy

Networks of components have a compositional description in terms of the algebra of symmetric or braided monoidal categories in which each object has a commutative Frobenius or separable algebra structure compatible with the tensor product. They also have a geometric description [11] - the free such algebra (in the separable symmetric case) is the category of cospans of multigraphs, arrows of which have a pictorial representation; in the Frobenius braided case the geometry is more complicated, capturing not only the connection between components but also their entanglement. These results are in the line introduced by Penrose [1], and Joyal and Street [3], and were obtained by Sabadini and Walters with collaborators Katis and Rosebrugh in earlier work, especially [6–9, 11, 12], beginning with the work on relations with Carboni [2] in 1987. The work has numerous antecedents - we mention just S. Eilenberg, S.L. Bloom, Z. Esik, Gh. Stefanescu. The algebra has connections with quantum field theory ([10]).

The present work presents two developments. The first is some initial work in classifying tangled circuits; the second is a tool for composing cospans of graphs and calculating executions of nets of parallel automata.

1 Tangled circuits

The free braided algebra of the type described above was introduced by Rosebrugh, Sabadini and Walters [14] under the name Tangled Circuits since the geometry captures not only the connection between components but their entanglement.

A *commutative Frobenius algebra* in a braided monoidal category with twist τ ([4]) consists of an object G and four arrows $\nabla : G \otimes G \to G$, $\Delta : G \to G \otimes G$, $n : I \to G$ and $e : G \to I$ making (G, ∇, e) a monoid, (G, Δ, n) a comonoid and satisfying the equations

$$\begin{aligned}(1_G \otimes \nabla)(\Delta \otimes 1_G) = \Delta\nabla = (\nabla \otimes 1_G)(1_G \otimes \Delta) : G \otimes G \to G \otimes G \\ \nabla\tau = \nabla : G \otimes G \to G \\ \tau\Delta = \Delta : G \to G \otimes G\end{aligned}$$

A *multigraph* M consists of two sets M_0 (objects, vertices or wires) and M_1 (arrows, edges or components) and two functions $dom : M_1 \to M_0^*$ and $cod : M_1 \to M_0^*$ where M_0^* is the free monoid on M_0.

Given a multigraph M the free braided strict monoidal category in which the objects of M are equipped with commutative Frobenius algebra structures is called $\mathbf{TCircD}_M$. Its arrows are called *tangled circuit diagrams*, or more briefly circuit diagrams. In the case that M has one vertex and no arrows we will denote $\mathbf{TCircD}_M$ simply as $\mathbf{TCircD}$.

Determining whether two arrows are equal in this category is difficult since it seems to include the problem of classifying knots as a special case. We present some initial work in classifying a special class of arrows which we call blocked braids, that is, an arrow of the form $S \circ B \circ R$ where $R : I \to X^n$ (X^n the tensor power of X), $B : X^n \to X^n$ is a braid on n strings, and $S : X^n \to I$. Here are two examples of distinct blocked braids:

We show that blocked braids on less than four strings are finite in number, whereas with four or more strings the number is infinite. (Notice that there are an infinite number of braids on two strings, whereas there are only two blocked braids on two strings: $\tau = \tau^{-1}$ and 1.)

The method of proof that two tangled circuits are distinct is by associating invariants to tangled circuits - if we manage to find distict invariants then certainly the circuits are distinct. The invariants we use are in fact functors which preserve the braided monoidal and Frobenius structure from the category of tangled circuits to simpler categories such as the following category of *tangled relations*:

Let G be a group. The objects of $\mathbf{TRel}_G$ are the formal powers of G, and the arrows from G^m to G^n are relations R from the set G^m to the set G^n satisfying:

1) if $(x_1, ..., x_m)R(y_1, ...y_n)$ then also for all g in G
   $(g^{-1}x_1g, ..., g^{-1}x_mg)R(g^{-1}y_1g, ..., g^{-1}y_mg)$,
2) if $(x_1, ..., x_m)R(y_1, ...y_n)$ then $x_1...x_m(y_1...y_n)^{-1} \in Z(G)$ (the center of G).

Composition and identities are defined to be composition and identity of relations.

It is straightforward to verify that $\mathbf{TRel}_G$ is a category. We introduce some useful notation. Write $x = (x_1, ..., x_m)$, $y = (y_1, ..., y_n)$, and so on. Write $\overline{x} = x_1x_2...x_m$ and for g, h in G, as $g^h = hgh^{-1}$. For g in G write $x^g = (x_1^g, x_2^g, ..., x_m^g)$. Thus, $(\overline{x})^g = \overline{x^g}$, and of course for any x, y in $G^m \times G^n$, $x^g y^g = (xy)^g$ where we write xy for $(x_1, ..., x_m, y_1, ..., y_n)$.

Then $\mathbf{TRel}_G$ is a braided strict monoidal category with tensor defined on objects by $G^m \otimes G^n = G^{m+n}$ and on arrows by product of relations. The twist

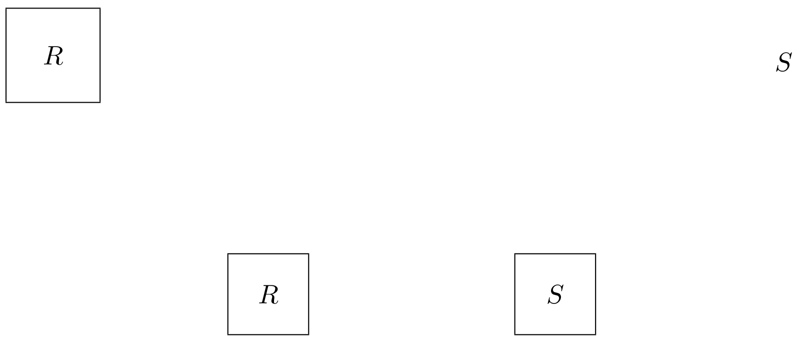
$$\tau_{m,n} : G^m \otimes G^n \to G^n \otimes G^m$$

is the functional relation

$$(x, y) \sim (y^{\overline{x}}, x)$$

The results we mention above are obtained by choosing suitable groups and tangled relations to distinguish tangled circuits.

Instead, an interesting equation in the Tangled Circuit category is the so-called Dirac's belt trick, the unwinding of two full twists of a belt without rotating the ends; it amounts to the equality of the following two blocked braids:



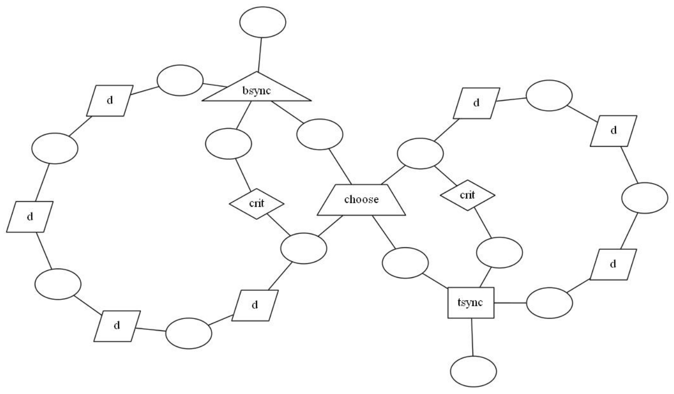
The category of tangled circuits may be relevant to modelling quantum computing with anyons, in a way similar to [13].

2 A graphic tool for executing nets of automata

The second development is a graphic tool for calculating compositionally cospans of multigraphs and for searching the state space of sequential and parallel networks of automata (the components have state). The state space of a sequential network is a colimit, and of a parallel network is a limit [12]. This has close connections with our paper [5] and recent work of Sobocinski [15, 16] on Petri nets.

The tool defines a language for describing expressions in *Span*(*Graph*) ([6]). However instead of evaluating the expression (with associated explosion of state) it calculates the geometry of the expression by composing in *Cospan*(*Graph*), with a graphic output using GraphViz, and permits (incomplete) exploration of the state space.

The following is an example of the (static) graphic output with the geometry of a system:



As an example of the language for specifying components notice that the component *bsynch* above is defined by

```
bsync=span(3,1){
 00, 01, 10, 11
 00 -> 00 : 0,0,0/0
 00 -> 01 : 0,0,1/0
 00 -> 11 : 0,3,1/0
 10 -> 11 : 5,0,1/0
 10 -> 10 : 5,0,0/0
 01 -> 11 : 5,3,0/0
 01 -> 01 : 5,0,0/0
 11 -> 00 : 0,0,0/2}
```

Further the system as a whole is defined (where D=Δ, ID=identity, U=unit, CU=counit, d=delay, crit=critical delay) by

```
diagonal_delay = crit ; D
expr = (tsync * bsync) ; (diagonal_delay * diagonal_delay) ; (ID * choose * ID) ;
               ( (d;d;d) * ID * ID * (d;d;d;d))
system = (U * ID * ID * U) ; (ID * U * ID * ID * ID * ID * U * ID) ;
         (ID * ID * expr * ID * ID) ; (ID * CU * ID * ID * CU * ID) ; (CU * CU)
```

For more details, and an example of the execution of this (mutual exclusion) system see the post at *http://rfcwalters.blogspot.it/2012/01/petri-nets.html.*

References

1. R. Penrose, *Applications of negative dimensional tensors*, in Combinatorial mathematics and its applications, Academic Press (1971).

