9th International Workshop on Developments in Computational Models

DCM 2013

Buenos Aires, Argentina
26 August 2013

Preliminary Proceedings

Edited by Eduardo Bonelli, Mauricio Ayala-Rincón and Ian Mackie

Selected papers will be published in
Electronic Proceedings in Theoretical Computer Science.

DCM 2013 is a satellite event of CONCUR 2013

Preface

This volume contains the papers presented at the *Ninth International Workshop on Developments in Computational Models* (DCM) held in Buenos Aires, Argentina on 26th August 2013, as a satellite event of CONCUR 2013.

Several new models of computation have emerged in the last years, and many developments of traditional computational models have been proposed with the aim of taking into account the new demands of computer systems users and the new capabilities of computation engines. A new computational model, or a new feature in a traditional one, usually is reflected in a new family of programming languages, and new paradigms of software development.

The aim of this workshop is to bring together researchers who are currently developing new computational models or new features for traditional computational models, in order to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area. DCM 2013 will be a one-day satellite event of CONCUR 2013. This is the 9th event in the series since 2005 - see the DCM website for details of previous events.

Topics of interest include all abstract models of computation and their applications to the development of programming languages and systems. This includes (but is not limited to):

- Functional calculi: lambda-calculus, rho-calculus, term and graph rewriting;
- quantum computation, including implementations and formal methods in quantum protocols;
- probabilistic computation and verification in modelling situations;
- chemical, biological and bio-inspired computation, including spatial models, self-assembly, growth models;
- general concurrent models including the treatment of mobility, trust, and security;
- infinitary models of computation;
- information-theoretic ideas in computing.

The Programme Committee selected seven papers for presentation at DCM 2013. In addition, the programme includes invited talks by Verónica Becher, University of Buenos Aires, CONICET, Argentina, (*Turing's Normal Numbers: Towards Randomness*) and Joos Heintz, University of Buenos Aires, CONICET, Argentina, (*Quiz games: a new approach to information hiding based algorithms in scientific computing*).

Many people helped to make DCM 2013 a success. In particular, we wish to thank the CONCUR 2013 organisation team. We are also grateful to the external referees for their careful and efficient work in the reviewing process, and in particular the programme committee members: Pablo Arrighi, Pablo Barceló, Mario Benevides, Paola Bonizzoni, Nachum Dershowitz, Ruben Gamboa, Rajeev Goré, Holger Hermanns, Nao Hirokawa, Jean Krivine, Luis Lamb, Cesar Muñoz, Carlos Olarte, Femke van Raamsdonk, Camilo Rocha, Nora Szasz and René Thiemann.

Eduardo Bonelli, Mauricio Ayala-Rincón and Ian Mackie
DCM 2013 co-chairs

Contents

Turing's Normal Numbers: Towards Randomness

Verónica Becher
Universidad de Buenos Aires and CONICET

In a manuscript entitled ”A note on normal numbers” and written presumably in 1938 Alan Turing gave an algorithm that produces real numbers normal to every integer base. This proves, for the first time, the existence of computable instances and provides an answer to Borel's problem on giving examples of normality. Furthermore, with this work Turing pioneers the theory of randomness and shows that he had the insight, ahead of his time, that traditional mathematical concepts, like measure or continuity, could be made computational. In this talk I will highlight the ideas in these achievements of Turing, which are largely unknown because his manuscript remained unpublished until it appeared in his Collected Works in 1992.

Submitted to:
DCM 2013

The probability of non-confluent systems

Alejandro Díaz-Caro
Université Paris Ouest
200 avenue de la République
92001 Nanterre, France
INRIA
23 avenue d'Italie, CS 81321
75214 Paris Cedex 13, France
alejandro@diaz-caro.info

Gilles Dowek
INRIA
23 avenue d'Italie, CS 81321
75214 Paris Cedex 13, France
gilles.dowek@inria.fr

We show how to provide a structure of probability space to the set of execution traces on a non-confluent abstract rewrite system, by defining a variant of a Lebesgue measure on the space of traces. Then, we show how to use this probability space to transform a non-deterministic calculus into a probabilistic one. We use as example λ_+, a recently introduced calculus defined through type isomorphisms.

1 Introduction

Many probabilistic calculi has been developed in the pasts years, e.g. [1, 9, 13, 19, 20]. In particular, the algebraic versions of λ-calculus [5, 24] are extensions to λ-calculus where a linear combination of terms, e.g. $\alpha.\mathbf{r}+\beta.\mathbf{s}$, is also a term. One way to interpret such a linear combination is that it represents a term which is the term $\mathbf{r}$ with probability α, or the term $\mathbf{s}$ with probability β. However, endowing such a calculus with a non-restrictive type system is a challenge [3, 4].

A simpler framework is that of non determinisitic calculi which can be seen as algebraic calculi withouth scalars. They have been studied, for instance in [8, 10–12, 14–17, 21], however moving back from non-determinism to probabilities is not trivial. In this paper we propose, instead of changing these models, to define a probability measure on reductions in non-deterministic systems. In fact, as we shall see, such a probability measure can be defined on any abstract non-deterministic transition systems, or non-confluent abstract rewrite systems (ARS) (cf. [23, Chapter 1]).

Consider for example the following non-confluent ARS

$$\mathbf{a}\to\mathbf{b}\ ,\qquad \mathbf{a}\to\mathbf{c}\ ,\qquad \mathbf{c}\to\mathbf{d}\ ,\qquad \mathbf{c}\to\mathbf{e}\ ,$$

$$\begin{tikzcd}
 & \mathbf{a} \arrow[dl] \arrow[dr] & & \\
\mathbf{b} & & \mathbf{c} \arrow[dl] \arrow[dr] & \\
 & \mathbf{d} & & \mathbf{e}
\end{tikzcd}$$

we want to associate a probability to events such as

$$\mathbf{a}\to^*\mathbf{b}\ ,\qquad \mathbf{a}\to^*\mathbf{c}\ ,\qquad \mathbf{a}\to^*\mathbf{d}\ ,\qquad \mathbf{a}\to^*\mathbf{e}\ .$$

In this example, assuming equiprobability, we have $\mathtt{P}(\mathbf{a}\to^*\mathbf{b})=\frac{1}{2}$, $\mathtt{P}(\mathbf{a}\to^*\mathbf{c})=\frac{1}{2}$, $\mathtt{P}(\mathbf{a}\to^*\mathbf{d})=\frac{1}{4}$, $\mathtt{P}(\mathbf{a}\to^*\mathbf{e})=\frac{1}{4}$. Notice that these events are not disjoints and that their sum is larger than 1. In particular, $\mathbf{a}\to^*\mathbf{d}$ implies $\mathbf{a}\to^*\mathbf{c}$. Defining the elements of the set Ω of elementary events is not completely straightforward, in particular because we want to make it general enough to also consider infinite cases. For example, in the following system

$$\mathbf{a}_i\to\mathbf{a}_{i+1},\quad \mathbf{a}_i\to\mathbf{a}'_{i+1}\ ,$$

we naturally would like that $\mathtt{P}(\mathbf{a}_0\to^*\mathbf{a}_n)=\frac{1}{2^n}$.

Besides defining the elements of the set Ω, we need to define a notion of a measurable subset of Ω and endow such a subset with a probability distribution verifying the Kolmogorov axioms.

Preliminary Report. Final version to appear in:
DCM 2013

Our idea is to follow Lebesgue: define first the probability of rectangles, or boxes, then the probability of any set and finally measurable sets as those that verify Lebesgue's property. Thus besides defining the set Ω, we need to define a subset of $\mathcal{P}(\Omega)$ of boxes.

The first intuition would be to take paths as elements of the set Ω, for instance assigning the probability $\frac{1}{2}$ to the paths $\mathbf{a} \to \mathbf{b}$, $\frac{1}{4}$ to $\mathbf{a} \to \mathbf{c} \to \mathbf{d}$ and $\frac{1}{4}$ to $\mathbf{a} \to \mathbf{c} \to \mathbf{e}$. In fact it seems more convenient to extend such paths to strategies prescribing one reduct for each non normal object. Boxes are then defined as sets of strategies agreeing on a finite domain. We show in this paper that this is sufficient to define a probability space on strategies, consistent with the intuitive probability of events of the form $\mathbf{a} \to^* \mathbf{b}$.

Our study is generic enough to be applicable to several settings, such as automatons, or any other kind of transition systems. We use the nomenclature of abstract rewrite systems, but that of states and transitions could be used as well. Finally, we apply this construction to λ_+ [14, 15].

Plan of the paper. Section 2 introduces the basic concepts of strategies and boxes, it defines the Lebesgue measures. Section 3 proves that the space of strategies forms a probability space. Finally, in section 4 we show how to modify the calculus λ_+ into a probabilistic calculus λ_+^p. Also, we provide an encoding of an algebraic λ-calculus into λ_+^p and, to some extend, the inverse translation. The omitted proofs can be found at `http://www.diaz-caro.info/probas.pdf`.

2 Preliminaries

Let Λ be a set of objects and $\to$ a function from $\Lambda \times \Lambda$ to $\mathbb{N}$ such that for all $\mathbf{a}$ the set $\{\mathbf{b} \mid \to(\mathbf{a},\mathbf{b}) \neq 0\}$ is finite. We write $\mathbf{a} \to \mathbf{b}$ if $\to(\mathbf{a},\mathbf{b}) \neq 0$. We allow a term to be written to the same symbol more than once, so its probability increases, e.g. if $\to(\mathbf{a},\mathbf{b}) = 2$ and $\to(\mathbf{a},\mathbf{c}) = 1$, then the probability of getting $\mathbf{b}$ will be the double than the probability of getting $\mathbf{c}$. Think for example in a non-deterministic choice between two objects, which happen to be equal, then there would be two ways to get such an object by doing the choice. For a given object $\mathbf{a} \in \Lambda$, we denote by $\rho(\mathbf{a})$ its degree, that is, the number of objects to which it can be rewritten to in one step. Definition 2.1 formalises this.

Definition 2.1 (Degree of an object)**.** $\rho : \Lambda \to \mathbb{N}$ is a function defined by

$$\rho(\mathbf{a}) = \sum_{\mathbf{b}} \to(\mathbf{a},\mathbf{b}) \quad .$$

An object is normal if its degree is 0. We denote by $\Lambda^+ = \{\mathbf{a} \mid \mathbf{a} \in \Lambda \text{ and } \rho(\mathbf{a}) \geq 1\}$ to the set of non-normal objects, that is, objects that can be rewritten to other objects.

A strategy prescribing one reduct for each non-normal object is defined as a function from Λ^+ to Λ (cf. [23, Def. 4.9.1]).

Definition 2.2 (Strategy)**.** A strategy is a total function $f : \Lambda^+ \to \Lambda$ such that $f(\mathbf{a}) = \mathbf{b}$ implies $\mathbf{a} \to \mathbf{b}$. For instance, if $\mathbf{a} \to \mathbf{b}$ and $\mathbf{a} \to \mathbf{b}'$, there are two functions, f and f' assigning different results to $\mathbf{a}$. We denote by Ω the set of all such functions.

A box is a set of strategies agreeing on a finite domain.

Definition 2.3 (Box)**.** A box $B \subseteq \Omega$ is a set of the form $\{f \mid f(\mathbf{a}_1) = \mathbf{a}'_1, \ldots, f(\mathbf{a}_n) = \mathbf{a}'_n\}$ for some objects $\mathbf{a}_i$, $\mathbf{a}'_i$. We write $\mathcal{B}(\Omega)$ the subset of $\mathcal{P}(\Omega)$ containing all the boxes.

Example 2.4. Continuing with the example given at the introduction, $\Lambda^+ = \{\mathbf{a},\mathbf{c}\}$. Let $f_1(\mathbf{a}) = \mathbf{b}, f_1(\mathbf{c}) = \mathbf{d}$ and $f_2(\mathbf{a}) = \mathbf{b}, f_2(\mathbf{c}) = \mathbf{e}$ be two of the four strategies of Ω. Then the box $\{f \mid f(\mathbf{a}) = \mathbf{b}, f(\mathbf{c}) = \mathbf{d}\} = \{f_1\}$, and the box $\{f \mid f(\mathbf{a}) = \mathbf{b}\}$ is $\{f_1, f_2\}$.

$$\left\{ f_1 = \begin{tikzcd} & \mathbf{a} \arrow[dl] & \\ \mathbf{b} & & \mathbf{c} \arrow[dl] \\ & \mathbf{d} & \end{tikzcd} ; \quad f_2 = \begin{tikzcd} & \mathbf{a} \arrow[dl] & & \\ \mathbf{b} & & \mathbf{c} \arrow[dr] & \\ & & & \mathbf{e} \end{tikzcd} \right\} = \overset{\text{Box}}{\boxed{\begin{tikzcd} & \mathbf{a} \arrow[dl] \\ \mathbf{b} & \end{tikzcd}}}$$

A probability distribution can be defined in term of boxes, and then be extended to arbitrary sets of strategies.

Definition 2.5 (Probability function)**.** Let $\mathtt{p} : \mathcal{B}(\Omega) \to [0,1]$ be a total function defined over boxes as follows. If $B = \{f \mid f(\mathbf{a}_1) = \mathbf{a}'_1, \dots, f(\mathbf{a}_n) = \mathbf{a}'_n\}$, then

$$\mathtt{p}(B) = \prod_{i=1}^{n} \frac{\to(\mathbf{a}_i, \mathbf{a}'_i)}{\rho(\mathbf{a}_i)} \quad .$$

By convention, if no condition is given in B (i.e. $B = \Omega$), we have $n = 0$, and we consider the product of zero elements to be 1, the neutral element of the product.

Then we define the probability measure $\mathtt{P} : \mathcal{P}(\Omega) \to [0,1]$ for arbitrary sets of strategies as follows

$$\mathtt{P}(S) = \begin{cases} 0 & \text{if } S = \emptyset \\ \inf\{\sum_{B\in\mathcal{C}} \mathtt{p}(B) \mid \mathcal{C} \text{ is a countable family of boxes s.t. } S \subseteq \bigcup_{B\in\mathcal{C}} B\} & \text{in other case} \end{cases}$$

Example 2.6. Consider the ARS $\mathbf{a} \to \mathbf{b}$ with multiplicity 2 and $\mathbf{a} \to \mathbf{c}$ with multiplicity 1.

$$\begin{tikzcd} & \mathbf{a} \arrow[dl] \arrow[d] \arrow[dr] & \\ \mathbf{b} & \mathbf{b} & \mathbf{c} \end{tikzcd}$$

Let B be the box $B = \{f \mid f(\mathbf{a}) = \mathbf{b}\}$. Then we have $\mathtt{p}(B) = \frac{\to(\mathbf{a},\mathbf{b})}{\rho(\mathbf{a})} = \frac{2}{3}$. Intuitively, $\mathtt{P}(B)$ is the same as $\mathtt{p}(B)$ (this will be later formalised in Lemma 3.10), because B is the minimum cover of B. Hence $\mathtt{P}(B) = \frac{2}{3}$.