2. A. Carboni and R.F.C. Walters, *Cartesian bicategories I*, Journal of Pure and Applied Algebra, 49 (1987) 11-32.
3. A. Joyal, R.H. Street, *The geometry of tensor calculus I*, Adv. Math. 88 (1991) 55-112.
4. A. Joyal, R.H. Street, *Braided monoidal categories*, Adv. Math. 102 (1993) 20-87.
5. P. Katis, N. Sabadini, R.F.C. Walters, *Representing P/T nets in Span(Graph)*, Proceedings AMAST '97, SLNCS 1349 (1997) 307-321.
6. P. Katis, N. Sabadini, R.F.C. Walters, *Span(Graph): an algebra of transition systems*, Proceedings AMAST '97, SLNCS 1349 (1997) 322-336.
7. P. Katis, N. Sabadini, R.F.C. Walters, *A formalisation of the IWIM Model*, in: Proc. COORDINATION 2000,(Eds.) Porto A., Roman G.-C., LNCS 1906, Springer Verlag, (2000) 267-283.
8. P. Katis, N. Sabadini, R.F.C. Walters, *On the algebra of systems with feedback and boundary*, Rendiconti del Circolo Matematico di Palermo Serie II, Suppl. 63 (2000), 123-156.
9. P. Katis, N. Sabadini, R.F.C. Walters, *Compositional minimization in Span(Graph): Some examples*, Electronic Notes in Theoretical Computer Science 104C,(2004) 181-197.
10. J. Kock, *Frobenius algebras and 2D Topological Quantum Field Theories*, Cambridge University Press (2004).
11. R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Generic commutative separable algebras and cospans of graphs*, Theory and Applications of Categories, 15, (2005) 264-277.
12. R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Calculating colimits compositionally*, Montanari Festschrift, LNCS 5065,(2008) 581592
13. L. de Francesco Albasini, N. Sabadini and R.F.C. Walters, *An algebra of automata that includes both classical and quantum entities*, Electr. Notes Theor. Comput. Sc. 270 (2011) 263-270.
14. R. Rosebrugh, N. Sabadini, R.F.C. Walters, *Tangled Circuits*, arXiv:1110.0715 (2011).
15. Jennifer Lantair, Pawel Sobocinski, *WiCcA: LTS generation tool for wire calculus*, Proceeding CALCO'11 Proceedings of the 4th international conference on Algebra and coalgebra in computer science, (2011) 407-412.
16. Roberto Bruni, Hernan Melgratti, Ugo Montanari and Pawel Sobocinski, *Connector algebras for C/E and P/T nets interactions*, preprint, (2012).

On Pushdown Store Languages[⋆] [⋆⋆]

Andreas Malcher[1], Katja Meckel[1], Carlo Mereghetti[2], and Beatrice Palano[2]

[1] Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
`{malcher,meckel}@informatik.uni-giessen.de`
[2] Dipartimento di Informatica, Università degli Studi di Milano
via Comelico 39/41, 20135 Milano, Italy
`{carlo.mereghetti,beatrice.palano}@unimi.it`

Abstract. We design *succinct nondeterministic finite automata* accepting *pushdown store languages* — i.e., the languages consisting of the pushdown contents along accepting computations of pushdown automata. Then, several restricted variants of pushdown automata are considered, leading to improved constructions. Finally, we apply our results to decidability questions related to pushdown automata.

Keywords: pushdown automata; pushdown store languages;

1 Introduction

Beside the formal definition of the accepted or generated language, the introduction of an accepting or generating device always brings the attention to several "auxiliary" formal structures related to the device itself (see, e.g., [5, 10]). Such structures are not only interesting *per se*, but their investigation has often other relevant motivations.

In this paper, we focus on *pushdown store languages* for pushdown automata (PDA). Given a PDA M, its pushdown store language $P(M)$ consists of all words occurring on the pushdown store along *accepting* computations of M. It is known from [1, 4] that, surprisingly enough, $P(M)$ is *regular*. Here, we design *succinct nondeterministic finite automata (NFA)* for $P(M)$. In Section 3, we outline the construction of NFA, whose size (i.e., number of states) is *quadratic* in the number of states and linear in the number of pushdown symbols of M. Then, we show that this size bound cannot be improved in general by pointing out its asymptotical optimality. In Section 4, we deal with restricted versions of PDA, namely: PDA which never pop, stateless PDA, and counter machines. For any of these restrictions, we present optimal NFA for pushdown store languages, which are strictly smaller than the NFA given for the general case. Finally, in Section 5, we apply these results to the analysis of the hardness of some decision

[⋆] Partially supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models."

[⋆⋆] An enlarged version of this work has been accepted at the *14th Int. Workshop on Descriptional Complexity of Formal Systems (DCFS) 2012*, and will appear in the Springer LNCS series.

problems related to PDA. We show that the questions of whether $P(M)$: (i) is a finite set, or (ii) is a finite set of words having at most length k, for a given $k \geq 1$, or (iii) is unary, can be answered in deterministic polynomial time. Moreover, we also prove the P-completeness of these questions. As an application, we obtain that it is P-complete to decide whether a given unambiguous PDA is a constant height PDA [2, 3], or is a PDA of constant height k, for a given $k \geq 1$, or to decide whether a given PDA is essentially a counter machine. Due to lack of space, some proof details are moved to the Appendix.

2 Preliminaries

A *pushdown automaton* (PDA, see e.g., [6]) is formally defined to be a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite pushdown alphabet, δ is the transition function mapping[3] $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is a particular pushdown symbol, called the bottom-of-pushdown symbol, initially appearing on the pushdown store, and $F \subseteq Q$ is a set of accepting (or final) states. Roughly speaking, a *nondeterministic finite automaton* (NFA) is a PDA where the pushdown store is never used. A *configuration* of M is a triple (q, w, γ), where q is the current state, w the unread part of the input, and γ the current content of the pushdown store, the *leftmost* symbol of γ being the *top* symbol. For $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $\gamma, \beta \in \Gamma^*$, and $Z \in \Gamma$, we write $(q, aw, Z\gamma) \vdash (p, w, \beta\gamma)$ if $(p, \beta) \in \delta(q, a, Z)$. The reflexive transitive closure of $\vdash$ is denoted by $\vdash^*$. The language accepted by M by *accepting states* is the set $L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (f, \lambda, \gamma),\ \text{for some } f \in F \text{ and } \gamma \in \Gamma^*\}$.

The *pushdown store language* of M (see, e.g., [1, 4]) is the set $P(M)$ of all words occurring on the pushdown store along accepting computations of M:

$$\begin{aligned} P(M) = \{u \in \Gamma^* \mid \exists x, y \in \Sigma^*,\ q \in Q,\ f \in F :& \\ (q_0, xy, Z_0) \vdash^* (q, y, u) \vdash^* (f, \lambda, \gamma),\ &\text{for some } \gamma \in \Gamma^*\}. \end{aligned}$$

Throughout the rest of the paper, we assume PDA to be in *normal* form, i.e., they can push at most two symbols at each move.

3 Pushdown Store Languages: the General Case

Already in [4], it is proved that $P(M)$ is regular. Here, inspired by [1], we construct an optimal size NFA for $P(M)$ as:

Theorem 1. *Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ be a PDA. Then, $P(M)$ is accepted by an NFA with $|Q|^2(|\Gamma| + 1) + |Q|(2|\Gamma| + 3) + 2$ states. Moreover, there exist infinitely many PDA $M_{Q,\Gamma}$ such that every NFA accepting $P(M_{Q,\Gamma})$ needs $\Omega(|Q|^2|\Gamma|)$ states.*

[3] The empty word is here denoted by λ.

Proof. (outline, see Appendix) We define the set $Acc(Q)$ (resp., $Co\text{-}Acc(Q)$) representing all the pushdown contents reachable from the initial configuration (resp., from which a final state can be reached). We let $[Q] = \{[q] \mid q \in Q\}$, and define:

$$Acc(Q) = \{[q]u \in [Q]\Gamma^* \mid \exists x, y \in \Sigma^* : (q_0, xy, Z_0) \vdash^* (q, y, u)\},$$
$$Co\text{-}Acc(Q) = \{[q]u \in [Q]\Gamma^* \mid \exists y \in \Sigma^*,\ f \in F,\ u' \in \Gamma^* : (q, y, u) \vdash^* (f, \lambda, u')\}.$$

We get[4] $P(M) = [Q]^{-1}(Acc(Q) \cap Co\text{-}Acc(Q))$. A left-linear (resp., right-linear) grammar for $Acc(Q)$ (resp., $Co\text{-}Acc(Q)$) can be built, and turned into an equivalent NFA with $|Q| \cdot (|\Gamma| + 1) + 1$ (resp., $|Q| + 2$) states. From these two NFA, an NFA for $P(M)$ with $|Q|^2(|\Gamma| + 1) + |Q|(2|\Gamma| + 3) + 2$ states is built. □

4 Pushdown Store Languages for Special Cases

For restricted models of PDA, we are able to provide NFA for their pushdown store languages whose size is strictly below the general upper bound in Theorem 1. Namely, we focus on: PDA *which never pop* a symbol from the pushdown, *stateless* PDA (i.e., with a single state [6]), and *counter machines* (i.e., PDA with pushdown alphabets having a single symbol Z beside Z_0 [6]):

Theorem 2. *Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ be a PDA of type displayed in the first column of Table 1. Then, an NFA for $P(M)$ can be built, whose number of states is bounded as in the second column. These size bounds are optimal.*

PDA type	**Size of NFA for $P(M)$**
never popping	$\lvert Q\rvert \cdot \lvert\Gamma\rvert + 1$
stateless	$\lvert\Gamma\rvert + 1$
counter machine	$\lvert Q\rvert + 2$

Table 1. Size of NFA accepting pushdown store languages of restricted PDA.

Proof. (outline, see Appendix) For never popping PDA and stateless PDA, the grammars for $Acc(Q)$ and $Co\text{-}Acc(Q)$ in Theorem 1 can be "optimized". For counter machines, by pigeonhole arguments on possible pushdown contents, we prove that $P(M)$ can be only of the form Z^*Z_0 or Z^hZ_0, with $h \leq |Q|$. In all three cases, PDA witnessing optimality can be exhibited. □

5 Computational Complexity of Decidability Questions

The complexity of deciding some properties of $P(M)$ for a given PDA M, namely, finiteness and being subset of Z^*Z_0, can be answered by first constructing the NFA N for $P(M)$ and then deciding finiteness or inclusion in Z^*Z_0 for $L(N)$, respectively. For the first step, we get (see Appendix):

Theorem 3. *Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ be a PDA. Then, an NFA for $P(M)$ can be constructed in deterministic polynomial time.*

[4] Given $A, B \subseteq \Sigma^*$, we let $A^{-1}B = \{y \in \Sigma^* \mid \exists x \in A : xy \in B\}$.

This leads to P-completeness of the following decision problems:

Theorem 4. *Given a PDA M, it is* P*-complete to decide whether $P(M)$: (i) is a finite set, (ii) is a finite set of words having at most length k, for a given $k \geq 1$, (iii) is a subset of $Z^* Z_0$.*

Proof. (outline, see Appendix) We consider only point (i). The problem belongs to P: by Theorem 3, an NFA N for $P(M)$ is built in polynomial time. Then, the infiniteness of $L(N)$ can be decided in NLOGSPACE $\subseteq$ P [8], which is closed under complementation [7, 11]. Hence, the finiteness of $L(N)$ can be decided in NLOGSPACE as well. For completeness, we log-space reduce the emptiness problem for context-free grammars, which is known to be P-complete [9]. $\square$

As a consequence, we get the P-completeness of deciding whether a PDA is of a certain "nature". More precisely, a PDA M is of *constant height* if there is a constant $k \geq 1$ such that, for any word in $L(M)$, there exists an accepting computation along which the pushdown store never contains more than k symbols [2, 3]. M is *essentially* a counter machine [6] if in all of its accepting computations the pushdown storage is used as a counter. By Theorem 4, we get

Corollary 5. *For an unambiguous PDA M, it is* P*-complete to decide whether M: (i) is of constant height, (ii) is of constant height k, for a given $k \geq 1$. If M is a PDA, it is* P*-complete to decide whether it is essentially a counter machine.*

References

1. Autebert, J.-M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Handbook of Formal Languages, Vol. 1. pp. 111–174. Springer (1997)
2. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: The size-cost of Boolean operations on constant height deterministic pushdown automata. In: DCFS 2011. LNCS 6808, pp. 80–92. Springer (2011)
3. Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. Comput. 208, 385–394 (2010)
4. Greibach, S.A.: A note on pushdown store automata and regular systems. Proc. Amer. Math. Soc. 18, 263–268 (1967)
5. Hartmanis, J.: Context-free languages and Turing machines computations. Proc. Symposium on Applied Mathematics 19, 42–51 (1967)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)
7. Immerman, N.: Nondeterministic space is closed under complementation. SIAM J. Comput. 17, 935–38 (1988)
8. Jones, N.D.: Space-bounded reducibility among combinatorial questions. J. Comput. System. Sci. 11, 68–85 (1975)
9. Jones, N.D., Laaser, W.T.: Complete problems for deterministic polynomial time. Theoretical Computer Science 3, 105–118 (1976).
10. Mereghetti, C., Palano, B.: Quantum finite automata with control language. Theoretical Informatics and Applications 40, 315–332 (2006)
11. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Inform. 26, 279–84 (1988)

Undecidability of Quantized State Feedback Control for Discrete Time Linear Hybrid Systems

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci
Computer Science Department, Sapienza University of Rome, Italy

Abstract. We show that existence of a quantized controller for a given *Discrete Time Linear Hybrid System* (DTLHS) is undecidable. This is a relevant class of controllers since *control software* always implements a quantized controller.

Introduction Many embedded systems are software based control systems. A software based control system consists of two main subsystems: the *controller* and the *plant*. Typically, the plant is a physical system consisting, for example, of mechanical or electrical devices while the controller consists of *control software* running on a microcontroller. In an endless loop, each T seconds (sampling time), the controller, after an *Analog-to-Digital* (AD) conversion (*quantization*), reads sensor outputs from the plant and, possibly after a *Digital-to-Analog* (DA) conversion, sends commands to plant actuators. The controller selects commands in order to guarantee that the closed loop system (that is, the system consisting of both plant and controller) meets given safety and liveness properties, i.e. system level specifications. Formal verification of system level specifications for software based control systems requires modelling both continuous systems (typically, the plant) as well as discrete systems (the controller). This is typically done using *Hybrid Systems* (e.g., see [2, 1]). In [7], we presented a constructive necessary condition and a constructive sufficient condition for the existence of a (*quantized sampling*) controller for a software based control system when the plant is modelled using a *Discrete Time Linear Hybrid System* (DTLHS), that is a discrete time hybrid system whose dynamics is defined as a linear predicate (i.e., a boolean combination of linear constraints) on its continuous as well as discrete variables. System level safety as well as liveness specifications may be modelled as set of states defined in turn as linear predicates. From [5], we know that existence of a sampling controller, even for relatively simple linear hybrid automata, is undecidable. Considering that, given a *quantization schema* (i.e. number of bits used in AD conversion), the number of quantized sampling controllers is finite and that when using DTLHSs also the plant is modelled using a discrete model of time, one may be led to think that existence of a quantized sampling controller might be decidable. In this paper we show that also for DTLHSs, existence of a quantized sampling controller meeting given specifications is undecidable. We prove such a result by showing that any two-counter machine can be coded as a DTLHS thereby extending to DTLHSs the proof technique in [5]. Undecidability results of the control synthesis problem for dense as well as discrete time linear hybrid systems have been presented in [6, 5, 9, 3]. A more general problem is considered in [4], namely the discrete time control with unknown sampling rate, that is undecidable even for TA. Moreover, we note that none of the above papers addresses the issue of quantized control.

Labeled Transition Systems An LTS $\mathcal{S}$ is a tuple (S, A, T) where S is a set of *states*, A is a set of *actions*, and $T : S \times A \times S \to \mathbb{B}$ is the *transition relation* of $\mathcal{S}$. $\mathcal{S}$ is *deterministic* if $\forall s \in S, a, a', a'' \in A$, $T(s, a, s')$ and $T(s, a, s'')$ imply $s' = s''$. A *run* or *path* for an LTS $\mathcal{S}$ is a sequence $\pi = s_0, a_0, s_1, a_1, \ldots$ of states s_t and actions a_t such that $\forall t \geq 0$ $T(s_t, a_t, s_{t+1})$. The length $|\pi|$ of a finite run π is the number of actions in π. $\pi^{(S)}(t)$ denotes the t-th state element of π, and $\pi^{(A)}(t)$ the t-th action element of π.

Definition 1. *A* reachability problem *is a triple $(\mathcal{S}, I, G)$, where $\mathcal{S}$ is an LTS (S, A, T), and $I, G \subseteq S$. G is* reachable *from I if there exists a run π of $\mathcal{S}$ such that $\pi^{(S)}(0) \in I$ and $\pi^{(S)}(t) \in G$ for some $t \in \mathbb{N}$.*

LTS Control Problem A *controller* for an LTS $\mathcal{S}$ is used to restrict the dynamics of $\mathcal{S}$ so that all states in the initial region will reach in one or more steps the goal region. In what follows, let $\mathcal{S} = (S, A, T)$ be an LTS, I, $G \subseteq S$ be, respectively, the *initial* and *goal* regions of $\mathcal{S}$.

Definition 2. *A* controller *for $\mathcal{S}$ is a function $K : S \times A \to \mathbb{B}$ s. t. $\forall s \in S$, $\forall a \in A$. $K(s, a) \to \exists s'\ T(s, a, s')$. The* domain *of K is the set of states for which a control action is enabled, i.e.* $\mathrm{dom}(K) = \{s \in S \mid \exists a\, K(s, a)\}$. *The* closed loop system *$\mathcal{S}^{(K)}$ is the LTS $(S, A, T^{(K)})$, where $T^{(K)}$ $(s, a, s') = T(s, a, s') \wedge K(s, a)$.*

A path π is a *fullpath* if either it is infinite or its last state has no successors. We denote with $\mathrm{Path}(s, a)$ the set of fullpaths starting in state s with action a. Given a path π in $\mathcal{S}$, we define $j(\mathcal{S}, \pi, G)$ as follows. If there exists $n > 0$ such that $\pi^{(S)}(n) \in G$, then $j(\mathcal{S}, \pi, G) = \min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\}$. Otherwise, $j(\mathcal{S}, \pi, G) = +\infty$. We require $n > 0$ since our systems are nonterminating and each controllable state (including a goal state) must have a path of positive length to a goal state. Taking $\sup \varnothing = +\infty$ and $\inf \varnothing = +\infty$, the *worst case distance* (pessimistic view) of a state s from the goal region G is $J_s(\mathcal{S}, G, s) = \sup\{j_s(\mathcal{S}, G, s, a) \mid a \in \mathrm{Adm}(\mathcal{S}, s)\}$, being $j_s(\mathcal{S}, G, s, a) = \sup\{j(\mathcal{S}, G, \pi) \mid \pi \in \mathrm{Path}(s, a)\}$. The *best case distance* (optimistic view) of a state s from the goal region G is $J_w(\mathcal{S}, G, s) = \sup\{j_w(\mathcal{S}, G, s, a) \mid a \in A\}$, being $j_w(\mathcal{S}, G, s, a) = \inf\{j(\mathcal{S}, G, \pi) \mid \pi \in \mathrm{Path}(s, a)\}$.