Example 2.7. We continue with the same running example depicted in the introduction. Let $f_1(\mathbf{a}) = \mathbf{b}$, $f_1(\mathbf{c}) = \mathbf{d}$ and $f_3(\mathbf{a}) = \mathbf{c}$, $f_3(\mathbf{c}) = \mathbf{e}$ be two strategies. Then the set $S = \{f_1, f_3\}$ is minimally covered by the boxes $B_1 = \{f_1\} = \{f \mid f(\mathbf{a}) = \mathbf{b}, f(\mathbf{c}) = \mathbf{d}\}$ and $B_2 = \{f_3\} = \{f \mid f(\mathbf{a}) = \mathbf{c}, f(\mathbf{c}) = \mathbf{e}\}$. So we have $P(S) = \mathtt{p}(B_1) + \mathtt{p}(B_2) = \frac{1}{2\times 2} + \frac{1}{2\times 2} = \frac{1}{2}$.

$$S = \left\{ f_1 = \begin{tikzcd} & \mathbf{a} \arrow[dl] & \\ \mathbf{b} & & \mathbf{c} \arrow[dl] \\ & \mathbf{d} & \end{tikzcd} ; \quad f_3 = \begin{tikzcd} \mathbf{a} \arrow[dr] & & \\ & \mathbf{c} \arrow[dr] & \\ & & \mathbf{e} \end{tikzcd} \right\}$$

Now we can define the Lebesgue measure in terms of the given probability measure.

Definition 2.8 (Measurable)**.** Let A be an element of $\mathcal{P}(\Omega)$, we write $A^\sim$ for the complement of A, that is $\Omega \setminus A$. The set A is Lebesgue measurable if $\forall S \in \mathcal{P}(\Omega)$, we have

$$\mathtt{P}(S) = \mathtt{P}(S \cap A) + \mathtt{P}(S \cap A^\sim) \quad .$$

We define $\mathcal{A} = \{A \mid A \text{ is measurable}\}$.

3 A probability space of strategies

The aim of this section is to prove that $(\Omega, \mathcal{A}, \mathtt{P})$ is a probability space. That is, the sample space Ω (the set of all possible strategies), the set of events $\mathcal{A}$, which is the set of the Lebesgue measurable sets of strategies, and the probability measure $\mathtt{P}$, form a probability space. Our proof follows [7]. We proceed by proving that this triplet satisfies the Kolmogorov axioms, that is the probability of any event is between 0 and 1, the probability of Ω is 1, and the probability of any countable sequence of pairwise disjoint (that is incompatible) events, is the sum of their probabilities. In order to do so, we need first to prove several properties.

Lemma 3.1 establishes several known properties of Lebesgue measurable sets.

Lemma 3.1.

1. *Let $A \in \mathcal{A}$ and $S \in \mathcal{P}(\Omega)$. If $A \cap S = \emptyset$, then $\mathtt{P}(A \cup S) = \mathtt{P}(A) + \mathtt{P}(S)$.*
2. *Let $A_1, A_2 \in \mathcal{A}$. If $A_1 \subseteq A_2$, then $\mathtt{P}(A_1) \leq \mathtt{P}(A_2)$.*
3. *$\emptyset$, the empty set, is Lebesgue measurable.*
4. *A is Lebesgue measurable if and only if $A^\sim$ is Lebesgue measurable.*
5. *If A_1, A_2 are Lebesgue measurable, then $A_1 \cup A_2$ is Lebesgue measurable.* □

The concept of algebra (Definition 3.2) gives a closure property of subsets. As a corollary of the Lemma 3.1 we can show that the set $\mathcal{A}$ of Lebesgue measurable sets form an algebra (Corollary 3.3).

Definition 3.2 (Algebra)**.** Let X be a set. We say that a set $\mathbb{A} \in \mathcal{P}(X)$ is an algebra over X if for all $A, B \in \mathbb{A}$, $A \cup B$, $A^\sim$ and X itself are also in $\mathbb{A}$.

Corollary 3.3. *$\mathcal{A}$ is an algebra over Ω.*

Proof. $\mathcal{A} \in \mathcal{P}(\Omega)$. Let $A, B \in \mathcal{A}$, then by Lemma 3.1(5), $A \cup B \in \mathcal{A}$. By Lemma 3.1(4), $A^\sim \in \mathcal{A}$. Finally, by Lemma 3.1(3) and (4), $\Omega \in \mathcal{A}$. □

Moreover, we can show that A is a σ-algebra, that is an algebra, completed to include countably infinite operations. Definition 3.4 formalises it.

Definition 3.4 (σ-algebra)**.** Let X be a set. We say that a set $\Sigma \in \mathcal{P}(X)$ is a σ-algebra over X if it is an algebra and it is closed under countable unions, that is, if $A_1, A_2, A_3, \ldots$ are in Σ, then so is $\bigcup A_i$.

Theorem 3.7 states that the set $\mathcal{A}$ of Lebesgue measurable sets is a σ-algebra. We need to prove two properties of Lebesgue measurable sets first (Lemmas 3.5 and 3.6).

Lemma 3.5. *Let $S \subseteq \Omega$ and $A_1, \ldots, A_n \in \mathcal{A}$ be a disjoint family. Then*

$$\mathtt{P}\left(S \cap \left(\bigcup_{i=1}^{n} A_i\right)\right) = \sum_{i=1}^{n} \mathtt{P}(S \cap A_i) \ .$$ □

Lemma 3.6. *Let $S_1, S_2, \cdots \subseteq \Omega$. Then*

$$\mathtt{P}\left(\bigcup_{i=1}^{\infty} S_i\right) \leq \sum_{i=1}^{\infty} \mathtt{P}(S_i) \ .$$ □

Using these properties, we can prove that $\mathcal{A}$ is a σ-algebra (Theorem 3.7).

Theorem 3.7. *$\mathcal{A}$ is a σ-algebra over Ω.*

Proof. By Corollary 3.3, $\mathcal{A}$ is an algebra. We only have to prove that $\mathcal{A}$ is closed under any countable unions. That is, if $B_1, B_2, \dots \in \mathcal{A}$, then $\bigcup_{i=1}^{\infty} B_i \in \mathcal{A}$. Since $\mathcal{A}$ is an algebra (Corollary 3.3), there is a disjoint family $A_1, A_2, \dots \in \mathcal{A}$ such that $A = \bigcup_{i=1}^{\infty} B_i = \bigcup_{i=1}^{\infty} A_i$. For example, we can take $A_1 = B_1$, $A_2 = B_2 \setminus B_1, A_3 = B_3 \setminus (B_1 \cup B_2), \dots$. Let $C_n = \bigcup_{i=1}^{n} A_i$, so $C_n \in \mathcal{A}$ again using that $\mathcal{A}$ is an algebra. Also notice that $A^\sim \subseteq C_n^\sim$ because $C_n \subseteq A$.

Since C_n is measurable, take any $S \subseteq \Omega$ and, using Lemma 3.1(2), we can calculate $\mathtt{P}(S) = \mathtt{P}(S \cap C_n) + \mathtt{P}(S \cap C_n^\sim) \geq \mathtt{P}(S \cap C_n) + \mathtt{P}(S \cap A^\sim)$. Since $\mathtt{P}(S \cap C_n) = \mathtt{P}(S \cap (\bigcup_{i=1}^{n} A_i))$, using Lemma 3.5, we obtain $\mathtt{P}(S) \geq \sum_{i=1}^{n} \mathtt{P}(S \cap A_i) + \mathtt{P}(S \cap A^\sim)$ and, since the left-hand side is independent of n, $\mathtt{P}(S) \geq \sum_{i=1}^{\infty} \mathtt{P}(S \cap A_i) + \mathtt{P}(S \cap A^\sim)$. Thus, by Lemma 3.6, $\mathtt{P}(S) \geq \mathtt{P}(S \cap (\bigcup_{i=1}^{\infty} A_i)) + \mathtt{P}(S \cap A^\sim) = \mathtt{P}(S \cap A) + \mathtt{P}(S \cap A^\sim)$.

For the converse inequality, notice that $S = (S \cap A) \cup (S \cap A^\sim)$, so using Lemma 3.6 we have $\mathtt{P}(S) = \mathtt{P}((S \cap A) \cup (S \cap A^\sim)) \leq \mathtt{P}(S \cap A) + \mathtt{P}(S \cap A^\sim)$. Hence, $A \in \mathcal{A}$. □

As intuited in Example 2.6, the probability of a box B is $\mathtt{p}(B)$. Lemma 3.10 formalises it. Before proving this lemma, we need two auxiliary ones (Lemmas 3.8 and 3.9). For short, we use the notation $B \cap \mathbf{a} = \mathbf{b}$ for $B \cap \{f \mid f(\mathbf{a}) = \mathbf{b}\}$.

Lemma 3.8. *Let $N \subseteq \mathbb{N}$ and for all $i \in N$, let $B, B_i \subseteq \Omega$ be boxes s.t. $B \subseteq \bigcup_{i \in N} B_i$ and $\mathtt{p}(B) > \sum_{i \in N} \mathtt{p}(B_i)$. Then for every object $\mathbf{a}$, there exists an object $\mathbf{b}$ such that, $\mathtt{p}(B \cap \mathbf{a} = \mathbf{b}) > \sum_{i \in N} \mathtt{p}(B_i \cap \mathbf{a} = \mathbf{b})$.* □

Lemma 3.9. *Let $N \subseteq \mathbb{N}$ and for all $i \in N$, let $B, B_i \subseteq \Omega$ be boxes s.t. $B \subseteq \bigcup_{i \in N} B_i$ and $\mathtt{p}(B) > \sum_{i \in N} \mathtt{p}(B_i)$. Then for all family $\{\mathbf{a}_j\}$ of objects, there exists a family $\{\mathbf{b}_j\}$ such that, for every k, $\mathtt{p}(B \cap \mathbf{a}_1 = \mathbf{b}_1 \cap \dots \cap \mathbf{a}_k = \mathbf{b}_k) > \sum_{i \in N} \mathtt{p}(B_i \cap \mathbf{a}_1 = \mathbf{b}_1 \cap \dots \cap \mathbf{a}_k = \mathbf{b}_k)$.* □

Lemma 3.10. *Let $B \subseteq \Omega$ be a box, then $\mathtt{P}(B) = \mathtt{p}(B)$.*

Proof. Let $B = \{f \mid f(\mathbf{a}_1) = \mathbf{a}'_1, \dots, f(\mathbf{a}_n) = \mathbf{a}'_n\}$. Since $B \subseteq B$, by definition of $\mathtt{P}$, we have $\mathtt{P}(B) \leq \mathtt{p}(B)$. We must prove $\mathtt{p}(B) \leq \mathtt{P}(B) = \inf\{\sum_{i \in N} \mathtt{p}(B_i) \mid B \subseteq \bigcup_{i \in N} B_i\}$. In other words, we must prove that $B \subseteq \bigcup_{i \in N} B_i$ implies $\mathtt{p}(B) \leq \sum_{i \in N} \mathtt{p}(B_i)$. We proceed by induction on n.

- If $n = 0$, $\mathtt{p}(B) = 1$. Notice that, without restrictions in B, $B = \Omega$. We prove this case by contradiction. Let $\mathtt{p}(F) > \sum_{i \in N} \mathtt{p}(B_i)$. Then by Lemma 3.9, there exists g such that for all k,
$$\mathtt{p}(\mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_k = g(\mathbf{a}_k)) > \sum_{i \in N} \mathtt{p}(B_i \cap \mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_k = g(\mathbf{a}_k)) \tag{1}$$
Since $g \in \Omega \subseteq \bigcup_{i \in N} B_i$, there exists j such that $g \in B_j$. Let B_j be defined with constraints on objects $\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_q}$. Let $k = q$ and from equation (1),
$$\mathtt{p}(\mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_q = g(\mathbf{a}_q)) > \sum_{i \in N} \mathtt{p}(B_i \cap \mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_q = g(\mathbf{a}_q)) \tag{2}$$
We know that $\mathtt{p}(\mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_q = g(\mathbf{a}_q)) = \prod_{h=1}^{q} \frac{\to(\mathbf{a}_h, g(\mathbf{a}_h))}{\rho(\mathbf{a}_h)}$, and since $g \in B_j$, we know that this is also equal to $\mathtt{p}(B_j \cap \mathbf{a}_1 = g(\mathbf{a}_1) \cap \dots \cap \mathbf{a}_q = g(\mathbf{a}_q))$. Hence equation (2) leads to a contradiction.

- Consider the case $n - 1$. Let $B' = \{f \mid \exists g \in B \text{ s.t. } \forall \mathbf{a} \neq \mathbf{a}_n,\ f(\mathbf{a}) = g(\mathbf{a})\}$. Then if $B' \subseteq \bigcup_{i \in N'} B'_i$ we have $\mathtt{p}(B') \leq \sum_{i \in N'} \mathtt{p}(B'_i)$. Notice that either $B'_i = B_i$ or B_i has a constraint on $\mathbf{a}_n$ and so $\frac{\to(\mathbf{a}_n, g(\mathbf{a}_n))}{\rho(\mathbf{a}_n)} \mathtt{p}(B'_i) = \mathtt{p}(B_i)$. In any case, $\frac{\to(\mathbf{a}_n, g(\mathbf{a}_n))}{\rho(\mathbf{a}_n)} \mathtt{p}(B'_i) \leq \mathtt{p}(B_i)$. Then $\mathtt{p}(B) = \frac{\to(\mathbf{a}_n, g(\mathbf{a}_n))}{\rho(\mathbf{a}_n)} \mathtt{p}(B') \leq \sum_{i \in N'} \frac{\to(\mathbf{a}_n, g(\mathbf{a}_n))}{\rho(\mathbf{a}_n)} \mathtt{p}(B'_i) \leq \sum_{i \in N'} \mathtt{p}(B_i)$. □

Theorem 3.11 (Space of strategies)**.** $(\Omega, \mathcal{A}, \mathtt{P})$ *is a probability space.*

Proof. We prove it satisfies the Kolmogorov axioms.

1st axiom: $\forall A \in \mathcal{A}$, $0 \leq \mathtt{P}(A) \leq 1$.

Since $\mathtt{P}$ is defined as an inf of sums of p, and p is always positive, so $\mathtt{P}$ cannot be negative. By the second Kolmogorov axiom $\mathtt{P}(\Omega) = 1$. Notice that A is measurable and $A \subseteq \Omega$, so $1 = \mathtt{P}(\Omega) = \mathtt{P}(\Omega \cap A) + \mathtt{P}(\Omega \setminus A) = \mathtt{P}(A) + \mathtt{P}(\Omega \setminus A)$, hence $1 - \mathtt{P}(\Omega \setminus A) = \mathtt{P}(A)$. Since $\mathtt{P}$ is not negative, $\mathtt{P}(A) \leq 1$.

2nd axiom: $\mathtt{P}(\Omega) = 1$.

Notice that Ω is the box including all the functions. Hence, there is no condition on the functions and so $n = 0$. Then $\mathtt{p}(\Omega) = 1$. By Lemma 3.10, $\mathtt{P}(\Omega) = \mathtt{p}(\Omega) = 1$.

3rd axiom: Any countable sequence of pairwise disjoint (i.e. incompatible) events $A_1, A_2 \dots \in \mathcal{A}$, satisfies $\mathtt{P}(A_1 \cup A_2 \dots) = \sum_{i=1}^{\infty} \mathtt{P}(A_i)$.

Let $\emptyset \neq I \subsetneq \mathbb{N}$. Since the sets A_i are in $\mathcal{A}$, consider $n \in \mathbb{N} \setminus I$ and we have

$$\mathtt{P}\left(\bigcup_{i \in \mathbb{N}\setminus I} A_i\right) = \mathtt{P}\left(\left(\bigcup_{i \in \mathbb{N}\setminus I} A_i\right) \cap A_n\right) + \mathtt{P}\left(\left(\bigcup_{i \in \mathbb{N}\setminus I} A_i\right) \cap A_n^{\sim}\right)$$

Notice that $\left(\bigcup_{i \in \mathbb{N}\setminus I} A_i\right) \cap A_n = A_n$ and since the A_i's are pairwise disjoint $\left(\bigcup_{i \in \mathbb{N}\setminus I} A_i\right) \cap A_n^{\sim} = \bigcup_{i \in \mathbb{N}\setminus (I \cup \{n\})} A_i$. Therefore, considering that this is valid for any I and $n \notin I$, we have

$$\mathtt{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \mathtt{P}(A_1) + \mathtt{P}\left(\bigcup_{i=2}^{\infty} A_i\right) = \mathtt{P}(A_1) + \mathtt{P}(A_2) + \mathtt{P}\left(\bigcup_{i=3}^{\infty} A_i\right) = \dots = \sum_{i=1}^{\infty} \mathtt{P}(A_i). \qquad \square$$

Example 3.12. Consider the non-strongly-normalising non-confluent rewrite system described in the introduction $\mathbf{a}_i \to \mathbf{a}_{i+1}$, $\quad \mathbf{a}_i \to \mathbf{a}'_{i+1}$, where each reduction is equiprobable and each symbol is different from each other. It can be depicted as follows.

$$\begin{tikzcd}
\mathbf{a}_0 \arrow[r] \arrow[dr] & \mathbf{a}_1 \arrow[r] \arrow[dr] & \mathbf{a}_2 \arrow[r, dotted] \arrow[dr] & {} \\
 & \mathbf{a}'_1 & \mathbf{a}'_2 & \mathbf{a}'_3
\end{tikzcd}$$

The probability that this rewrite system stops after exactly n steps, starting from term $\mathbf{a}_0$ is $\mathtt{P}(B)$, with $B = \{f \mid f(\mathbf{a}_0) = \mathbf{a}_1, \dots f(\mathbf{a}_{n-2}) = \mathbf{a}_{n-1}$ and $f(\mathbf{a}_{n-1}) = \mathbf{a}'_n\})$, and since B is a box, by Lemma 3.10 it is the same to $\mathtt{P}(B) = \mathtt{p}(B) = \dfrac{1}{\rho(\mathbf{a}_0)\dots\rho(\mathbf{a}_{n-1})} = \dfrac{1}{2^n}$.

The probability of stopping at the step n or before, starting at any point before a_{n-1}, is just the probability of the box $\{f \mid f(\mathbf{a}_{n-1}) = \mathbf{a}'_n\}$, which is $\dfrac{1}{2}$.

The probability of stopping at the step n or m, starting at any point before $\mathbf{a}_{n-1}$ and $\mathbf{a}_{m-1}$ is the probability of the union of two boxes, however they are not independent events (its intersection is not empty). Hence let $B_1 = \{f \mid \mathbf{a}_{n-1} = \mathbf{a}'_n\}$ and $B_2 = \{f \mid \mathbf{a}_{m-1} = \mathbf{a}'_m\}$. The probability $\mathtt{P}(B_1 \cup B_2) = \mathtt{P}((B_1 \setminus B_2) \cup B_2) = \mathtt{P}(B_1 \setminus B_2) \cup \mathtt{P}(B_2) = \mathtt{P}(\{f \mid \mathbf{a}_{n-1} = \mathbf{a}'_n, \mathbf{a}_{m-1} = \mathbf{a}'_m) + \mathtt{P}(B_2) = \dfrac{1}{4} + \dfrac{1}{2} = \dfrac{3}{4}$.

Finally, the probability of not stopping at all, is the probability of the set $S = \{f \mid f(\mathbf{a}_i) = \mathbf{a}_{i+1}$ for $i \in \mathbb{N}\}$, which is not a box, since there is an infinite number of conditions. It is easy to check that we need an infinite number of boxes to cover such a set, however we can chose boxes as small as we want (that is, with a big number of conditions), which makes the infimum of their sums to be 0, and so the probability of not stopping is, as expected, 0.

In other words, $\mathtt{P}(S) \leq \{f \mid f(\mathbf{a}_i) = \mathbf{a}_{i+1}, i \in [0,n]\} = \frac{1}{2^n}$, for any n. Hence when n tends to ∞, $\mathtt{P}(S)$ tends to 0.

4 Transforming a non-deterministic into a probabilistic calculus

4.1 The calculus λ_+

In [14, 15] we have introduced a non-deterministic calculus called λ_+, which is a simplification of an earlier probabilistic calculus by keeping non-determinism but removing explicit probabilities. Now we can transform this calculus into a probabilistic one.

The full calculus is depicted in Table 1. Typing judgements are of the form $\mathbf{r} : A$. A term $\mathbf{r}$ is typable if there exists a type A such that $\mathbf{r} : A$. Following [18, 22], we use a presentation of typed lambda-calculus without contexts and where each variable occurrence is labelled by its type, such as. $\lambda x^A.x^A$ or $\lambda x^A.y^B$. We sometimes omit the labels when they are clear from the context and write, for example, $\lambda x^A.x$ for $\lambda x^A.x^A$. We use different letters for different variables and the type system forbids terms such as $\lambda x^A.x^B$ when A and B are different, by imposing preconditions to when the typing rules apply. Let $S = \{x_1^{A_1}, \dots, x_n^{A_n}\}$ be a set of declarations, we write S^f when this set is functional, that is when $x_i = x_j$ implies $A_i = A_j$. For example $\{x^A, y^{A\Rightarrow B}\}^f$, but not $\{x^A, x^{A\Rightarrow B}\}^f$. Typing rules have the following structure:

$$[\text{Preconditions}]\frac{\text{Hypotheses}}{\text{Derived judgement}}\ (\text{Rule name})$$

The α-conversion and the sets $FV(\mathbf{r})$ of free variables of $\mathbf{r}$ and $FV(A)$ of free variables of A are defined as usual in the λ-calculus (cf. [6, §2.1]). For example $FV(x^A y^B) = \{x^A, y^B\}$. We say that a term $\mathbf{r}$ is closed whenever $FV(\mathbf{r}) = \emptyset$. If $FV(\mathbf{r}) = \{x_1^{A_1}, \dots, x_n^{A_n}\}$, we write $\Gamma(\mathbf{r}) = \{A_1, \dots, A_n\}$. $FV(\{A_1, \dots, A_n\})$ is defined by $\bigcup_{i=1}^n FV(A_i)$. Given two terms $\mathbf{r}$ and $\mathbf{s}$ we denote by $\mathbf{r}[\mathbf{s}/x]$ the term obtained by simultaneously substituting the term $\mathbf{s}$ for all the free occurrences of x in $\mathbf{r}$, subject to the usual proviso about renaming bound variables in $\mathbf{r}$ to avoid capture of the free variables of $\mathbf{s}$. Analogously $A[B/X]$ denotes the substitution of the type B for all the free occurrences of X in A, and $\mathbf{r}[B/X]$ the substitution in $\mathbf{r}$. For example, $(x^A)[B/Y] = x^{(A[B/Y])}$, $(\lambda x^A.\mathbf{r})[B/X] = \lambda x^{A[B/X]}.\mathbf{r}[B/X]$ and $(\pi_A(\mathbf{r}))[B/X] = \pi_{A[B/X]}(\mathbf{r}[B/X])$. Simultaneous substitutions are defined in the same way. Finally, terms and types are considered up to α-conversion.

Each term of the language has a main type associated, which can be obtained from the type annotations, and other types induced by the type equivalences.

The operational semantics of λ_+ is also given in Table 1, where there are two distinct relations between terms: a symmetric relation $\rightleftarrows$ and a reduction relation $\hookrightarrow$. We write $\rightleftarrows^*$ and $\hookrightarrow^*$ for the transitive and reflexive closures of $\rightleftarrows$ and $\hookrightarrow$ respectively. In particular, notice that $\rightleftarrows^*$ is an equivalence relation. We just write $\to$ when we do not want to make the distinction between these relations. We write $n.\mathbf{r}$ in λ_+ as a shorthand for $\underbrace{\mathbf{r} + \cdots + \mathbf{r}}_{n \text{ times}}$.

This calculus has a non-deterministic projector. Indeed, the rule "If $\mathbf{r} : A$, then $\pi_A(\mathbf{r} + \mathbf{s}) \hookrightarrow \mathbf{r}$" is not-deterministic because the symbol $+$ is commutative, so if $\mathbf{s} : A$, this rule can produce either $\mathbf{r}$ or $\mathbf{s}$ non-deterministically. In any case, both reducts are valid proofs of A, and so the proof system is consistent. Refer to [14] for details.

Grammar of types and terms

$$A,B,C,\dots ::= X \mid A \Rightarrow B \mid A \wedge B \mid \forall X.A \ .$$
$$\mathbf{r},\mathbf{s},\mathbf{t} ::= x^A \mid \lambda x^A.\mathbf{r} \mid \mathbf{rs} \mid \mathbf{r}+\mathbf{s} \mid \pi_A(\mathbf{r}) \mid \Lambda X.\mathbf{r} \mid \mathbf{r}\{A\} \ .$$

Equivalence between types

$$A \wedge B \equiv B \wedge A \ , \qquad (A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \ , \qquad A \Rightarrow (B \wedge C) \equiv (A \Rightarrow B) \wedge (A \Rightarrow C) \ .$$

Rewriting system

Symmetric relation:

$$\mathbf{r}+\mathbf{s} \rightleftarrows \mathbf{s}+\mathbf{r} \ , \qquad (\mathbf{r}+\mathbf{s})\mathbf{t} \rightleftarrows \mathbf{rt}+\mathbf{st} \ , \qquad \text{If } \mathbf{r} : A \Rightarrow (B \wedge C), \text{ then}$$
$$(\mathbf{r}+\mathbf{s})+\mathbf{t} \rightleftarrows \mathbf{r}+(\mathbf{s}+\mathbf{t}) \ , \qquad \lambda x^A.(\mathbf{r}+\mathbf{s}) \rightleftarrows \lambda x^A.\mathbf{r} + \lambda x^A.\mathbf{s} \ , \qquad \pi_{A\Rightarrow B}(\mathbf{r})\mathbf{s} \rightleftarrows \pi_B(\mathbf{rs}) \ .$$

Reductions:

$$(\lambda x^A.\mathbf{r})\ \mathbf{s} \hookrightarrow \mathbf{r}[\mathbf{s}/x] \ , \qquad (\Lambda X.\mathbf{r})\{A\} \hookrightarrow \mathbf{r}[A/X] \ , \qquad \text{If } \mathbf{r} : A, \text{ then } \pi_A(\mathbf{r}+\mathbf{s}) \hookrightarrow \mathbf{r} \ .$$

Typing system

$$[A\equiv B]\frac{\mathbf{r} : A}{\mathbf{r} : B}\ (\equiv) \qquad \frac{}{x^A : A}\ (ax) \qquad [(FV(\mathbf{r})\cup\{x^A\})^f]\frac{\mathbf{r} : B}{\lambda x^A.\mathbf{r} : A \Rightarrow B}\ (\Rightarrow_i) \qquad [FV(\mathbf{rs})^f]\frac{\mathbf{r} : A \Rightarrow B \quad \mathbf{s} : A}{\mathbf{rs} : B}\ (\Rightarrow_e)$$

$$[FV(\mathbf{r}+\mathbf{s})^f]\frac{\mathbf{r} : A \quad \mathbf{s} : B}{\mathbf{r}+\mathbf{s} : A \wedge B}\ (\wedge_i) \qquad \frac{\mathbf{r} : A \wedge B}{\pi_A(\mathbf{r}) : A}\ (\wedge_e) \qquad [X \notin FV(\Gamma(\mathbf{r}))]\frac{\mathbf{r} : A}{\Lambda X.\mathbf{r} : \forall X.A}\ (\forall_i) \qquad \frac{\mathbf{r} : \forall X.A}{\mathbf{r}\{B\} : A[B/X]}\ (\forall_e)$$

Table 1: The non-deterministic calculus λ_+

4.2 From non-determinism to probabilities (or from λ_+ to λ_+^p)

Consider the following example (cf. [15, Example 5]). Two possible reduction paths can be fired from $(\Lambda X.(\pi_A(x^A + y^X)))\{A\}$: Reducing first the projection, $(\Lambda X.x^A)\{A\} \hookrightarrow x^A$, or reducing first the beta $\pi_A(x^A + y^A) \hookrightarrow x^A$. The former path is deterministic and will always reduce to x^A, on the contrary, the latter can non-deterministically chose between x^A and y^A. However, in both cases a proof of A is obtained.

Hence, the non-determinism is present not only due to the projector, but also by a combination of not defining a reduction strategy and the polymorphism, which can turn a deterministic projection into a non-deterministic one. We want to associate a probability to the second case, that is, to the non-deterministic projector (the π reduction). With this aim, we consider the following ARS, called $\lambda_+^{\downarrow}$. The closed normal terms of λ_+ are objects of $\lambda_+^{\downarrow}$. If $\mathbf{r}_1,\dots,\mathbf{r}_n$ are objects, then it is also an object. The function $\to$ is given by the relations $\rightleftarrows$ and $\hookrightarrow$. In particular, if $\mathbf{r} : A$, then $\pi_A(\mathbf{r}+\mathbf{r}) \to \mathbf{r}$, with multiplicity 2, i.e. $\to(\pi_A(\mathbf{r}+\mathbf{r}),\mathbf{r}) = 2$.