Definition 3. *A* control problem *for $\mathcal{S}$ is a triple $\mathcal{P} = (\mathcal{S}, I, G)$. A* strong *(resp.* weak*) solution to $\mathcal{P}$ is a controller K for $\mathcal{S}$, such that $I \subseteq \mathrm{dom}(K)$ and for all $s \in \mathrm{dom}(K)$, $J_s(\mathcal{S}^{(K)}, G, s)$ (resp. $J_w(\mathcal{S}^{(K)}, G, s)$) is finite.*

Proposition 1. *Each strong solution of a control problem $(\mathcal{S}, I, G)$ is also a weak solution. If $\mathcal{S}$ is deterministic, any weak solution is also a strong solution.*

Discrete Time Linear Hybrid Systems A *Discrete Time Linear Hybrid System* (DTLHS) $\mathcal{H}$ is a tuple (X, U, Y, N) where: $X = X^r \cup X^d$ is a finite sequence of real (X^r) and discrete (X^d) *present state* variables. We denote with X' the sequence of *next state* variables obtained by decorating with $'$ all variables in X. $U = U^r \cup U^d$ is a finite sequence of *input* variables. $Y = Y^r \cup Y^d$ is a finite sequence of *auxiliary* variables. Auxiliary variables are typically used to model *modes* or *uncontrollable inputs*. $N(X, U, Y, X')$ is a linear predicate over $X \cup U \cup Y \cup X'$ defining the *transition relation* (*next state*) of the system. The dynamics of a DTLHS $\mathcal{H}$ is defined by the labeled transition system $\mathrm{LTS}(\mathcal{H}) =$

$(\mathcal{D}_X,\ \mathcal{D}_U,\ \bar{N})$ where: $\bar{N} : \mathcal{D}_X \times \mathcal{D}_U \times \mathcal{D}_X \to \mathbb{B}$ is a function s.t. $\bar{N}(x, u, x') = \exists\, y \in \mathcal{D}_Y\ N(x, u, y, x')$. A *state* x for $\mathcal{H}$ is a state x for LTS($\mathcal{H}$) and a *path* for $\mathcal{H}$ is a path for LTS($\mathcal{H}$).

Definition 4. *Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS and let I and G be linear predicates over X. The* DTLHS reachability problem *$\mathcal{R} = (\mathcal{H}, I, G)$ is defined as the LTS reachability problem* (LTS($\mathcal{H}$)*, I, G). Similarly, the* DTLHS control problem *$(\mathcal{H}, I, G)$ is defined as the LTS control problem* (LTS($\mathcal{H}$)*, I, G).*

Quantized Control Problem Quantization is the process of approximating a continuous interval by a set of integer values. A *quantization function* $\gamma : \mathbb{R} \mapsto \mathbb{Z}$ is a non-decreasing function, such that for any bounded interval $I = [a, b] \subset \mathbb{R}$, $\gamma(I)$ is a bounded integer interval. Given a DTLHS $\mathcal{H} = (X, U, Y, N)$ a *quantization* $\mathcal{Q}$ for $\mathcal{H}$ is a pair $(\mathcal{A}, \Gamma)$, where (let $W = X \cup U \cup Y$): $\mathcal{A}$ is a predicate over W that explicitely bounds each variable in W. For each $w \in W$, we denote with $\mathcal{A}_w$ its *admissible region* and with $\mathcal{A}_W = \prod_{w \in W} \mathcal{A}_w$ and Γ is a set of maps $\{\gamma_w \mid w \in W$ and γ_w is a quantization function$\}$. Let $W = [w_1, \ldots, w_k]$ and $v = [v_1, \ldots, v_k] \in \mathcal{A}_W$. We write $\Gamma(v)$ for the tuple $[\gamma_{w_1}(v_1), \ldots, \gamma_{w_k}(v_k)]$.

A control problem admits a *quantized* solution if control decisions can be made by just looking at quantized values. This enables a software implementation for a controller.

Definition 5. *Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS, $\mathcal{Q} = (\mathcal{A}, \Gamma)$ be a quantization for $\mathcal{H}$ and $\mathcal{P} = (\mathcal{H}, I, G)$ be a DTLHS control problem. A $\mathcal{Q}$* Quantized Feedback Control *(QFC) strong (resp. weak) solution to $\mathcal{P}$ is a strong (resp. weak) solution $K(x, u)$ to $\mathcal{P}$ such that $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ where $\hat{K} : \Gamma(\mathcal{A}_X) \times \Gamma(\mathcal{A}_U) \to \mathbb{B}$.*

Undecidability of Quantized Feedback Control Problem We prove the undecidability of the DTLHS quantized feedback control problem along the same lines of similar undecidability proofs [6, 5]. We first show that a two-counter machine M can be encoded as a deterministic DTLHS $\mathcal{H}_M$ without controllable actions in such a way that M halts if and only if $\mathcal{H}_M$ reaches a goal region. This immediately implies that DTLHS reachability is undecidable. Since $\mathcal{H}_M$ has no controllable inputs, existence of a weak controller is equivalent to a reachability problem. For the same reason, actions enabled by any controller for $\mathcal{H}_M$ do not depend on state variables. As a consequence, a quantized weak control problem is equivalent to a DTLHS control problem. Finally, by Proposition 1, weak solutions to deterministic LTS control problems are also strong solutions. Therefore, since $\mathcal{H}_M$ is deterministic, the quantized strong control problem for DTLHS is undecidable, too.

Two-Counter Machines. A *two-counter machine* [8] M consists of two counters that store unbounded natural numbers and a finite control that is a finite sequence of statements $\langle 1 : stmt_1, \ldots, n : stmt_n \rangle$, where $stmt ::= \mathsf{inc}\ i\ k \mid \mathsf{dec}\ i\ k \mid \mathsf{beq}\ i\ k \mid \mathsf{halt}$, with $i \in \{0, 1\}$. As an example, if the counter i is 0, the execution of $j : \mathsf{beq}\ i\ k$ causes a jump to the statement labeled k, otherwise the execution proceed with statement $j + 1$. The halting problem for two-counter machine is undecidable [8].

Lemma 1. *For any two-counter machine M, there exists a* bounded *and* deterministic *DTLHS $\mathcal{H}_M$, and two predicates I and G such that M halts if and only if G is reachable from I in $\mathcal{H}_M$.*

Proof. (Sketch) Let M be a two-counter machine and let $\mathcal{H}_M$ be the DTLHS (X, U, Y, N), where $X^r = \{x_0, x_1\}$, $X^d = \{l, g\}$, and $U = Y = \varnothing$. We use two real variables x_0 and x_1 to encode values stored in counters. Each natural number m is encoded by the rational number $1/2^m$. A discrete variable l stores the label of the statement currently under execution. Finally, the boolean variable g encodes termination of the computation of M. The transition relation N encodes the execution of the control program. A program $\langle 1 : stmt_1, \ldots, n : stmt_n \rangle$ is encoded by the predicate $N = \bigwedge_{j=1}^{n} [\![j : stmt_j]\!]$. As an example, here we present the encoding of the beq statement: $[\![j : \mathtt{beq}\ i\ k]\!] \equiv (l \neq j) \vee (((x_i \neq 1) \vee (l' = k)) \wedge ((x_i = 1) \vee (l' = l + 1)) \wedge (x_{1-i} = x'_{1-i}) \wedge (g = g'))$.

An immediate consequence of Lemma 1 is the undecidability of the DTLHS reachability problem.

Theorem 1. *The reachability problem for bounded DTLHSs is undecidable. Existence of strong and weak solutions to a control problem for a bounded DTLHS is undecidable. Existence of QFC strong and weak solutions to a DTLHS control problem is undecidable.*

Conclusions We have shown that, for DTLHSs, existence of a quantized sampling controller meeting given (safety and liveness) system level specifications is undecidable. The relevance of such a problem stems from the fact that the *control software* implementing the controller in a software based control system always yields a quantized sampling controller. Investigating interesting classes of (discrete time) hybrid systems for which quantized sampling control is decidable appears to be an interesting future work.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3 – 34 (1995)
2. Alur, R.: Timed automata. In: CAV. pp. 8–22. LNCS 1633 (1999)
3. Asarin, E., Bouajjani, A.: Perturbed turing machines and hybrid systems. In: LICS. pp. 269–278 (2001)
4. Cassez, F., Henzinger, T.A., Raskin, J.F.: A comparison of control problems for timed and hybrid systems. In: HSCC. pp. 134–148 (2002)
5. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. In: ICALP. pp. 582–593 (1997)
6. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. of Computer and System Sciences 57(1), 94–124 (1998)
7. Mari, F., Melatti, I., Salvo, I., Tronci, E.: Synthesis of quantized feedback control software for discrete time linear hybrid systems. In: CAV. pp. 180–195. LNCS 6174 (2010)
8. Minsky, M.L.: Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. The Annals of Mathematics 74(3), pp. 437–455 (1961)
9. Vidal, R., Schaffert, S., Shakernia, O., Lygeros, J., Sastry, S.: Decidable and semidecidable controller synthesis for classes of discrete time hybrid systems. In: CDC. pp. 1243–1248. IEEE Computer Society (2001)

Exploiting Fine Grained Parallelism on the SPE[*]

Emanuele Milani and Nicola Zago

Department of Information Engineering,
University of Padova, Padova, ITALY
{milaniem,zagonico}@dei.unipd.it

Abstract. In this paper we propose a simulation of Work-Time framework algorithms on the recently proposed Speculative Prefetcher and Evaluator (SPE) processor, using a pipelined hierarchical memory. This allows us to inherit the efficency of work-optimal parallel algorithms in this new model.