Theorem 4.1. *Let $(\Omega,\mathcal{A},\mathtt{P})$ be a probability space over $\lambda_+^{\downarrow}$. Let $B_{\mathbf{r}_i} = \{f \mid f(\pi_A(\sum_{j=1}^n m_j.\mathbf{r}_j)) = \mathbf{r}_i\}$ be a box. Then $\mathtt{P}(B_{\mathbf{r}_i}) = \frac{m_i}{\sum_{j=1}^n m_j}$.*

Proof. Notice that

$$\rho(\pi_A(\sum_{i=1}^n m_i.\mathbf{r}_i)) = \sum_{\mathbf{r}} \to (\ \pi_A(\sum_{i=1}^n m_i.\mathbf{r}_i),\mathbf{r}) = \sharp[\underbrace{\mathbf{r}_1,\cdots,\mathbf{r}_1}_{m_1 \text{ times}},\dots,\underbrace{\mathbf{r}_n,\cdots,\mathbf{r}_n}_{m_n \text{ times}}] = \sum_{j=1}^n m_j$$

And $\to(\pi_A(\sum_{i=1}^n m_i.\mathbf{r}_i),\mathbf{r}_i) = m_i$. Hence, $\mathtt{P}(B_{\mathbf{r}_i}) = \mathtt{p}(B_{\mathbf{r}_i}) = \frac{m_i}{\sum_{j=1}^n m_j}$. □

Definition 4.2 (The probabilistic calculus λ_+^p)**.** Let λ_+^p be the language of Table 1, with the following modification:
Replace rule "If $\mathbf{r} : A$, then $\pi_A(\mathbf{r}+\mathbf{s}) \hookrightarrow \mathbf{r}$" by
"For $i = 1,\dots,n$, let $\mathbf{r}_i : A$ and $\mathbf{s} \not: A$, be closed normal terms. Then

$$\pi_A(\sum_{i=1}^{n} m_i.\mathbf{r}_i + \mathbf{s}) \hookrightarrow \mathbf{r}_i \qquad \text{with probability } \frac{m_i}{\sum_{j=1}^{n} m_j}\text{''} \ .$$

Remark 4.3. Notice that by Theorem 4.1 the probabilistic reduction is well defined.

4.3 The calculus Alg_F^p

The calculus Alg_F^p is inspired from [5, 24]. We restrict the algebraic calculus to only have probabilistic superpositions, and we type it with a simple extension of System F (cf. [2, Def. 5.1]). The grammar of terms ensures that the linear combinations of terms are probability distributions, however the type system allows typing pseudo-terms, that is, terms that are not probability distributions. A term in this language, is a term produced by the grammar of terms, and typed. The full calculus is depicted in Table 2.

Grammar of types

$$A,B,C,\dots \ ::= \quad X \mid A \Rightarrow B \mid \forall X.A \ .$$

Grammar of pseudo-terms

$$\mathbf{r},\mathbf{s},\mathbf{t} \ ::= \quad x^A \mid \lambda x^A.\mathbf{r} \mid \mathbf{rs} \mid \Lambda X.\mathbf{r} \mid \mathbf{r}\{A\} \mid p.\mathbf{r} \mid \mathbf{r}+\mathbf{s}$$

Grammar of terms

$$\mathbf{r},\mathbf{s},\mathbf{t} \ ::= \quad x^A \mid \lambda x^A.\mathbf{r} \mid \mathbf{rs} \mid \Lambda X.\mathbf{r} \mid \mathbf{r}\{A\} \mid \sum_{i=1}^{n} p_i.\mathbf{r}_i \qquad \text{with } \begin{cases} n>0, \\ p_i \in \mathbb{Q}(0,1] \text{ and} \\ \sum_{i=1}^{n} p_i = 1 \end{cases}$$

Rewriting system

Symmetric relation:

$$\mathbf{r}+\mathbf{s} \rightleftarrows \mathbf{s}+\mathbf{r} \ , \qquad (\mathbf{r}+\mathbf{s})\mathbf{t} \rightleftarrows \mathbf{rt}+\mathbf{st} \ , \qquad 1.\mathbf{r} \rightleftarrows \mathbf{r} \ .$$

$$(\mathbf{r}+\mathbf{s})+\mathbf{t} \rightleftarrows \mathbf{r}+(\mathbf{s}+\mathbf{t}) \ , \qquad \lambda x^A.(\mathbf{r}+\mathbf{s}) \rightleftarrows \lambda x^A.\mathbf{r}+\lambda x^A.\mathbf{s} \ ,$$

Reductions:

Beta

$$(\lambda x^A.\mathbf{r})\,\mathbf{s} \hookrightarrow \mathbf{r}[\mathbf{s}/x] \ , \qquad (\Lambda X.\mathbf{r})\{A\} \hookrightarrow \mathbf{r}[A/X] \ ,$$

Elementary

$$p.q.\mathbf{r} \hookrightarrow pq.\mathbf{r} \ , \qquad p.(\mathbf{r}+\mathbf{s}) \hookrightarrow p.\mathbf{r}+p.\mathbf{s} \ ,$$

Factorisation

$$p.\mathbf{r}+q.\mathbf{r} \hookrightarrow (p+q).\mathbf{r} \ .$$

Typing system

$$\frac{}{x^A : A}\,(ax) \qquad [(FV(\mathbf{r})\cup\{x^A\})^f]\frac{\mathbf{r} : B}{\lambda x^A.\mathbf{r} : A \Rightarrow B}\,(\Rightarrow_i) \qquad [FV(\mathbf{rs})^f]\frac{\mathbf{r} : A \Rightarrow B \quad \mathbf{s} : A}{\mathbf{rs} : B}\,(\Rightarrow_e)$$

$$[FV(\mathbf{r}+\mathbf{s})^f]\frac{\mathbf{r} : A \quad \mathbf{s} : A}{\mathbf{r}+\mathbf{s} : A}\,(+_i) \qquad \frac{\mathbf{r} : A}{p.\mathbf{r} : A}\,(p_i) \qquad [X \notin FV(\Gamma(\mathbf{r}))]\frac{\mathbf{r} : A}{\Lambda X.\mathbf{r} : \forall X.A}\,(\forall_i) \qquad \frac{\mathbf{r} : \forall X.A}{\mathbf{r}\{B\} : A[B/X]}\,(\forall_e)$$

Table 2: The algebraic calculus Alg_F^p.

4.4 From Alg_F^p to λ_+^p

We give a translation from the probabilistic calculus Alg_F^p, including scalars, to the probabilistic calculus λ_+^p.

$$\begin{array}{rclrclrcl} [\![x^A]\!] &=& x^A & [\![\mathbf{rs}]\!] &=& [\![\mathbf{r}]\!][\![\mathbf{s}]\!] & [\![\mathbf{r}\{A\}]\!] &=& [\![\mathbf{r}]\!]\{A\} \\ [\![\lambda x^A.\mathbf{r}]\!] &=& \lambda x^A.[\![\mathbf{r}]\!] & [\![\Lambda X.\mathbf{r}]\!] &=& \Lambda X.[\![\mathbf{r}]\!] & [\![\sum_{i=1}^n \frac{n_i}{d_i}.\mathbf{r}_i]\!] &=& \pi_A(\sum_{i=1}^n m_i.[\![\mathbf{r}_i]\!]) \end{array}$$
$$\text{where } \mathbf{r}_i : A, d_i \in \mathbb{N}^*, m_i = n_i(\prod_{\substack{k=1\\k\neq i}}^n d_k), \text{ for } i = 1,\dots,n.$$

Example 4.4. Let $\mathbf{r} : A$, $\mathbf{t} : A$ and $\mathbf{s} : A$. $[\![\frac{3}{4}.\mathbf{r}+\frac{1}{8}.\mathbf{t}+\frac{1}{8}.\mathbf{s}]\!] = \pi_A(192.[\![\mathbf{r}]\!]+32.[\![\mathbf{t}]\!]+32.[\![\mathbf{s}]\!])$. By Theorem 4.1, this last term reduces to $[\![\mathbf{r}]\!]$ with probability $\frac{192}{192+32+32} = \frac{3}{4}$, to $[\![\mathbf{t}]\!]$ with probability $\frac{32}{192+32+32} = \frac{1}{8}$, and to $[\![\mathbf{s}]\!]$ with probability $\frac{32}{192+32+32} = \frac{1}{8}$.

Lemma 4.5. $[\![\mathbf{r}]\!][[\![\mathbf{s}]\!]/x] = [\![\mathbf{r}[\mathbf{s}/x]]\!]$. □

Theorem 4.6. *If* $\mathbf{r} \to^* \sum_{i=1}^n p_i.\mathbf{t}_i$, *with* $\mathbf{t}_i$ *in* Alg_F^p, *with* $\sum_{i=1}^n p_i = 1$ *and* $[\![\mathbf{t}_i]\!] \to^* \mathbf{s}_i$, *then* $[\![\mathbf{r}]\!] \to^* \mathbf{s}_i$ *with probability* $p_i\left(\sum_{j=1}^n p_j\right)^{-1}$ *in* λ_+^p.

Proof. Let $\mathbf{r} : A$ in Alg_F^p. For $i = 1,\dots,n$, assume $p_i = \frac{n_i}{d_i}$ with $n_i, d_i \in \mathbb{N}^*$. We proceed by a case analysis on the last reduction step to reach $\sum_{i=1}^n p_i.\mathbf{t}_i$. We only give two cases as example.

- If $\mathbf{r} = \sum_{i=1}^n p_i.\mathbf{t}_i$, then $\pi_A(\sum_{i=1}^n(\prod_{\substack{k=1\\k\neq i}}^n d_k n_i).[\![\mathbf{t}_i]\!]) \to^* \pi_A(\sum_{i=1}^n(\prod_{\substack{k=1\\k\neq i}}^n d_k n_i).\mathbf{s}'_i)$ By Theorem 4.1, this term reduces in one step to $\mathbf{s}'_i$ with probability
$$\frac{\prod_{\substack{k=1\\k\neq i}}^n d_k n_i}{\sum_{i=1}^n\left(\prod_{\substack{k=1\\k\neq i}}^n d_k n_i\right)} = \left(\frac{\frac{n_i}{d_i}}{\sum_{i=1}^n \frac{n_i}{d_i}}\right)\cdot\left(\frac{\prod_{k=1}^n d_k}{\prod_{k=1}^n d_k}\right) = p_i\left(\sum_{j=1}^n p_j\right)^{-1}.$$
- Consider $(\lambda x^A.\mathbf{r})\ \mathbf{s} \hookrightarrow \mathbf{r}[\mathbf{s}/x]$, with $\mathbf{r}[\mathbf{s}/x] = \sum_{i=1}^n p_i.\mathbf{t}_i$. Then $[\![(\lambda x^A.\mathbf{r})\ \mathbf{s}]\!] = (\lambda x^A.[\![\mathbf{r}]\!])\ [\![\mathbf{s}]\!] \hookrightarrow [\![\mathbf{r}]\!][[\![\mathbf{s}]\!]/x]$ which, by Lemma 4.5, is equal to $[\![\mathbf{r}[\mathbf{s}/x]]\!] = [\![\sum_{i=1}^n p_i.\mathbf{t}_i]\!]$ and this, by definition is equal to $\pi_A\left(\sum_{i=1}^n(\prod_{\substack{k=1\\k\neq i}}^n d_k n_i).[\![\mathbf{t}_i]\!]\right)$. We conclude with Theorem 4.1. □

4.5 Back from λ_+^p to Alg_F^p

The inverse translation is given by

$$\begin{array}{rclrclrcl} (\!|x^A|\!) &=& x^A & (\!|\mathbf{rs}|\!) &=& (\!|\mathbf{r}|\!)(\!|\mathbf{s}|\!) & (\!|\mathbf{r}\{A\}|\!) &=& (\!|\mathbf{r}|\!)\{A\} \\ (\!|\lambda x^A.\mathbf{r}|\!) &=& \lambda x^A.(\!|\mathbf{r}|\!) & (\!|\Lambda X.\mathbf{r}|\!) &=& \Lambda X.(\!|\mathbf{r}|\!) & (\!|\mathbf{r}+\mathbf{s}|\!) &=& (\!|\mathbf{r}|\!)+(\!|\mathbf{s}|\!) \end{array}$$
$$\text{If } \pi_A(\mathbf{t}) \hookrightarrow \mathbf{s}_i \text{ with probability } p_i, \text{ for } i=1,\dots,n,\quad (\!|\pi_A(\mathbf{t})|\!) = \sum_{i=1}^n p_i.(\!|\mathbf{s}_i|\!)$$

Remark 4.7. This translation does not admit translating a term of the form $\pi_A(\mathbf{t})$ in normal form. Moreover, let Π be the rule "$\pi_{A\Rightarrow B}(\mathbf{r})\mathbf{s} \rightleftarrows \pi_B(\mathbf{rs})$ with $\mathbf{r} : A \Rightarrow (B \wedge C)$", then the translation keep reductions, except for the one using rule Π, as expressed in Theorem 4.9.

Lemma 4.8. $(\!|\mathbf{r}|\!)[(\!|\mathbf{s}|\!)/x] = (\!|\mathbf{r}[\mathbf{s}/x]|\!)$. □

Theorem 4.9. *Let* $\mathbf{r},\mathbf{s},\mathbf{s}_i$ *in* λ_+^p.

- *If* $\mathbf{r} \rightleftarrows \mathbf{s}$*, then* $(\!|\mathbf{r}|\!) \rightleftarrows (\!|\mathbf{s}|\!)$.
- *If* $\mathbf{r} \hookrightarrow \mathbf{s}$*, with probability* 1*, then* $(\!|\mathbf{r}|\!) \hookrightarrow (\!|\mathbf{s}|\!)$*, except if the reduction is done by rule* Π.
- *If* $\mathbf{r} \hookrightarrow \mathbf{s}_i$ *with probability* p_i*, for* $i = 1,\dots,n$*, then* $(\!|\mathbf{r}|\!) = \sum_{i=1}^n p_i.(\!|\mathbf{s}_i|\!)$.

Proof. Case by case analysis. We only give three cases as example.