Keywords: computational models, hierarchical pipelined memory, work-time simulation, efficient merge

1 Introduction

The *Random Access Machine* (RAM) is an idealized sequential computational model, in which the time to access any memory location is independent from memory size [8]. This assumption is unsuitable for physical machines, because of the principles of maximum information density and maximum information speed [4]. Indeed their combination imposes a minimum access latency, which grows with the size of the memory. RAM complexity is therefore an automatic lower bound for a given problem on any real sequential machine.

One of the main aims of recent literature is to devise implementable machine designs which hide or limit the latency impact, matching the ideal lower bounds. To this purpose, two major algorithmic strategies have been investigated: *locality* and *concurrency* of memory accesses. At the same time, models suitable to exhibit and measure them have been devised, respectively by means of hierarchical memories and pipelined memories.

Among the first we recall the Hierarchical Memory Model (HMM) [1], which is characterized by a non-decreasing function $a(x)$ that describes the access time to location x, implying that locations near to the processor take a lower time to be accessed. The Block Transfer (BT) [2] model extends the HMM, allowing the transfer of B adjacent locations starting from address x in $a(x) + B$ steps. These models encourage the design of algorithms exhibiting *temporal locality*, that is to use more often memory locations with a lower address, since they have a lower access time. The BT considers also *spatial locality*, or the access

[*] This work was supported, in part, by MIUR-PRIN Project *AlgoDEEP*, by PAT-INFN Project *AuroraScience*, and by the University of Padova Projects *STPD08JA32* and *CPDA099949*.

of adjacent data in a short time window. These models are good simplification of actual computer memories, which are hierarchically divided in several levels – from fast yet small caches to slow yet huge mass storage – among which data is transferred in blocks. A special case of BT is the *Disk Model* (DM) [15], which has been used to model the disk bottleneck and study the disk I/O efficiency of algorithms. Although general simulations of RAM algorithms in these models yield a worst case slowdown proportional to the memory latency, algorithms exhibiting locality can reach the same performance, as in *matrix multiplication* [1]. Nevertheless, algorithms which need to read the whole input present superlinear lower bounds; for example the *touch problem* – which consists in accessing each of the n elements in input and triavially solvable in n step in RAM model – has $\Theta(n) = n \log^* n$ complexity in the BT model with access function $a(x) = \log x$ [2].

On the other hand, *Pipelined Memories* (PM) [14] allow latency hiding through overlap of accesses. In particular they can perform k independent requests to the memory of M locations waiting only $O(a(M)+k)$ step for receiving all the responses. One should note that, unlike BT, accesses need not involve adjacent locations, nevertheless we have to know in advance enough independent requests to amortize the latency cost. PMs can solve the touch problem in linear time, still they have superlinear performance in problems where there are strong dependencies among instructions.

Recently [3] introduced a pipelined and hierarchical memory design which complies with physical constraints. This, jointly with the SPE processor, forms the Pipelined Hierarchical Memory Machine (PHMM), which is able to match RAM complexity ($O(1)$ slowdown) on wide classes of programs, exploiting both concurrency and locality.

Before memory models, these strategies were already been extensively studied in parallel computing, since concurrency allows independent executions among the processors and locality limits communication among these. This fact is pointed out by several works which show how to effectively simulate parallel models in memory models, partially carrying the knowledge of parallel computing in this field. For example, in [6] and [14], general Parallel Random Access Machine (PRAM, the ideal parallel model) simulations are proposed respectively on DM and PM, deriving new upper bounds for some problems on these memory models exploiting previously known parallel results. In [9] is shown how to turn the submachine locality of the Decomposable Bulk Synchronous Parallel model (D-BSP, a parallel model where also communication and synchronization costs are considered) in locality of references for the HMM.

Our paper is related to these works; in fact we propose a simulation of Work-Time (WT) framework [12] – a parallel framework which highlights parallelism and critical path of parallel algorithms – on the PHMM. This simulation, when applied to work-optimal WT algorithms, provides optimal algorithms for the PHMM. In particular we use it to obtain an optimal merge implementation, improving the previously best known result, which has superlinear complexity.

We also show how the whole exploitation of available parallelism can lead to a simulation with a huge memory footprint and degrading its performance.

The paper is organized as follows: in Sections 2 and 3 we recall concepts about SPE and parallelism. Section 4 contains our simulation, whose applications are shown in Sect. 5. Conclusions and further research directions are in Sect. 6.

2 The Speculative Prefetcher and Evaluator

The Speculative Prefetcher and Evaluator is a processor design which is able to exploit a *Pipelined Hierarchical Memory* (PHM) while complying with the physical constraints discussed in [4]. Both have been introduced in [3].

The memory features a size M and latency access function $a(x)$. It can accept a request per cycle for an arbitrary location x, guaranteeing a response within $a(x)$ cycles. One should note that the latency of k requests is determined by the location with higher address accessed, unlike PMs where latency is always $a(M)$.

SPE has an Instruction Generator Unit (IGU), connected to an instruction PHM, and an Instruction Execution Unit (IEU), connected to a data PHM of M locations. Both IGU and IEU have $O(k)$ constant–sized units called *stations* and arranged as linear arrays. Parameter k denotes the *processor size* and is choosen to match the worst case latency $a(M)$.

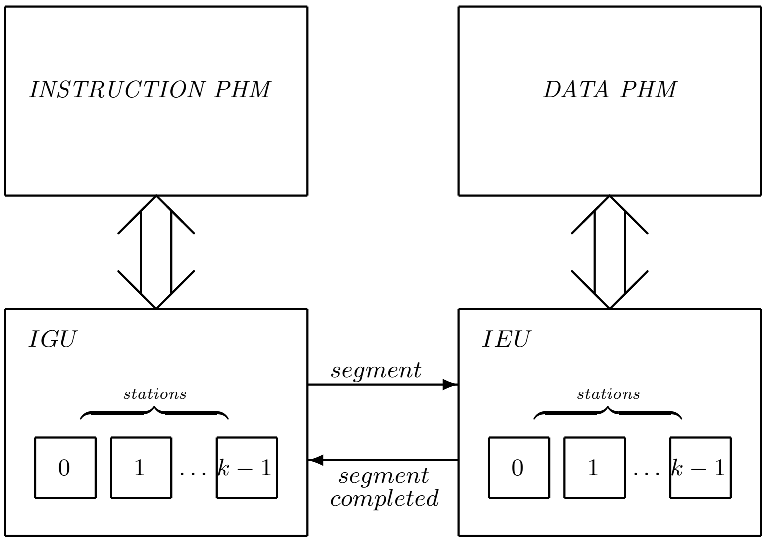


Fig. 1. Scheme of the PHMM architecture: IGU reads instructions from the *instruction PHM*, produces a *segment* and passes it to IEU stations. Here data are speculatively prefetched and instructions are executed, solving possible invalid data with the internal forwarding or reading them from *data PHM*.

The computation consists in a sequence of *stages*. In each stage the IGU reads instructions from the instructions PHM, assembling a *segment* (the number of stations used in the IEU) of machine code instructions for the IEU. In particular IGU predicts possible branches of the code and then loads the obtained sequential segment in the IEU, one instruction per station. The execution consists in a series of *rounds* in which instructions are executed. Each round is divided in two parts: in the first the IEU speculatively prefetches the operands of all the instructions of the segment, both directly and indirectly addressed; in the second, all the stations execute sequentially with the speculative operands they hold. In case of mispredicted operands, due for example to indirectly addressed operands whose base address has been changed by previous stations, another round takes place.

Anyway, some memory accesses are avoided through a mechanism of internal forwarding among stations. This mechanism allows the stations to forward their computed result along the linear array connecting them. In this way, if the value of an operand is modified by a station, making invalid the prefetched values of the followings stations, the new value will be immediately available to them without new memory accesses, but simply receiving it from the linear array.

One should note that the load of $a(M)$ instructions per round gives us an amortized $O(1)$ set up time per station. Moreover the segment size can be dinamically decreased to take advantage from the hierarchical structure of the memory. When the speculative prefetch or the internal forwarding succeed in avoiding further memory accesses, also each instruction execution takes $O(1)$ step.

In order to understand the segment assembly problems, we recall the definition of dependency-related concepts.

Definition 1. *There is* Functional Dependency *(FD) between instructions I_j and I_k if I_j modifies $m[x]$, and I_k uses the content of $m[x]$ as operand or to indirectly address the output location, while no operation among I_j and I_k modifies $m[x]$.*

Definition 2. *There is* Address Dependency *(AD) between instructions I_j and I_k if I_j modifies $m[x]$, and I_k uses the content of $m[x]$ to indirectly access to an operand, while no operation among I_j and I_k modifies $m[x]$.*

Clearly, memory accesses due to FD can be avoided in SPE thanks to internal forwarding. Instead AD is harder to be addressed and requires a new fetch of operands.

Definition 3. *Given the instruction stream $(I_1, I_2, \dots, I_N)$ produced by the execution of a SPE program $\mathcal{P}$ on a particular input, its* address dependence depth *D is the maximum length of a subsequence $I_{j_1}, I_{j_2}, \dots, I_{j_L}$ with $j_1 < j_2 < \cdots < j_L$ where subsequent instructions have address dependency.*

Let us recall two important program categories.

Definition 4. *A program $\mathscr{P}$ is* straigth–line *if it consists of only data processing instructions.*

Definition 5. *A program $\mathscr{P}$ is* direct–flow *if it is straigth–line and does not use indirect addressing.*

In [3] it is shown that any N instructions straigth–line program with address dependence depth D and accessing memory locations with address smaller than M can be executed by an SPE with PHM in time $T = O((D+1)(N + a(M)))$. Moreover, a direct–flow program can be executed in $T = O(N + a(M))$.

In particular the following result from [3] holds:

Proposition 1. *A program consisting in nested for loops where the only branches are those related to the cycles, can be executed in $T = O(D(N + a(M)))$.*

Proposition 1 applies to wide classes of programs, such as FFT and Matrix Multiplication.

We quote the following example, which intuitively shows how this occurs.

Example 1. Let's consider the execution of a C-like code that increments every element of an array: `for i=1 to k; A[i] = A[i]+1`. Using the naive branch prediction policy that always reenters the loop, the IGU can unroll the loop in `i=1; m[i]=m[i]+1; i=i+1; m[i]=m[i]+1; ...`. At this point, the SPE speculatively calculates all the i values in the first round, prefetching the right operands at the beginning of the second one, whose speculative execution correctly completes the segment execution. So it can resolve address dependencies in O(1) amortized time.

3 Parallel Computing Background

Definition 6. *A* Parallel Random Access Machine *(PRAM) [10, 11] is an abstract parallel machine model, that consists in a collection of P synchronous processors and M shared memory locations.*

Definition 7. *A* PRAM program *is a sequence of parallel steps, each of which specifies an instruction per processor.*

Beside the number of nodes, the computational power of a PRAM is determined by which shared memory operations are permitted. Within a step, in fact, each memory location may or may not be accessed by more than one processor. In other words, a PRAM can be provided with either an *exclusive read* (resp. *exclusive write*) memory, or a *concurrent read* (resp. *concurrent write*) memory. Moreover, when concurrent writes are allowed, a contention policy must be specified in order to determine the actual memory state after the access. The most studied configurations, in order of increasing power, are exclusive read exclusive write (EREW), concurrent read exclusive write (CREW), concurrent read concurrent write (CRCW).

Both SIMD and MIMD versions have been studied, anyway they are equivalent [7] if they feature the same memory access policy.

Definition 8. *The* Work-Time model *(WT) [12] is a parallel programming model in which an algorithm $\mathscr{A}_{WT}$ consists in an ordered sequence of T sets $s_0, \ldots, s_{T-1}$ of independent operations on M_{WT} memory locations. Different sets may differ in size and therefore exhibit more or less parallelism. Let $|s_i| = p_i$, then we define the* work W *of $\mathscr{A}_{WT}$ as $W = \sum_{i=0}^{T-1} p_i$.*

It should be noted that, since it is always possible to simulate $\mathscr{A}_{WT}$ on a RAM in time $T_{RAM} = W$, lower bounds on RAM complexity automatically hold also for the work. In particular, let T^*_{RAM} be the best RAM complexity for a given problem. Then, the equivalent WT algorithm $\mathscr{A}_{WT}$ is *work–optimal* if and only if W is $O(T^*_{RAM})$.

WT algorithms are meant to be executed by PRAMs, by means of a schedule. A sufficient condition for a valid schedule of $\mathscr{A}_{WT}$ in a PRAM is that each operation in s_i is executed after all operations in s_{i-1} and before any operation in s_{i+1}. This schedule allows us to apply *Brent's Theorem* [5] and to execute $\mathscr{A}_{WT}$ in a PRAM with P processors in a time $O(\frac{W}{P} + T)$.

On the other hand, it is not clear how we can reschedule a PRAM program for P processors as the processor number increases, since possible dependencies between steps are not stated explicitly. For this reason WT framework is much more convenient if we need to extract dependencies and available parallelism.

The major construct of the WT model is the `pardo`, which specifies a parallel step with a syntax similar to a traditional `for`. The main difference is that the cycle index denotes just the index of an element of the set of instructions and can not be modified by the instructions. For example

```
for $j, 1 \leq j \leq p$ pardo
    operation$_j$
```

denotes a set of p independent operations, whose execution order is irrelevant.

Since WT framework is a very high level model, it is important to pay attention to some hidden low level details. In particular, one should note that in each parallel step:

F1 guarantees that all the reads take place before any write;
F2 allows addresses to be expressed in a high level fashion.

Therefore any simulation or implementation of an algorithm which relies on such features has to provide them. For the sake of clarity we avoid to explicitly address these issues by resorting to a slightly more constrained yet equivalent case. We then show how to map the general case to this.

Let us introduce a class of WT algorithms.

Definition 9. *A step of WT algorithm $\mathscr{A}_{WT}$ is* CRCW decoupled *if any concurrently accessed memory location is either read or written. $\mathscr{A}_{WT}$ is itself* CRCW decoupled *if this condition holds for each step.*

Any CRCW decoupled algorithm does not rely on (F1). In the opposite case, it is possible to devise an equivalent CRCW decoupled algorithm with the same work and time complexity. In fact, it suffices to split each parallel step into two sub–steps. The first fills an auxiliary array with the operation inputs, while the second performs the actual execution, reading from the array. In the worst case, the memory overhead is $O(p)$.

Consider now, without loss of generality a SIMD parallel step in the WT framework, which executes on an initial memory state $\mathscr{M}_i$ and leads to final state $\mathscr{M}_{i+1}$. Available parallelism and the memory locations that must be read or written (both usually parametrized with the size of the input) are indicated by a `pardo` statement. In particular an index j is used to distinguish each concurrent operation. Note that the memory to store one step is constant and therefore the whole program takes $O(T)$ memory.

One can think the operands of operation j to be a function of j. Typically, such function is simple enough to be expressed by the addressing modes of a modern instruction set (for example a base address and an index–dependent offset). In the most general case, when (F2) is fully exploited, the function can be explicitly used to prepare an auxiliary operand vector. This preliminary phase can be implemented with a memory overhead depending on how much parallelism we want exploit (up to p_i). In particular for the SPE, $O(k)$ memory locations are sufficient.

4 Simulation of WT Algorithms

One way to write efficient programs for SPE is to exploit the parallelism of Work–Time algorithms. Parallel steps can be efficiently coded into programs, also in case of concurrent memory accesses, which can be implemented with little effort. In fact, the address dependence depth of the resulting segments is $O(1)$, and memory accesses can be fully pipelined.

However, it must be noted that a trivial static unrolling of a `pardo` statement could lead to an SPE program with $O(W)$ size. In this case, instruction fetch latencies could be larger that data latencies, thus hindering time efficiency.

A more complicated unrolling, proportional to processor size, can be devised, which leads to programs with $O(kT)$ size. This last strategy is not the most compact; still it is interesting because it also applies to the SP processor (see [3]). On the contrary SP does not efficiently support the strategy described next.

Without loss of generality (see Section 3), let us restrict our scope to CRCW decoupled WT algorithms. The resulting simulation is itself rather simple. If only exclusive writes are used, WT statement

```
for $j, 1 \leq j \leq p$ pardo
      operation$_j$
```

can be coded for SPE with a loop in the form:

```
segmentsize(min($k$, $p$))
for $j, 1 \leq j \leq p$ do
      instructions$_j$
```

where `instructions`$_j$ is the SPE coding for the high level WT operation.

As for concurrent writes, contention policies are quite different one from another, and therefore different approaches are needed for their implementation. For example, in case of reduction–like policies, such as MAX, + or logical AND, it is sufficient to append the appropriate reduction instruction to the core of the loop. The resulting SPE code would look like:

```
segmentsize(min($k$, $p$))
for $j, 1 \leq j \leq p$ do
    instructions$_j$
    $acc \leftarrow \max\{acc; output_j\}$
```

where $output_j$ is the result of `instructions`$_j$, the reduce operation is a MAX and the final result is accumulated in variable *acc*.

Another common policy, *priority* CW PRAM, can be implemented recurring to *predicated instructions*, whose output is committed to memory only if a certain condition is verified.

It must be noted that the correctness of the simulation relies on the following fatcs:

– SPE instructions are chosen to match the corresponding WT operations;
– each SPE instruction gets the right operands;
– memory writes are consistent with the policy specified by $\mathscr{A}_{WT}$.

Hence, Proposition 2 holds true.

Proposition 2. *Given WT parallel step s_i, it is possible to implement an equivalent SPE program $\mathscr{P}$, such that they both lead from memory state $\mathscr{M}_{i-1}$ to $\mathscr{M}_i$.*

Before examining the time complexity of the simulation, we must consider its memory footprint.

Proposition 3. *Let $M_{WT}^{(i)}$ (resp. $M_{WT}^{(i+1)}$) be the size of memory state $\mathscr{M}_i$ (resp. $\mathscr{M}_{i+1}$), and let M_{PH} be the amount of memory needed by $\mathscr{P}$. Then, both M_{PH} and $M_{WT}^{(i+1)}$ are $O(M_{WT}^{(i)} + p)$.*

Proof. Since p is the number of operations of the parallel step, $M_{WT}^{(i+1)}$ is $O(M_{WT}^{(i)} + p)$.