- Consider $(\mathbf{r}+\mathbf{s})\mathbf{t} \rightleftarrows \mathbf{rt}+\mathbf{st}$. Notice that $(\!|(\mathbf{r}+\mathbf{s})\mathbf{t}|\!) = ((\!|\mathbf{r}|\!)+(\!|\mathbf{s}|\!))(\!|\mathbf{t}|\!) \rightleftarrows (\!|\mathbf{r}|\!)(\!|\mathbf{t}|\!)+(\!|\mathbf{s}|\!)(\!|\mathbf{t}|\!) = (\!|\mathbf{rt}+\mathbf{st}|\!)$.
- Consider $(\lambda x^A.\mathbf{r})\mathbf{s} \hookrightarrow \mathbf{r}[\mathbf{s}/x]$. Notice that $(\!|(\lambda x^A.\mathbf{r})\mathbf{s}|\!) = (\lambda x^A.(\!|\mathbf{r}|\!))(\!|\mathbf{s}|\!) \hookrightarrow (\!|\mathbf{r}|\!)[(\!|\mathbf{s}|\!)/x]$, and this, by Lemma 4.8, is equal to $(\!|\mathbf{r}[\mathbf{s}/x]|\!)$.
- Consider $\pi_A(\sum_{i=1}^n m_i.\mathbf{r}_i+\mathbf{s}) \hookrightarrow \mathbf{r}_i$ with probability $\frac{m_i}{\sum_{j=1}^n m_j}$, where $\mathbf{r}_i : A$ and $\mathbf{s} \not: A$ are closed normal terms. Notice that, by definition, $(\!|\pi_A(\sum_{i=1}^n m_i.\mathbf{r}_i+\mathbf{s})|\!) = \sum_{i=1}^n \frac{m_i}{\sum_{j=1}^n m_j}.(\!|\mathbf{r}_i|\!)$. $\square$

5 Conclusion

In this paper we have defined a probability space on the execution traces of non-confluent abstract rewrite systems. We define a sample space on strategies deciding the rewrite to apply at each state (cf. Definition 2.2).

Our main motivation has been to be able to use this probability space in non-deterministic calculi, hence being able to encode a probability superposition of the kind $\alpha.\mathbf{t}+\beta.\mathbf{r}$, with $\alpha+\beta = 1$, as a term having probability α of rewriting to $\mathbf{t}$ and probability β of rewriting to $\mathbf{r}$. As an example, we provided such an encoding from an algebraic calculus into a non-deterministic calculus.

References

[1] Suzana Andova (1999): *Process Algebra with Probabilistic Choice*. In Joost-Pieter Katoen, editor: *Formal Methods for Real-Time and Probabilistic Systems*, *Lecture Notes in Computer Science* 1601, pp. 111–129, doi:10.1007/3-540-48778-6_7.

[2] Pablo Arrighi & Alejandro Díaz-Caro (2012): *A System F Accounting for Scalars*. *Logical Methods in Computer Science* 8(1:11), doi:10.2168/LMCS-8(1:11).

[3] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2012): *A Type System for the Vectorial Aspects of the Linear-Algebraic Lambda-Calculus*. In Elham Kashefi, Jean Krivine & Femke van Raamsdonk, editors: *Developments in Computational Models*, *Electronic Proceedings in Theoretical Computer Science* 88, pp. 1–15, doi:10.4204/EPTCS.88.1.

[4] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2013): *The Vectorial Lambda-Calculus*. Available at `http://www.diaz-caro.info/TheVectorialCalculus.pdf`. (Submitted).

[5] Pablo Arrighi & Gilles Dowek (2008): *Linear-algebraic λ-calculus: higher-order, encodings, and confluence*. In Andrei Voronkov, editor: *Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 5117, pp. 17–31, doi:10.1007/978-3-540-70590-1_2. Available at `http://arxiv.org/abs/quant-ph/0612199`.

[6] Henk Barendregt (1984): *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam.

[7] John J. Benedetto & Wojciech Czaja (2009): *Integration and Modern Analysis*, chapter Lebesgue Measure and General Measure Theory. Birkhiäuser Advanced Texts Basler Lehrbücher, Birkhäuser Boston.

[8] Gérard Boudol (1994): *Lambda-Calculi for (Strict) Parallel Functions*. *Information and Computation* 108(1), pp. 51–127, doi:10.1006/inco.1994.1003.

[9] Olivier Bournez & Mathieu Hoyrup (2003): *Rewriting Logic and Probabilities*. In Robert Nieuwenhuis, editor: *Rewriting Techniques and Applications*, *Lecture Notes in Computer Science* 2706, pp. 61–75, doi:10.1007/3-540-44881-0_6. Available at http://hal.inria.fr/inria-00099620.

[10] Antonio Bucciarelli, Thomas Ehrhard & Giulio Manzonetto (2012): *A Relational Semantics for Parallelism and Non-Determinism in a Functional Setting*. *Annals of Pure and Applied Logic* 163(7), pp. 918–934, doi:10.1016/j.apal.2011.09.008. Available at hal.inria.fr:inria-00628887.

[11] Ugo de'Liguoro & Adolfo Piperno (1995): *Non Deterministic Extensions of Untyped λ-calculus*. *Information and Computation* 122(2), pp. 149–177, doi:10.1006/inco.1995.1145.

[12] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro & Adolfo Piperno (1998): *A filter model for concurrent λ-calculus*. *SIAM Journal on Computing* 27(5), pp. 1376–1419, doi:10.1137/S0097539794275860.

[13] Alessandra Di Pierro, Chris Hankin & Herbert Wiklicky (2005): *Probabilistic λ-Calculus and Quantitative Program Analysis*. *Journal of Logic and Computation* 15(2), pp. 159–179, doi:10.1093/logcom/exi008.

[14] Alejandro Díaz-Caro & Gilles Dowek (2013): *Non determinism through type isomorphism*. In Delia Kesner & Petrucio Viana, editors: *Logical and Semantic Frameworks, with Applications*, *Electronic Proceedings in Theoretical Computer Science* 113, pp. 137–144, doi:10.4204/EPTCS.113.13.

[15] Alejandro Díaz-Caro & Gilles Dowek (2013): *Normalisation of a non-deterministic type isomorphic λ-calculus*. Available at http://www.arxiv.org/abs/1306.5089. (Submitted).

[16] Alejandro Díaz-Caro, Giulio Manzonetto & Michele Pagani (2013): *Call-by-value non-determinism in a linear logic type discipline*. In Sergei Artemov & Anil Nerode, editors: *Logical Foundations of Computer Science*, *Lecture Notes in Computer Science* 7734, pp. 164–178, doi:10.1007/978-3-642-35722-0_12.

[17] Alejandro Díaz-Caro & Barbara Petit (2012): *Linearity in the non-deterministic call-by-value setting*. In Luke Ong & Ruy de Queiroz, editors: *Logic, Language, Information and Computation*, *Lecture Notes in Computer Science* 7456, pp. 216–231, doi:10.1007/978-3-642-32621-9_16. Available at http://arxiv.org/abs/1011.3542.

[18] Herman Geuvers, Robbert Krebbers, James McKinna & Freek Wiedijk (2010): *Pure Type Systems without Explicit Contexts*. In Karl Crary & Marino Miculan, editors: *Logical Frameworks and Meta-languages: Theory and Practice*, *Electronic Proceedings in Theoretical Computer Science* 34, pp. 53–67, doi:10.4204/EPTCS.34.6.

[19] Oltea Mihaela Herescu & Catuscia Palamidessi (2000): *Probabilistic Asynchronous π-Calculus*. In Jerzy Tiuryn, editor: *Foundations of Software Science and Computation Structures*, *Lecture Notes in Computer Science* 1784, pp. 146–160, doi:10.1007/3-540-46432-8_10. Available at http://arxiv.org/abs/cs/0109002.

[20] Ugo Dal Lago & Margherita Zorzi (2012): *Probabilistic operational semantics for the lambda calculus*. *RAIRO - Theoretical Informatics and Applications* 46(03), pp. 413–450, doi:10.1051/ita/2012012. Available at http://arxiv.org/abs/1104.0195.

[21] Michele Pagani & Simona Ronchi Della Rocca (2010): *Linearity, non-determinism and solvability*. *Fundamental Informaticae* 103(1–4), pp. 173–202, doi:10.3233/FI-2010-324.

[22] Jonghyun Park, Jeongbong Seo, Sungwoo Park & Gyesik Lee (2013): *Mechanizing Metatheory without Typing Contexts*. *Journal of Automated Reasoning*, doi:10.1007/s10817-013-9287-4.

[23] TeReSe (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

[24] Lionel Vaux (2009): *The algebraic lambda calculus*. *Mathematical Structures in Computer Science* 19(5), pp. 1029–1059, doi:10.1017/S0960129509990089. Available at http://hal.archives-ouvertes.fr/hal-00379750.

Proof-graphs for Minimal Implicational Logic

Marcela Quispe-Cruz
Informática PUC-Rio, Rio de Janeiro, Brazil
mcruz@inf.puc-rio.br

Edward Hermann Haeusler
Informática PUC-Rio, Rio de Janeiro, Brazil
hermann@inf.puc-rio.br

It is well-known that the size of propositional classical proofs can be huge. Proof theoretical studies discovered exponential gaps between normal or cut free proofs and their respective non-normal proofs. The aim of this work is to study how to reduce the weight of propositional deductions. In this sense, we present the formalism of *proof-graphs* for purely implicational logic, which are graphs of a specific shape that are intended to capture the logical structure of a deduction. The advantage of this formalism is that formulas can be shared resulting in the reduced proof.

In this paper, we give a precise definition of proof-graphs for the classical $\rightarrow$-fragment, together with a normalization procedure for these proof-graphs.

1 Introduction

The use of DAGs (Directed Acyclic Graphs), instead of trees or lists, for representing proofs is getting popular among proof-theoreticians. DAGs serve as a way to provide a better symmetry to the semantic of proofs ([6]), and a way to study complexity of propositional proofs and to provide more efficient theorem provers, concerning size of propositional proofs. In [1], one can find a complexity analysis of the size of Frege systems, Natural Deduction systems and Sequent Calculus concerning their tree-like and list-like representation. They show a $O(nlog(n))$ improvement in the size of the tree-like proof with respect to list-based proofs. The fact that hypothesis occurs only once in lists and more than that in trees contribute for this size relationship. On the other hand, investigations on the use of DAGs in [3], [2] and [4] show that the use of DAGs together with mechanisms of unification/substitution in proof representation have a compacting factor equivalent to cut-introduction. Of course, the use of DAGs is a good alternative to save space by means of reference instead of copying. This article shows a more interesting and diverse advantage of using DAGs for representing proofs. We show that by using DAGs for representing formulas and Natural Deduction derivations in the purely implicational minimal logic, we obtain the (weak) normalization theorem, that is in fact, a strong normalization theorem. The choice of purely implicational minimal logic ($M^{\rightarrow}$) was motivated by the fact that the computational complexity of the validity of $M^{\rightarrow}$ is PSPACE-complete and it can polynomially simulate Classical, Intuitionistic and full minimal logic ([7]) and any propositional logic with a Natural Deduction system with the subformula property ([5]). This work takes part in the investigation of propositional theorem provers able to provide short (polynomial) proofs.

In proof theory there are three main properties when studying a structural deductive system (Natural Deduction, Sequent Calculus, etc):

- Normal form: To each derivation of α from Δ there is a normal derivation of α from $\Delta' \subseteq \Delta$.
- Normalization: To each derivation of α from Δ there is a normal derivation of α from $\Delta' \subseteq \Delta$, obtained by a particular strategy of reductions application.
- Strong Normalization: To each derivation of α from Δ there is a normal derivation of α from $\Delta' \subseteq \Delta$. This normal form can be obtained by applying reductions to the original derivation in any ordering.

Submitted to:
DCM 2013

The traditional way to prove the strong normalization property for a natural deduction system is the so-called semantical method:

- Define a property $P(\pi)$ on derivations π in the Natural Deduction system;
- Prove that this property implies strong normalization, that is $\forall\pi(P(\pi) \to SN(\pi))$, where $SN(X)$ means that X is strongly normalizable;
- Prove that $\forall\pi P(\pi)$.

In the literature there are some famous and distinct examples of this property $P(X)$. (1) Prawitz uses "strong validity"; (2) Tait uses "convertibility"; (3) Jervell uses "regularity"; (4) Leivant defines "stability"; (5) Martin-Löf defines "computability"; and (6) Girard "candidate de reducibilité". It is worth noting that this kind of strategy to prove strong normalization is need in the case of the purely implicational fragment of minimal logic too. However, these semantical proofs provides no intuition on the combinatorics behind the strong normalization. Because of this feature, proof-theoreticians searched alternative combinatorial proofs for the strong normalization theorem. The strategy of showing that there is a worst sequence of reduction and this sequence always produces a normal derivation is a good representative of an alternative, and syntactic proof of Strong Normalization.

Other approaches includes assigning quite complicated measures to derivation, proving that reductions decreases the measure assigned to the reduced derivation and by an inductive argument providing a proof of strong normalization. In this article we show how to represent $M^{\to}$ derivations in a DAG-like form; how to reduce (eliminate maximal formulas) the derivation in DAG-like form; and; that by counting the number of maximal formulas in the original derivation, a normalization theorem can be proved. Strong normalization is then a direct consequence of normalization, since the decrease in the complexity measure of the derivation is obtained by means of any reduction application. We believe that the intuition behind our result comes from the fact that our DAG representation of Natural Deduction derivations uses only one node for any two identical formulas occurring in the original Natural Deduction derivation. That is, propositional variables must label an unique node in the DAG.

2 Mimp-graphs

Mimp-graphs consist of nodes representing (labelled by) logical constants occurrences, propositional variables and inference rules ($\to$-I and $\to$-E). The edges serve to connect syntactically these components. The edges are labelled with tokens that better identify the connection between the respective rule nodes and formula nodes. Formulas called formula graph may occur only once in the graph and are built inductively consisting of formula nodes. Sub-formulas of a formula called formula sub-graphs are indicated by outgoing edges with labels l (left) and r (right). The different types of formula nodes are shown in Figure 1.

We have rule nodes that, like rules in Natural Deduction, require the correct number of premises. Premises of a rule node are indicated by ingoing edges, and the node also has an edge to a conclusion formula, thus the number of ingoing and outgoing edges should be fixed. The right side of the Figure 1 shows the rule nodes $\to$-I (implication introduction) and $\to$-E (implication elimination). It is important to note that as consequence of the fact the discharging of hypotheses may be vacuous. This case in a mimp-graph is represented by a disconnected graph, where the discharged formula node is not linked to the conclusion of the rule by any directed path.