As for M_{PH}, an auxiliary operand vector only needs $O(\min\{p, k\}) = O(p)$ extra space. □

Proposition 4. *WT CRCW parallel step with p available parallelism can be translated into a SPE program $\mathscr{P}$ with $O(p + a(M_{PH}))$ time complexity.*

Proof. Consider the for loop which implements the simulation. Its body has $O(1)$ address dependence depth. Therefore, as in Example 1, the IGU is able to roll out $\lceil p/k \rceil$ segments with $O(k + a(M_{PH}))$ time complexity each. More precisely, at least $\lceil p/k \rceil - 1$ segments have $O(k)$ complexity, since $k \geq a(M_{total}) \geq a(M_{PH})$. Summing up, we get $O(p + a(M_{PH}))$.

As for reduction–like CWs, the simulation adds a functional dependency for each concurrent WT operation. Anyway, the internal forwarding mechanism of SPE can deal with them at a constant multiplicative slowdown. The same holds for CW implementations based on predicated instructions. □

Next we show how Proposition 4 can be repeatedly applied in order to get a whole SPE implementation of $\mathscr{A}_{WT}$. Correctness follows from the fact that each single application produces the same memory state as the correspondent parallel step.

Theorem 1. *Consider WT algorithm $\mathscr{A}_{WT}$, with W work and T time complexity. Then an equivalent SPE program $\mathscr{P}_A$ can be written, with complexity $O(W + Ta(M_{PH}))$, where M_{PH} is $\max_i\{M_{PH}^{(i)}\}$.*

Proof. $\mathscr{P}_A$ can be obtained with T applications of Proposition 4. The resulting complexity is therefore $\sum_i O(p_i + a(M_{PH}^{(i)}))$, which is $O(W + Ta(M_{PH}))$. □

One should note that this simulation results in a program of $O(T)$ instructions. Therefore instruction memory latencies can be ignored.

Corollary 1. *Let $\mathscr{A}_{WT}$ be a work–optimal WT algorithm. If $Ta(M_{PH})$ is $O(W)$, then there exists a SPE simulation of $\mathscr{A}_{WT}$ with optimal RAM complexity.*

As for M_{PH}, an immediate consequence of Prop. 3 follows.

Proposition 5. *$M_{PH} = \max_i\{M_{PH}^{(i)}\}$ is bounded from below by the maximum available parallelism of $\mathscr{A}_{WT}$ plus input size; from above by the work and the input size. Formally: $n + \max_i\{p_i\} \leq M_{PH} \leq n + W$.*

In general, any work–optimal parallel algorithm with polylogarithmic time complexity is a good candidate for efficient implementations on the SPE, if $a(x) < x$.

Anyway, another metric emerges from Prop. 5. In fact, exploiting all the available parallelism can affect memory usage, therefore increasing worst case latencies. Actually, once the condition $Ta(M_{PH}) = O(W)$ is met, any further parallelism would just increase memory footprint.

Section 5 contains a clear example of how this metric can be used.

5 Applications

Corollary 1 can be successfully applied to work–optimal WT algorithms which exhibit polylogarithmic time T, whenever the memory access function of the PHM is x^α, $0 < \alpha < 1$ or $\log x$. Finding *connected components* of a dense, undirected graph, for example, can be done with $T = \log^2 n$ and $W = n^2$ (see [12]) where input size is $O(n^2)$, n being the number of nodes. Therefore, we can implement a program for SPE with $O(n^2 + \log^2 n \cdot a(n^2)) = O(n^2)$ complexity, which is optimal.

An analogous result holds for the problem of merging two sorted lists of n elements. In this case we can resort to the work–optimal algorithm in [13], which results in a linear SPE program.

Some further analysis is needed, though, when these solutions are used as subroutines of larger programs. In particular, the underlying assumption that all the input is stored in the fastest memory locations may not hold. Consider for example an iterative bottom–up implementation of mergesort for an SPE with $a(x) = x^{\alpha}$. At iteration j, we have to merge pairs of 2^j-sized lists, with the ith pair starting at position $2^{j+1}i$. Each such merge has a $O(2^j + a(2^{j+1}i))$ complexity, which results to be superlinear if $O(2^j) < O(a(2^{j+1}i))$. For example the overall complexity of step $j = 0$ is $O(n^{1+\alpha})$.

In this case, a technique similar to the execution of consecutive searches of [14] can be employed. Basically, instead of merging one pair of sublists at a time, whenever the size of the subinstances is small enough, all the merges advance "concurrently". Therefore, it is possible to obtain segments of independent instructions, which are executed efficiently.

An interesting example is matrix multiplication of two $n \times n$ square matrices (input size is $O(n^2)$). Implementing the standard WT algorithm with n^3 parallelism yields an optimal SPE implementation $\mathscr{P}$, as far as time complexity is concerned. The result is achieved even if no locality is exploited. Anyway, $\mathscr{P}$ requires $\Omega(n^3)$ space to be executed. In other words, a SPE with M available memory size, could not multiply matrices bigger than $M^{1/3} \times M^{1/3}$. On the other hand, implementing a WT algorithm with n^2 available parallelism yields a both time and space optimal SPE program.

6 Conclusions and Future Work

This paper shows a general technique for exploiting the parallelism expressed by the WT framework in the SPE. In particular, it shows how concurrent operations can be sequentially executed in a pipelined–efficient way, and how such efficiency can be measured. Besides, it is also shown how too much parallelism can negatively affect memory usage, also when time complexity is not compromised.

The more straightforward research line involves assessing a larger group of problems and algorithms, possibly mapping whole computational categories to classes of SPE programs.

A second line is directed to the integration of this work with memory hierarchy exploitation.

Finally, as a link between coarse grained parallelism and memory hierarchy exploitation has already been proved [9], it would be interesting to see if and how such link exists also for pipelined memory exploitation.

References

[1] Aggarwal, A., Alpern, B., Chandra, A. K., and Snir, M.: A model for hierarchical memory. In *Proceedings of the 19th ACM Symposium on Theory of Computing*. ACM, New York, 305–314, 1987.

[2] Aggarwal, A., Chandra, A. K., and Snir, M.: Hierarchical memory with block transfer. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, 204–216, 1987.

[3] Bilardi, G., Ekanadham, K., and Pattnaik, P.: On approximating the ideal random access machine by physical machines. *Journal of the ACM 56*, 5, 1–57, August 2009.

[4] Bilardi, G., and Preparata, F.: Horizons of parallel computation. *J. Parall. Distrib. Comput. 27*, 2, 172–182, 1995.

[5] Brent, R. P.: The Parallel Evaluation of General Arithmetic Expressions. *Journal of the ACM 21*, 2, 201–206, April 1974.

[6] Chiang, Y., Goodrich, M. T., Grove, E. F., Tamassia, R., Vengroff, D. E., and Vitter, J. S.: External Memory Graph Algorithms. In *Proceeding SODA '95 Proceedings of the sixth annual ACM-SIAM Symposium on Discrete Algorithms*, 139–149, 1995.

[7] Collins, R. J.: MIMD Emulation on the Connection Machine. Technical Report CSD-910004, Dept. of Computer Science, Univ. of California, Los Angeles, Feb. 1988.

[8] Cook, S. A., Reckhow, R. A.: Time-bounded random access machines. *J. Comput. Syst. Sci. 7*, 4, 354–375, 1973.

[9] Fantozzi, C., Pietracaprina, A. A., Pucci, G.: Translating Submachine Locality into Locality of Reference. *Journal of Parallel and Distributed Computing 66*, 5, 633–646, 2006.

[10] Fortune, S., and Wyllie, J.: Parallelism in Random Access Machines. In *Proceeding STOC '78 Proceedings of the tenth annual ACM symposium on Theory of computing*, 114–118, 1978.

[11] Goldschlager, L. M.: A Unified Approach to models of Synchronous Parallel Machines. In *Proceeding STOC '78 Proceedings of the tenth annual ACM symposium on Theory of computing*, 89–94, 1978.

[12] Jájá, J. F.: An introduction to parallel algorithms. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1992. ISBN:0-201-54856-9.

[13] Kruskal, C.: Searching, Merging and Sorting in Parallel Computation. *IEEE Transactions on Computers - TC 32*, 10, 942–946, 1983.

[14] Luccio, F., and Pagli, L.: A model of sequential computation with pipelined access to memory. *Math. Syst. Theory 26*, 4, 343–356, 1993.

[15] Vitter, J. S.: External Memory Algorithms. In *Proceedings of the 6th Annual European Symposium on Algorithms*, Springer-Verlag, Berlin, Germany, 1–25, 1998.

A Continuous Cellular Automata Approach to Highway Traffic Modeling

Emanuele Rodaro[1] and Öznur Yeldan[2]

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

[2] Dipartimento di Matematica, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
oznur.yeldan@mail.polimi.it

1 Introduction

Traffic models are fundamental resources in the management of road network. A real progress in the study of traffic has obtained with the introduction of the models based on cellular automata (CA). CA models (CAMs) have the ability of being easily implemented for parallel computing because of their intrinsic synchronous behavior. They are conceptually simple, since a set of simple rules can be used to simulate a complex behavior. The traffic models based on CA are capable of capturing micro-level dynamics and relating these to macro-level traffic flow behavior. However, they are lack of the accuracy of other microscopic traffic models like the time-continuous car-following [1] ones where the behavior of a driver depends only on the leading vehicle. A basic one-dimensional CAM for highway traffic flow was first introduced by Wolfram, where he gave an extensive classification of CAMs as mathematical models for self-organizing dynamic systems [9, 10]. In 1992, Nagel and Schreckenberg proposed the first nontrivial traffic model (the NaSch model) based on CA for single-lane highway [5]. This paper gave rise to many other CAMs for traffic flow [3, 4, 6–8] whose common feature is that cells represent a piece of the road ("NaSch-type" models).