In the rule nodes, formulas are re-used, which is simply done by putting several arrows towards it, hence the number of ingoing/outgoing edges with label p (premise), M (major premise), m (minor

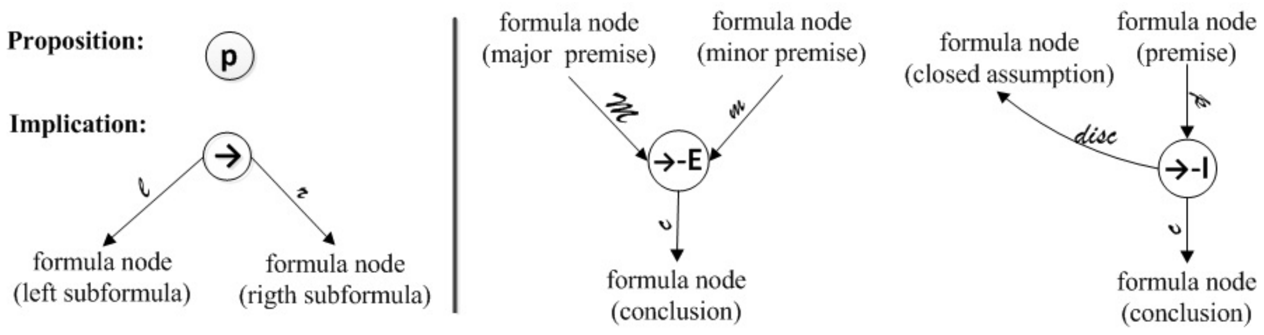

Proposition:
p
Implication:
l
r
formula node (left subformula)
formula node (rigth subformula)
formula node (major premise)
formula node (minor premise)
M
m
→-E
c
formula node (conclusion)
formula node (closed assumption)
formula node (premise)
disc
p
→-I
c
formula node (conclusion)

Figure 1: Types of formula nodes of the formula graph and types of rule nodes of the mimp-graph

$$\dfrac{\dfrac{\dfrac{\dfrac{[p]^1 \qquad [p\to q]^2}{q}\ \to\text{-}E \qquad [q\to r]^3}{r}\ \to\text{-}E}{p\to r}\ (\to\text{-}I,1)}{(q\to r)\to(p\to r)}\ (\to\text{-}I,3)$$

$$\Downarrow trans$$

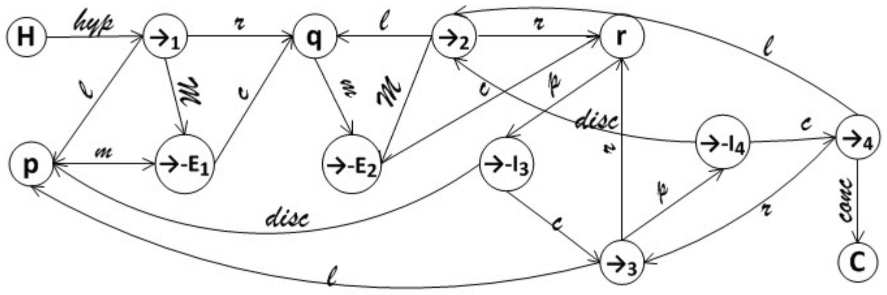


Figure 2: The transition from a natural deduction proof to a mimp-graph

premise) and c (conclusion) coming or going to a formula node should be arbitrary. To make all this a bit more intuitive we give an example of a mimp-graph in Figure 2, which can be seen as a derivation of $(q\to r)\to(p\to r)$ from $(p\to q)$. Indices of discarded hypotheses are replaced by an additional edge that goes marked with the label: *disc* (discharge).

The formula nodes in the graph (Figure 2) are labelled with propositional letters p, q and r, the connective $\to$; the rule nodes are labelled with $\to$-E and $\to$-I. The general idea is that there is an inferential order between rule nodes which gives a logical derivability order to the graph: the formula node labelled $\to_4$, linked to the delimiter node C by an edge labelled *conc*, is the root node; and the conclusion of the proof which the graph represents. Besides, the node $\to_1$, linked to the delimiter node H by the edge labelled *hyp* (hypothesis) in the graph is representing the premise $(p\to q)$.

Now we shall give a formal definition of mimp-graph.

Definition 1. L *is the union of three sets of labels types:*

- R-Labels *is the set of inference labels:* $\{\to\text{-}I_n/n\in\mathbb{Z}\}\cup\{\to\text{-}E_m/m\in\mathbb{Z}\}$,
- F-Labels *is the set of formula labels:* $\{\to_i/i\in\mathbb{N}\}$ *and the propositional letters* $\{p,q,r,\ldots\}$*, and*

- E-Labels *is the set of edge labels: $\{l$ (left), r (right), p (premise), m (minor premise), M (major premise), c (conclusion), disc (discharge), conc (final conclusion), ass (open assumption)$\}$*
- D-Labels *is the set of delimiter labels: $\{H, C\}$.*

Definition 2. Mimp-graph G *is a directed graph* $\langle$V, E, L, l_V, $l_E\rangle$ *where:* V *is a set of nodes,* E *is a set of* edges, L *is a set of labels,* $\langle v \in$ V, $t \in$ L, $v' \in$ V$\rangle$ *such that v is called the source of the edge and v' the target, l_V is a labelling function from* V *to* L, *l_E is a labelling function from* E *to* L.

Mimp-graph is defined inductively as follows:

Basis *One formula graph G_1 with root node α_m* [1]*, then the graph $G_2 = G_1$ with the delimiter nodes H and C, and the edges $(\alpha_m, conc, C)$ and (H, hyp, α_m) is a mimp-graph.*

$\rightarrow$-E *If G_1 and G_2 are mimp-graphs, and, $G_1 \oplus G_2$* [2] *contains the edge $(\rightarrow_q, l, \alpha_m)$ and the two nodes $\rightarrow_q$ and α_m linked to the delimiter node C then the graph $G_2 := G_1 \oplus G_2$ with*

1. the removal of the ingoing edges in the node C;

2. a rule node $\rightarrow$-E_i at the top level;

3. the edges: $(\alpha_m, m, \rightarrow\text{-}E_i)$, $(\rightarrow_q, M, \rightarrow\text{-}E_i)$, $(\rightarrow\text{-}E_i, c, \beta_n)$ and $(\beta_n, conc, C)$ is a mimp-graph (see the left-hand part of Figure 3).

$\rightarrow$-I *If G_1 is a mimp-graph and contains a node β_n linked to the delimiter node C and the node α_m linked to the delimiter node H, then the graph $G_2 := G_1$ with*

1. the removal of the edges to $(\beta_n, conc, C)$ and (H, hyp, α_m);

2. a rule node $\rightarrow$-I_j at the top level;

3. a formula node $\rightarrow_t$ linked to the delimiter node C by an edge $(\rightarrow_t, conc, C)$;

4. the edges: $(\rightarrow_t, l, \alpha_m)$, $(\rightarrow_t, r, \beta_n)$, $(\beta_n, p, \rightarrow\text{-}I_j)$, $(\rightarrow\text{-}I_j, c, \rightarrow_t)$, and $(\rightarrow\text{-}I_j, disc, \alpha_m)$ is a mimp-graph (see the right-hand part Figure 3; the α_m-node is discharged*).*

$\rightarrow$-I-v [3] *If G_1 is a mimp-graph, and G is a formula graph with root node α_m, and G_1 contains a node β_n linked to the delimiter node C , then the graph $G_2 := G_1 \oplus G$ with*

1. the removal of the edges to $(\beta_n, conc, C)$;

2. a rule node $\rightarrow$-I_j at the top level;

3. a formula node $\rightarrow_t$ linked to the delimiter node C by an edge $(\rightarrow_t, conc, C)$;

4. the edges: $(\rightarrow_t, l, \alpha_m)$, $(\rightarrow_t, r, \beta_n)$, $(\beta_n, p, \rightarrow\text{-}I_j)$, $(\rightarrow\text{-}I_j, c, \rightarrow_t)$, and $(\rightarrow\text{-}I_j, disc, \alpha_m)$ is a mimp-graph.

Lemma1 enables us to prove that a given graph G is a mimp-graph without explicitly supplying a construction. Among other things, basically says that we have to check that each node of G is of one of the possible types that generate the Basis, $\rightarrow$-E, $\rightarrow$-I and $\rightarrow$-I-v construction cases of Definition 2.

Lemma 1. *G is a mimp-graph if and only if the following hold:*

1. *There is an inferential order $>$ in all rule nodes of the mimp-graph,*
2. *Every node N of G is of one of the following six types:*

[1]We will use the terms α_m, β_n and γ_r to represent the principal connective of the formula α, β and γ respectively.

[2]The operation $G_1 \oplus G_2$ collapses into one the nodes of the graph G_1 with the nodes of G_2 that have the same label, and collapses edges with the same source, target and label into one.

[3]the v stands for vacuous, this case of the rule $\rightarrow$-I discharges a hypothesis vacuously

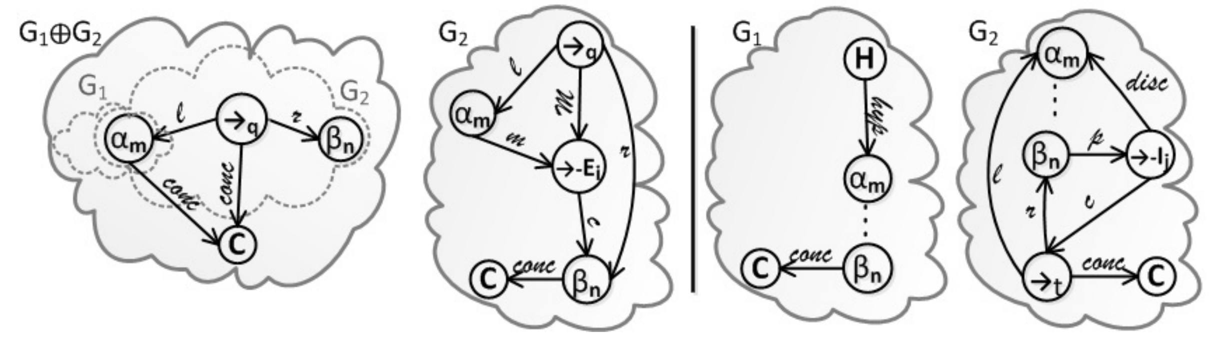


Figure 3: The rules $\rightarrow$-E and $\rightarrow$-I of mimp-graphs

L *N is labelled with one of the propositional letters: $\{p, q, r, ...\}$ indexed or not. N has not outgoing edges l and r. N has at least one outgoing edges with label p, m or M; or it has at least one ingoing edges with label c or disc. And it has at most one ingoing edges with label l or r.*

F *N has label $\rightarrow_n$ and has exactly two outgoing edges with label: l and r. N has at least one outgoing edges with label p, m or M; or it has at least one ingoing edges with label c or disc. And it has at most one ingoing edges with label l or r.*

E *N has label $\rightarrow$-E_i and has exactly one outgoing edge ($\rightarrow$-E_i, c, β_n), where β_n is a node type* L *or* F. *N has exactly two ingoing edges (α_m, m, β_n) and $(\rightarrow_q, M, \rightarrow\text{-}E_i)$, where α_m is a node type* L *or* F. *There are two outgoing edges from the node $\rightarrow_q$: $(\rightarrow_q, l, \alpha_m)$ and $(\rightarrow_q, r, \beta_n)$.*

I *N has label $\rightarrow$-I_j, has one outgoing edges ($\rightarrow$-I_j, $c, \rightarrow_t$), and at most outgoing edge $(\rightarrow\text{-}I_j, disc, \alpha_m)$. N has exactly one ingoing edge: $(\beta_n, p, \rightarrow\text{-}I_j)$, where β_n is a node type* L *or* F. *There are two outgoing edges from the node $\rightarrow_t$: $(\rightarrow_t, l, \alpha_m)$ and $(\rightarrow_t, r, \beta_n)$.*

H *N has label* H *and has only outgoing edges hyp.*

C *N has label* C *and has exactly one ingoing edge conc.*

Proof. $\Rightarrow$: By induction on the construction of mimp-graph (Definition 2). For every construction case for mimp-graphs we have to check the three properties stated in Lemma. Property (2) is immediate. For property (1), we know from the induction hypothesis that there is an inferential order $>$ on rule nodes of the mimp-graph. In the construction cases $\rightarrow$-I, $\rightarrow$-I-v or $\rightarrow$-E, we make the new rule node that is introduced highest in the $>$-ordering, which yields an inferential ordering on rule nodes. In the construction case $\rightarrow$-E, when we have two inferential orderings, $>_1$ on G_1 and $>_2$ on G_2. Then $G_1 \oplus G_2$ can be given an inferential ordering by taking the union of $>_1$ and $>_2$ and in addition putting $n > m$ for every rule node n, m such that $n \in G_1, m \in G_2$.

$\Leftarrow$: By induction on the number of rule nodes of G. Let $>$ be the topological order that is assumed to exist. Let n be the rule node that is maximal w.r.t. $>$. Then n must be on the top position. When we remove node n, including its edges linked (if n is of type I) and the node type C is linked to the premise of the rule node, we obtain a graph G' that satisfies the properties listed in Lemma. By induction hypothesis we see that G' is a mimp-graph. Now we can add the node n again, using one of the construction cases for mimp-graphs: *Basis* if n is a L node or F node, $\rightarrow$-E if n is an E node, $\rightarrow$-I if n is an I node. □

3 Normalization for mimp-graph

In this section, we define the normalization procedure for mimp-graph which is based on traditional normalization mechanism given by Prawitz. Thus a *maximal formula* in mimp-graphs is a *→-I* followed by a →-E of the same formula graph (see Definition 3). This is the same concept of maximal formulas that exists in natural deduction derivations. That is, a formula occurrence that is the consequence of an application of an introduction rule and major premise of an application of an elimination rule. Here we only consider that derivations are represented by DAGs. We want to eliminate such maximal formula by removing of nodes and edges that are involved in the maximal formula. However, it could also happen that between the rule nodes →-I and →-E there are several other maximal formulas.