In this paper, we abandon the idea of dividing the road into cells and we introduce a new traffic model for highways using continuous cellular automata (CCA) to introduce the continuity in space. We consider a hybrid between the usual microscopic models (in general defined by means of a system of differential equations) which are very accurate in predicting general traffic behavior but computationally expensive, and the usual CAMs which are very efficient due to their simplicity and intrinsic parallelism making them natural to be implemented for parallel computing. This process of passing from the typical coarse-granularity of CAMs to the continuity in space is done assuming that cells represent vehicles. In this way, we obtain the immediate advantage of having less cells to compute. The continuity also gives us the possibility to refine the microscopic rules that govern the traffic dynamics using fuzzy reasoning to mimic different real-world driver behaviors. All parameters of the decision process of the drivers are modeled individually by means of fuzzy subsets, thus various types of drivers can be taken into consideration. This gives us the possibility to study how the heterogeneity influences the traffic macroscopically. The CCA model proposed in this paper is defined first for a single-lane road and then we extend the model to the multi-lane case where the extension is not as natural as "NaSch-type" models.

2 Overview of the Model

The single lane model is a CCA $\mathcal{SL} = (\mathbb{Z}, \Sigma, \mathcal{N}, \delta)$ where the lattice is the set of integers and the set of cell states $\Sigma = (K \times \mathbb{R}_0^+ \times \mathbb{R}_0^+ \times \mathbb{R} \times \{L, 0, R\} \times \{L, 0, R\}) \cup \{\perp\}$. A cell with the empty state $\perp$ represents a cell without a vehicle. The generic i-th non-empty cell is in the state $\sigma_i(t) = (k_i, x_i(t), v_i(t), s_i(t), d_i(t), d'_i(t))$ where

- k_i represents the kind of vehicles (seen as a unique entity driver/vehicle). It contains all the parameters such as: *the maximum velocity* (v_{max}), *the optimal velocity* (v_{opt}), *the length* (l_i), *the fuzzy membership functions*, *the maximum stress* (s_{max}), *the minimum stress* (s_{min}), *the probability functions of lane-changing to the right lane* ($P_R(x)$) and *to the left lane* ($P_L(x)$) (used in the multi-lane model).
- $x_i(t)$ is the position, $v_i(t)$ is the velocity, and $s_i(t)$ is the stress, a variable to keep track of how much the driver is above or below of his optimal velocity. In the single-lane model, $s_i(t)$ is introduced to implement a more realistic driver behavior since drivers usually tend to decelerate when they are moving with a velocity higher than their optimal velocity. However, this parameter is mainly used in the lane-changing process of the multi-lane model.
- $d_i(t)$ is the variable describing the desire for: lane-changing to the left "L", to the right "R" and staying on his own-lane "0", and $d'_i(t)$ is the variable showing from which lane the i-th vehicle is transferred: from the left lane "L", from the right lane "R" and not transferred "0"(these variables are used just in the multi-lane model).

$\mathcal{N}$ is a kind of one-dimensional extended Moore neighborhood defined by $\mathcal{N}(i) = (i-1, i, i+1, i+2)$, and $\delta : \Sigma^4 \rightarrow \Sigma$ is the local transition function defined componentwise. The space is updated by $x_i(t+1) = x_i(t) + v_i(t+1)$ (the unit of time is fixed to 1 sec.) and the velocity is updated by $v_i(t+1) = \min(v_{max}, \Delta x_i^+(t), \max(0, v_i(t) + A_i(t)))$ where $A_i(t)$ is the acceleration calculated using two sets of fuzzy IF-THEN rules (see Fig. 1) which take into consideration the distances and collision times of the back, front and next front vehicles, and the velocity. Besides the fuzzy rules to calculate $A_i(t)$, it is worth noting that in the case $A_i(t)$ is the stochastic function defined by $A_i(t) = 7{,}5\ m/s^2$ with probability p and $A_i(t) = 0$ otherwise, we essentially obtain the Nagel and Schreckenberg's first stochastic model [5] with the only difference that the space here is continuous. However, we have chosen to implement the decision of the acceleration using two fuzzy modules to mime driver behaviors more realistically.

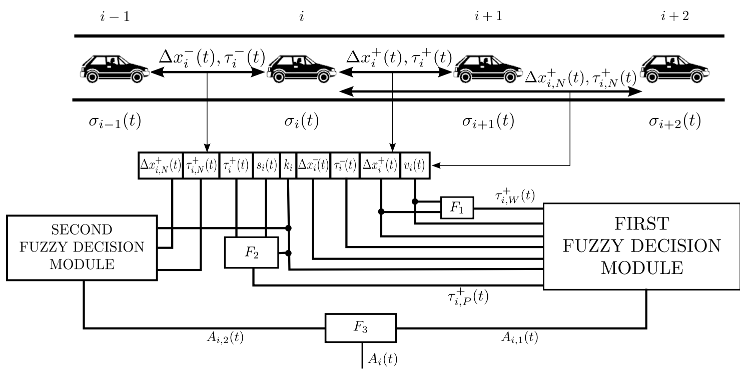


Fig. 1. A block diagram of the local transition function δ

The extension to the multi-lane case is not trivial as it is in the NaSch-type models where adding a lane simply means adding an array of cells and where the local transition function can naturally be extended. This is a consequence of having a clear physical interpretation of the model given by the fact that space is represented by cells. The most natural candidate is a union of interacting single-lane CCA where the interaction is given by a transfer operation. The process of transferring a vehicle from one lane to another depends on the desire of the vehicle to change lane (calculated using a stochastic process depending on the stress parameter) and the physical possibility of a vehicle to get transferred to a lane (depending on some safety constraints). Suppose that we have M-lanes represented by M-copies of the single-lane CCA $\mathcal{SL}$ in the configurations $c_1, \dots, c_M$. We scan each lane starting from the left-most lane (in the configuration c_1) and we transfer the vehicles to the adjacent lanes. After this process, for each lane we apply the single-lane CCA model to update the configuration and this update is done by means of the global transition function of $\mathcal{SL}$. In this way, we obtain a new array of configurations $c'_1, \dots, c'_M$, and this process represents 1 *sec.* of the simulation. Although this model is presented as an array of communicating CCA, we have proved that it is possible to define a CCA which actually simulates this model.

3 Conclusion

For a first test, we implement the model using Python with an object-oriented philosophy of programming. Using a questionnaire we set up two kinds of vehicles (long vehicles and passenger vehicles) which we have used to run a series of experiments. Analyzing the experimental results, we study the influence of different composition of vehicles on the macroscopic behavior of the traffic in order to observe the typical traffic flow phenomena (see Fig. 2). The code written in Python does not take advantage of the CA and its typical synchronous behavior.

For this reason, we also adapt the code using PyCuda to partially parallelize the multi-lane model on GPU's and we see that it is possible to boost the speed of execution by a factor of ~ 10, for instance, 1000 steps of the simulator with 5000 vehicles are run in 194 *sec.* instead of 1608 *sec.* (on a laptop equipped with a processor $i7$ intel and with a graphic card NVIDIA GeForce GT 555M).

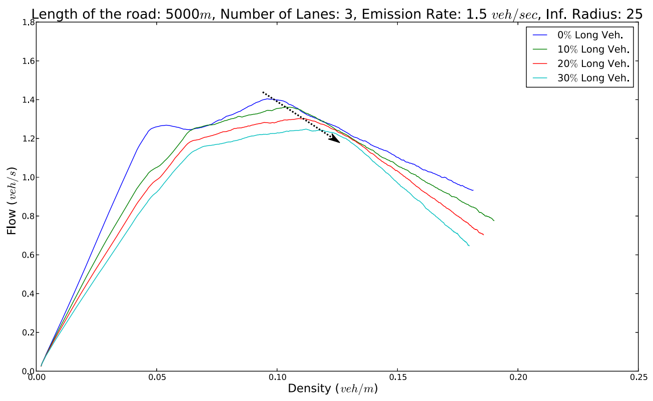


Fig. 2. The effect of heterogeneity on the fundamental diagram

References

1. Brackstone, M., McDonald, M.: Car-following: A historical review. Transportation Research Part F 2, 181–196 (1999)
2. Kari, J.: Theory of cellular automata: A survey. Theoretical Computer Science 334, 3–33 (2005)
3. Knospe, W., Santen, L., Schadschneider, A., Schreckenberg, M.: Disorder effects in cellular automata for two-lane traffic. Physica A: Statistical and Theoretical Physics 265(3–4), 614–633 (1999)
4. Maerivoet, S., De Moor, B.: Cellular automata models of road traffic. Physics Reports 419(1), 1–64 (2005)
5. Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. Journal de Physique I 2(12), 2221–2229 (1992)
6. Nagel, K., Wolf, D.E., Wagner, P., Simon, P.: Two-lane traffic rules for cellular automata: A systematic approach. Phys. Rev. E 58(2), 1425–1437 (1998)
7. Rickert, M., Nagel, K., Schreckenberg, M., Latour, A.: Two lane traffic simulations using cellular automata. Physica A: Statistical and Theoretical Physics 231(4), 534–550 (1996)
8. Wagner, P., Nagel, K., Wolf, D.E.: Realistic multi-lane traffic rules for cellular automata. Physica A: Statistical and Theoretical Physics 234(3–4), 687–698 (1997)
9. Wolfram, S.: Theory and applications of cellular automata. World Scientific Press, Singapore (1986)
10. Wolfram, S.: A new kind of science, Wolfram Media (2002)