Definition 3. *A* maximal formula *m in a mimp-graph G (see Figure 4) is a sub-graph of G consisting of:*

1. *the formula nodes α_m, β_n, $\rightarrow_q$ and the rule node $\rightarrow$-I_i,*
2. *the edges: $(\rightarrow_q, l, \alpha_m)$, $(\rightarrow_q, r, \beta_n)$, $(\beta_n, p, \rightarrow\text{-}I_i)$, $(\rightarrow\text{-}I_i, c, \rightarrow_q)$, $(\rightarrow\text{-}I_i, disc, \alpha_m)$, $(\alpha_m, m, \rightarrow\text{-}E_j)$, $(\rightarrow_q, M, \rightarrow\text{-}E_j)$ and $(\rightarrow\text{-}E_j, c, \beta_n)$,*
3. *the rule node $\rightarrow$-E_j at the top level.*

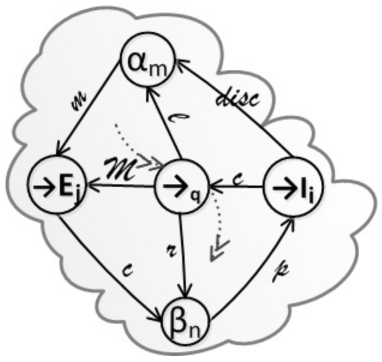


Figure 4: Maximal formula in mimp-graphs

Definition 4. *(1) For $n_i \in V$, a* p-path *in a proof-graph is a sequence of vertices and edges of the form: $n_1 \xrightarrow{l_1} n_2 \xrightarrow{l_2} \ldots \xrightarrow{l_{k-2}} n_{k-1} \xrightarrow{l_{k-1}} n_k$, such that n_1 is a hypothesis formula node, n_k is the conclusion formula node, n_i alternating between a rule node and a formula node. The edges l_i alternate between two types of edges: the first is $l_j \in \{m, M, p\}$ and the second $l_j \in \{c\}$. (2) A* branch *is an initial part of a* p-path *which stops at the conclusion formula node or at the first minor premise whose major premise is the conclusion of a rule node.*

Definition 5. *Given a mimp-graph G with a maximal formula m, eliminating a maximal formula is the following transformation of a mimp-graph, where we have two cases to consider:*

Case 1: *The maximal formula m holds the following requirements:*

1. *The formula node $\rightarrow_q$ has several ingoing edges or at least an edge $(\rightarrow\text{-}I_i, c, \rightarrow_q)$;*
2. *There is an edge $(\rightarrow_q, M, \rightarrow\text{-}E_j)$, and $\rightarrow_q$ is not premise of another rule node;*
3. *If a branch (see Definition 4) will be separated from the inferential order this branch must be insertable in the following branch, according to the order, i.e. the conclusion of this separated branch is the premise in the following branch.*

The elimination of a maximal formula is the following operation on a mimp-graph (see Figure 5):

1. *If the edge $(\rightarrow\text{-}I_i, c, \rightarrow_q)$ is the only ingoing edge to $\rightarrow_q$ then remove the edges to and from the formula node $\rightarrow_q$, and the formula node $\rightarrow_q$.*
2. *Remove the edges to and from the rule nodes $\rightarrow\text{-}I_i$ and $\rightarrow\text{-}E_j$.*
3. *Remove the nodes $\rightarrow\text{-}I_i$ and $\rightarrow\text{-}E_j$.*
4. *If a branch is separated from the inferential order, we insert this branch in the following branch, according to the order.*

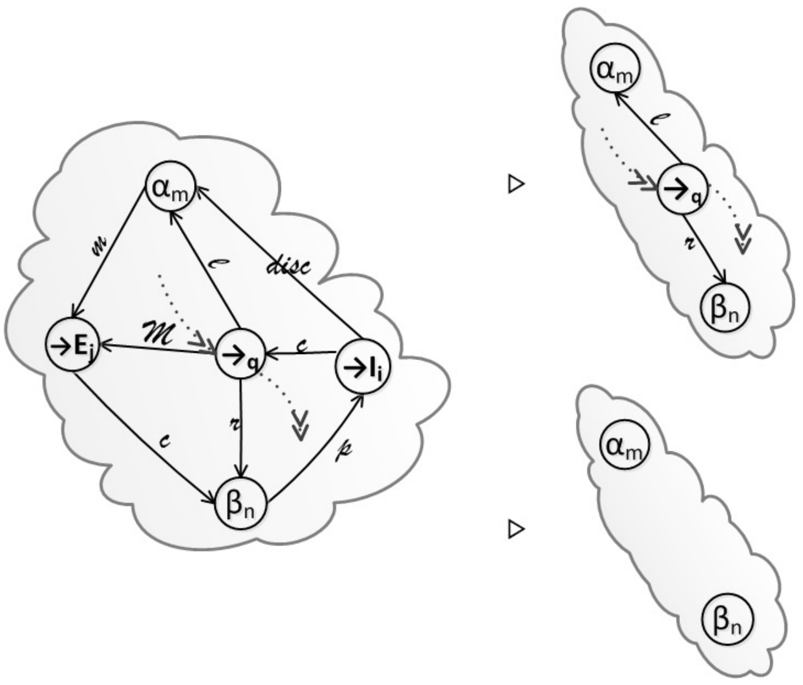


Figure 5: Case 1 of the elimination of a maximal formula in mimp-graphs

Case 2: *The maximal formula m holds the following requirements:*

1. *Between the rule nodes $\rightarrow$-I and $\rightarrow$-E there are other maximal formulas.*
2. *The formula node $\rightarrow_q$ has several ingoing edges or at least an edge $(\rightarrow\text{-}I_i, c, \rightarrow_q)$.*
3. *There is an edge $(\rightarrow_q, M, \rightarrow\text{-}E_j)$, and, the formula node $\rightarrow_q$ is the premise of zero or more of another rule nodes.*
4. *If a branch will be separated from the inferential order this branch must be insertable in the following branch, according to the order, i.e. the conclusion of this separated branch is the premise in the following branch.*

The elimination of a maximal formula is the following operation on a mimp-graph (see Figure 6, the dotted arrows are representing sets of edges):

1. *Eliminate the maximal formulas between the rule nodes $\rightarrow$-I and $\rightarrow$-E.*
2. *Finally eliminate this maximal formula.*

Note that the removal of a node $\rightarrow$-I generated by case $\rightarrow$-I-v, in the Definition 2, disconnects the graph meaning that the sub-graph hypotheses linked, by the edge m, to eliminated node $\rightarrow$-E is no longer connected to the delimiter C.

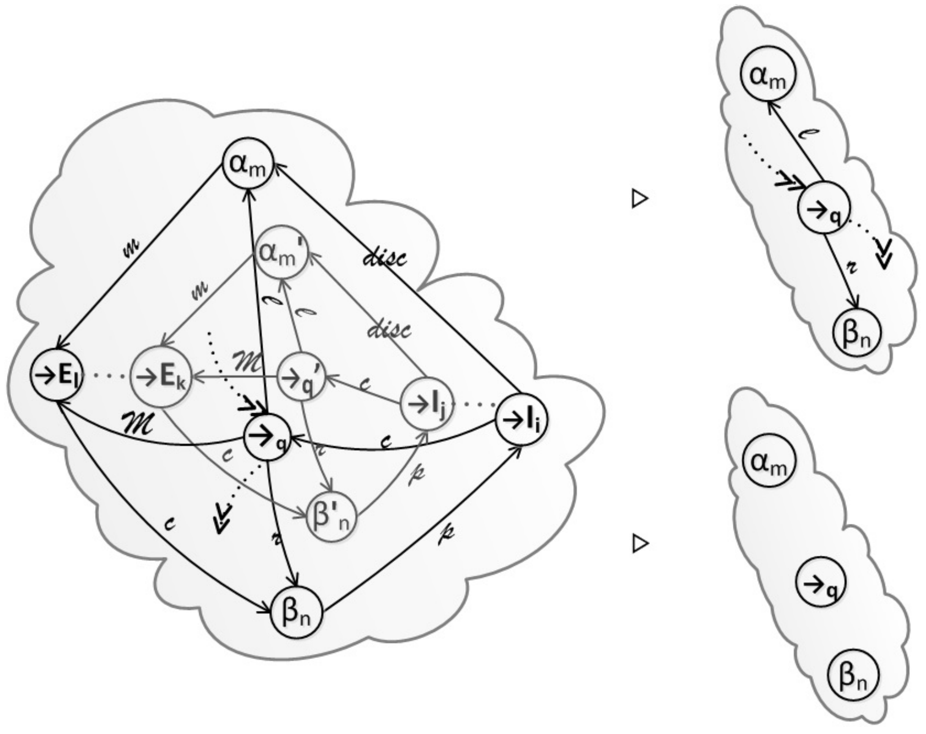


Figure 6: Case 2 of the elimination of a maximal formula in mimp-graphs

Let us show in Figure 7 an instance of this case of eliminating a maximal formula in tree form. Note that this case shows the reason why essentially our (weak) normalization theorem is directly a strong normalization theorem. The formula $\beta \rightarrow \gamma$ is not a maximal formula before a reduction is applied to eliminate the maximal formula $\alpha \rightarrow (\beta \rightarrow \gamma)$. This possibility of having hidden maximal formulas in Natural Deduction is the main reason to use more sophisticated methods whenever proving strong normalization. In mimp-graphs there is no possibility to hide a maximal formula. In this graph $\beta \rightarrow \gamma$ is already a maximal formula. If $\alpha \rightarrow (\beta \rightarrow \gamma)$ is chosen to be eliminated, by the mimp-graph normalization procedure, its reduction eliminates the $\beta \rightarrow \gamma$ too. On the other hand, the choice of $\beta \rightarrow \gamma$ to be reduced only eliminates itself. In any case the number of maximal formula decreases.

We shall construct the normalization proof for mimp-graphs. This proof is guided by the normalization measure. That is, the general mechanism from the proof determines that a given mimp-graph G should be transformed into a non-redundant mimp-graph by means of applying reduction steps and at each reduction step the measure must be decreased. The normalization measure will be the number of maximal formulas in the mimp-graph.

Theorem 1 (Normalization)**.** *Every mimp-graph G can be reduced into a normal mimp-graph G', where the hypotheses and the conclusion of G are the same as G'.*

Proof. This characteristic of preservation of the premises and conclusions of the derivation is proved naturally. Through an inspection of each elimination of maximal formula is observed that the reduction step (case 1 and case 2 of the Definition5) of the mimp-graph does not change the set of premises and conclusions (indicated by the delimiter nodes H and C) of the derivation that is being reduced.

In addition, the demonstration of this theorem has two primary requirements. First, we guarantee that through the elimination of maximal formulas in the mimp-graph, can not generate more maximal

$$\dfrac{\begin{array}{c}\Pi_2\\ \beta\end{array} \quad \dfrac{\begin{array}{c}\Pi_1\\ \alpha\end{array} \quad \dfrac{\dfrac{\begin{array}{c}[\beta]^v[\alpha]^u\\ \Pi_0\\ \gamma\end{array}}{\beta\to\gamma}\,v}{\alpha\to(\beta\to\gamma)}\,u}{\beta\to\gamma}}{\gamma} \quad \rhd \quad \begin{array}{c}\Pi_2\ \ \Pi_1\\ \beta\quad\ \alpha\\ \Pi_0\\ \gamma\end{array}$$

$$\Downarrow trans$$

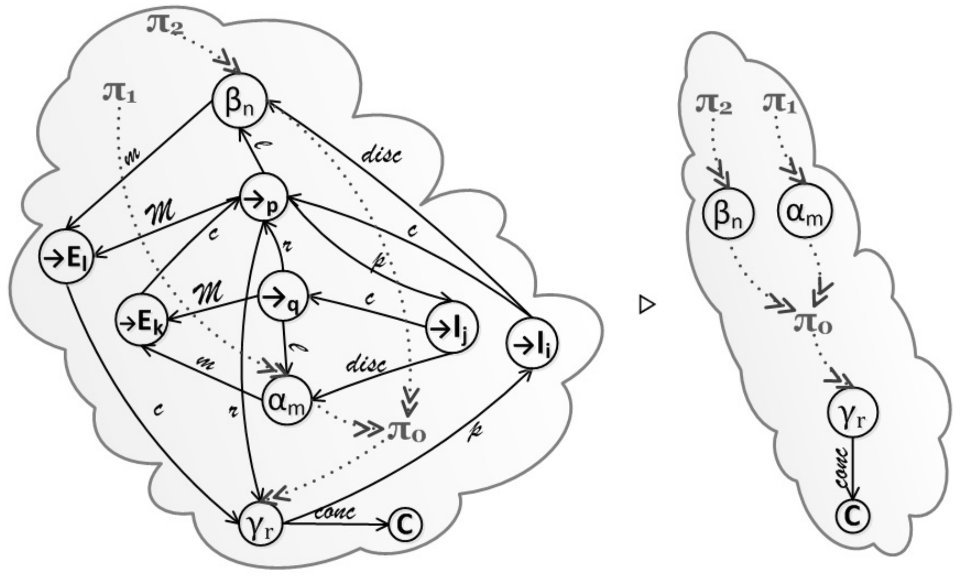


Figure 7: Eliminating a maximal formula in a natural deduction proof and its mimp-graph translation

formulas, it is a finite process. The second requirement is to guarantee that during the normalization process, the normalization measure adopted is always reduced.

The first requirement is easily verifiable through an inspection of each case in the elimination of maximal formulas. Thus, it is observed that no case produces more maximal formulas. The second requirement is established through the normalization procedure and demonstrated through an analysis of existing cases in the elimination of maximal formulas in mimp-graphs. To support this statement, it is used the notion of normalization measure, we adopt as measure the number of maximal formulas $Nmax(G)$. □

Normalization Process

We know that a specific mimp-graph G can have one or more maximal formulas represented by $M_1,...,M_n$. Thus, the normalization procedure is described by the following steps:

1. Choosing a maximal formula represented by M_k.
2. Identify the respective number of maximal formulas $Nmax(G)$.
3. Applying the respective case of eliminating maximal formula in M_k.
4. In this application one of the following three cases may occur:

a) The maximal formula is removed.

b) The maximal formula is removed but the formula node is maintained, hence $Nmax(G)$ is decreased;

c) All maximal formulas are removed.

5. We repeat this process until the normalization measure $Nmax$ is reduced to zero and G becomes a normal mimp-graph.

Since the process of the eliminating a maximal formula on mimp-graphs always ends in the elimination of at least one maximal formula, according to the case where the maximal formula is situated, and ends with the decrease in the number of vertices of the graph, we can say that this normalization theorem is directly a strong normalization theorem.

4 Conclusions

With this representation of a proof in mimp-graph get less nodes than the tree and the list representation of proofs. For the case of lists, it is enough to observe that a sub-formula of a formula is already in any graph representation of it. If both take part in the proof the size is smaller than in the mentioned representations. The ability to represent any Natural Deduction proof is preserved. Another important advantage of a compact representation of graphs is that it allows to deduce some structural properties of proof-graphs, for example based on a mimp-graph, it is easy to see an upper bound in the length of the reduction sequence to obtain a normal proof. It is the number of maximal formulas.

References

[1] Maria Luisa Bonet & Samuel R. Buss (1993): *The Deduction Rule and Linear and Near-Linear Proof Simulations*. *Journal of Symbolic Logic* 58(2), pp. 688–709.

[2] Vaston Gonçalves da Costa (2007): *Compactação de Provas Lógicas*. Ph.D. thesis, DI, PUC–Rio.

[3] M. Finger (2005): *DAG Sequent Proofs with a Substitution Rule*. In: *We will show Them – Essays in honour of Dov Gabbay 60th birthday*, *Kings College Publications* 1, Kings College, London, pp. 671–686.

[4] L. Gordeev, E. H. Haeusler & V. G. Costa (2009): *Proof compressions with circuit-structured substitutions*. *Journal of Mathematical Sciences* 158(5), pp. 645–658.

[5] E.H. Haeusler (2013): *A proof-theoretical discussion on the mechanization of propositional logics*. *Electronic Proceedings in Theoretical Computer Science Vol. 113, pp. 7-8.*

[6] Anjolina Grisi de Oliveira & Ruy J. G. B. de Queiroz (2003): *Geometry of deduction via graphs of proofs*. In: *Logic for concurrency and synchronisation*, Kluwer Academic Publisher, pp. 3–88.

[7] R. Statman (1974): *Structural Complexity of Proofs*. Stanford University.

Prefix Orders as a General Model of Dynamics

P.J.L. Cuijpers
Department of Mathematics and Computer Science
Eindhoven University of Technology
p.j.l.cuijpers@tue.nl

1 Introduction

Whenever I am confronted with a new type of dynamical system, one of the the first questions that arises is: "how does this system behave?" Also, any book that studies the dynamics of a computational system, a control system, a physical system, a biological system, etc., starts by defining in some way "what the executions of such a system look like." As an example, in automata and process theory, executions are described as runs over a transition system [5], while in control theory, executions are usually functions of time to some variable-space [6]. In hybrid and cyber-physical systems theory, these two notions have been combined by defining time as a mix of continuous and discrete steps [1].

The notion of 'a set of executions' appears to be crucial in the study of dynamical systems, and while executions are often defined as functions of time, there is still much debate on what an appropriate notion of 'time' is. For me, this was a reason to see if I could characterize the essential properties of a set of executions without considering the notion of time. Admittedly, the word 'essential' is biased towards process theory in this case, for in this paper I generalize the notion of execution of a dynamical system in such a way that computer-science notions like implementation, refinement, specification, parallel composition and branching bisimulation are still defined in a natural way.

Using category theory as a compass, I start by formally defining my 'object of study' in the next section. I give axioms that characterize the idea of a 'prefix order' on executions, and use this idea as a basis throughout the remainder of the paper. In the subsequent sections, I propose different 'structure preserving maps' to characterize the different notions from computer science mentioned above.

In this paper, I only develop a very basic theory of dynamics. Admittedly, this may raise more questions than it answers, and many possible continuations for research impose themselves immediately. In the concluding section, I sketch a number of these directions for future research in which I expect the proposed generalization will be useful.

2 Prefix Orders

In [2, 3] the notion of branching bisimulation between processes is studied on runs over a transition system, rather than directly on the transition systems themselves. The authors show that, after unfolding a transition system into its set of executions, branching bisimulation can be characterized using relations that are forward- and backward- bisimulation relations. It was observed in [3] that the resulting definition of bisimulation only uses the notion of 'prefix' on the runs, rather than requiring a notion of 'silent-steps followed by a single observable step'. This observation becomes important when developing a notion of bisimulation that works for arbitrary types of execution, in which a notion of 'next step' does not always exist. Moreover, the subsequent sections show that just capturing the notion of prefix order on executions in an order theoretic fashion already gives us a very flexible general model of dynamics.

Submitted to:
DCM 2013

In literature, the notion of 'prefix' is often defined using some notion of time. An execution is then defined as a function $e : [0,t] \to X$ from some interval $[0,t]$ over time to a set X, and another execution $f : [0,t'] \to X$ is a *prefix* of e if $t' \leq t$ and for all $\tau \in [0,t']$ it holds that $e(\tau) = f(\tau)$. This notion of prefixing leads to an order relation on executions, which on closer inspection satisfies the following axioms.

Definition 1 (Prefix order) *A* prefix order $\langle \mathbb{U}, \preceq \rangle$ *consists of a* set of executions $\mathbb{U}$ *and a* prefix relation $\preceq \subseteq \mathbb{U} \times \mathbb{U}$ *that is:*

- *reflexive:* $\forall_{a \in \mathbb{U}}\ a \preceq a$;
- *transitive:* $\forall_{a,b \in \mathbb{U}}\ a \preceq b \ \wedge\ b \preceq c \ \Rightarrow\ a \preceq c$;
- *anti-symmetric:* $\forall_{a,b \in \mathbb{U}}\ a \preceq b \ \wedge\ b \preceq a \ \Rightarrow\ a = b$;
- *downward total:* $\forall_{a,b,c \in \mathbb{U}}\ (a \preceq c \ \wedge\ b \preceq c) \ \Rightarrow\ (a \preceq b \ \vee\ b \preceq a)$;

In this definition, only the downward totality is special; the other three requirements simply say that prefixing is a *partial order* in the classical sense. Downward totality means that, although the future of a system may be branching from a given point of execution, the past is always totally ordered. In [4], this is called the *perfect recall property* of executions: at any point of execution the complete history of the system so far is remembered. Another way of looking at it, is saying that the set of executions behaves like a *tree* structure, except that it may be *dense* (in continuous systems there is no 'next' point of execution), there may be no *root* (in some systems history is infinite), and there may be multiple trees next to each other (for example because there are multiple initial states to consider).

Two important notions on a prefix order are the *future* and the *history* of an execution.

Definition 2 (History and future) *Given a prefix order $\langle \mathbb{U}, \preceq \rangle$ and an execution $u \in \mathbb{U}$, the* history *and* future *of u are defined by*

- *history:* $u^- \triangleq \{v \in \mathbb{U} \mid v \preceq u\}$;
- *future:* $u^+ \triangleq \{v \in \mathbb{U} \mid u \preceq v\}$;

A map $f : \mathbb{U} \to \mathbb{V}$ between two prefix orders is then

- *order preserving if:* $\forall_{u,u' \in \mathbb{U}}\ u \preceq u' \ \Rightarrow\ f(u) \preceq f(u')$;
- *history preserving if:* $\forall_{u \in \mathbb{U}}\ f(u^-) = f(u)^-$;
- *future preserving if:* $\forall_{u \in \mathbb{U}}\ f(u^+) = f(u)^+$;

with the obvious lifting $f(A) \triangleq \{f(a) \mid a \in A\}$ of f to subsets $A \subseteq \mathbb{U}$.

Incidentally, the well-known idea of 'computation trees' as executions (see e.g. [2, 3]) is obtained by studying only prefix orders in which each history is a finite set, while the idea of 'initial states' at which a system is turned on is captured by studying only prefix orders in which each history has a minimum. Furthermore, one should note that any history or future preserving function is also order preserving.

3 Bisimulations as history and future preserving surjections

In this section, the previously mentioned result of [2, 3], capturing branching bisimulation using futures and histories, is generalized to prefix orders. However, in contrast to [2, 3], I do not use a relational definition of branching bisimulation here, but a definition using spans (as proposed by [7]).

Definition 3 (Labeled transition system) *A* labeled transition system *is a tuple* $\langle X, A, i, \rightarrow \rangle$*, consisting of a set of states* X*, a set of observables* A*, an initial state* $i \in A$*, and a transition relation* $\rightarrow \subseteq X \times (A \cup \{\tau\}) \times X$ *with the unobservable* $\tau \notin A$*. Given* $a \in A \cup \{\tau\}$ *I write* $x \overset{a}{\rightarrow} x'$ *for* $(x, a, x') \in \rightarrow$ *and* $x_0 \overset{a}{\twoheadrightarrow} x_{n+1}$ *whenever there exists a sequence* $x_0 \ldots x_{n+1}$ *such that* $x_i \overset{\tau}{\rightarrow} x_{i+1}$ *for every* $i < n$ *and* $x_n \overset{a}{\rightarrow} x_{n+1}$*.*

Definition 4 (Run) *A* run *over a labeled transition system* $\langle X, A, i, \rightarrow \rangle$ *is a sequence* $\rho \in ((A \cup \{\tau\}) \times X)^*$ *such that, if* ρ *is not empty, it holds that* $i \overset{\rho_1(0)}{\rightarrow} \rho_2(0)$ *and* $\rho_2(n) \overset{\rho_1(n+1)}{\rightarrow} \rho_2(n+1)$ *for all* $n+1 \in dom(\rho)$*. The set of all runs is denoted* $\mathscr{R}(\rightarrow)$*, is prefix ordered in the usual way, and is observed by a function* $\pi : \mathscr{R}(\rightarrow) \rightarrow A^*$ *defined recursively as* $\pi(\varepsilon) = \varepsilon$*,* $\pi(\rho \cdot \tau) = \pi(\rho)$*, and* $\pi(\rho \cdot a) = \pi(\rho) \cdot a$*, for* $a \in A$*.*

Definition 5 (Branching Bisimulation) *Two labeled transition systems* $\langle X, A, i, \rightarrow_1 \rangle$ *and* $\langle Y, A, j, \rightarrow_2 \rangle$ *are* branching bisimilar *if there exists a relation* $\mathscr{R} \subseteq X \times Y$ *such that* $i \mathscr{R} j$ *and*

- *if* $x \mathscr{R} y$*, and* $x \overset{a}{\rightarrow}_1 x'$*, then either* $a = \tau$ *and* $x' \mathscr{R} y$*, or there exist* y', y'' *such that* $y \overset{\tau}{\twoheadrightarrow}_2 y'$ *and* $y' \overset{a}{\rightarrow}_2 y''$ *and* $x \mathscr{R} y'$ *and* $x' \mathscr{R} y''$*;*
- *if* $x \mathscr{R} y$*, and* $y \overset{a}{\rightarrow}_2 y'$*, then either* $a = \tau$ *and* $x \mathscr{R} y'$*, or there exist* x', x'' *such that* $x \overset{\tau}{\twoheadrightarrow}_1 x'$ *and* $x' \overset{a}{\rightarrow}_1 x''$ *and* $x' \mathscr{R} y$ *and* $x'' \mathscr{R} y'$*.*

Theorem 1 *Two labeled transition systems* $\langle X, A, i, \rightarrow_1 \rangle$ *and* $\langle Y, A, j, \rightarrow_2 \rangle$ *are branching bisimilar if and only if there exists a prefix order* $\langle \mathbb{U}, \preceq \rangle$ *and span* (f, g) *of history and future preserving (surjective) maps* $f : \mathbb{U} \rightarrow Runs(\rightarrow_1)$ *and* $g : \mathbb{U} \rightarrow Runs(\rightarrow_2)$ *such that* $\pi(g(u)) = \pi(f(u))$ *for every* $u \in \mathbb{U}$*.*

Note that the requirement that f and g are history and future preserving implies that they are surjective whenever their domains have a single minimum (i.e. a single initial state). However, I would like to propose the above theorem as an alternative definition for branching bisimulation in the future, and on arbitrary prefix orders surjection is not guaranteed while I feel a complete refinement should be surjective (thus guaranteeing that all initial states are related whenever there is more than one).

In the next section, I will argue that history preserving maps model refinements of a specification. As a consequence, the above theorem may be interpreted as: *two specifications are branching bisimilar if and only if they have a common refinement.* In other words, branching bisimulation is a way to define that two specifications are 'consistent' with each other.

The definition of bisimulation using spans is flexible, and can easily be adapted for multiple 'views' by considering systems that have multiple labels. For example, one can treat the observation of time and of actions separately in a system with timed runs. This separation of concerns is more difficult to achieve using the traditional definition. On the negative side, the notion of bisimulation through history and future preserving spans does not coincide in general with the notion of bisimulation through history and future preserving relations as proposed in [2, 3]. For certain pathological prefix orders (such as the 'negative natural numbers' $-\mathbb{N}$ and the 'negative countable ordinals $-\Omega$) a history and future preserving relation exists ($-\mathbb{N} \times -\Omega$) while there cannot exist a span of history and future preserving surjections. As I like the 'common refinement' interpretation of branching bisimulation, I vote for the more restrictive definition using spans in this paper.

4 Refinements as history preserving maps

In the previous section, I discussed maps that are both history and future preserving. However, the idea that elements of a prefix order represent executions puts more emphasis on the past than on the future. Even more strongly, the next theorem shows that the history of an execution in fact contains all information about that execution.

Theorem 2 *Any prefix order $\langle \mathbb{U}, \preceq \rangle$ is isomorphic to the prefix order $\langle \mathbb{U}^-, \subseteq \rangle$, with $\mathbb{U}^- = \{u^- \mid u \in \mathbb{U}\}$. I.e. there exists a bijection $f : \mathbb{U} \to \mathbb{U}^-$ such that f and f^{-1} are order preserving (and consequently history and future preserving).*

In itself, this already justifies the study of history preserving maps as a category. But further justification can be found in the observation that a history preserving map $f : \mathbb{U} \to \mathbb{V}$ models how each execution of $\mathbb{U}$ maps to a (more abstract) execution in $\mathbb{V}$. At every point of execution in $\mathbb{U}$, f tells you exactly where the system is in $\mathbb{V}$, thus showing how $\mathbb{U}$ is a refined version of the behavior in $\mathbb{V}$. Indeed, this refinement may not be complete. Therefore, the history preserving maps are suitable to describe *arbitrary refinements*, while surjective history and future preserving maps describe *complete refinements*. In the category of history preserving maps, notions like parallel composition and disjoint union arise naturally.

Definition 6 (Product) *Given a family of prefix-orders $\{\langle \mathbb{U}_i, \preceq_i \rangle \mid i \in I\}$ a* joint execution *is a* set *of tuples $H \subseteq \prod_{i \in I} \mathbb{U}_i$, modeling a history of concurrent points of execution (synchronous or interleaving), with a maximum: $\exists_{h \in H} \forall_{i \in I} H_i = h_i^-$, and no crossings: $\forall_{h,g \in H} (\forall_{i \in I} h_i \preceq_i g_i) \vee (\forall_{i \in I} g_i \preceq_i h_i)$.*
(Here $\prod$ denotes the usual Cartesian product on sets, h_i denotes the i'th element in a tuple h, and $H_i = \{h_i \mid h \in H\}$ lifts this to sets of tuples.)

The parallel composition *of this family, is the set $\|_{i \in I} \mathbb{U}_i$ of all joint executions, ordered by the relation $\sqsubseteq$, defined for all $G, H \in \|_{i \in I} \mathbb{U}_i$ by $G \sqsubseteq H \iff G \subseteq H \wedge \forall_{h \in H} \forall_{g \in G} (\forall_{i \in I}\, h_i \preceq_i g_i) \Rightarrow (h \in G)$. Together with the parallel composition, the family $\{\pi_i : (\|_{j \in I} \mathbb{U}_j) \to \mathbb{U}_i\}$ of* canonic projections *is given by $\pi_i(H) = \max(H_i)$ for every $i \in I$ and $H \in \|_{j \in I} \mathbb{U}_j$.*

Theorem 3 *The parallel composition of a family of prefix orders coincides with the categorical product in the category of history preserving maps.*

Definition 7 (Disjoint union) *Given a family of prefix-orders $\{\langle \mathbb{U}_i, \preceq_i \rangle \mid i \in I\}$ the* disjoint union *is the disjoint union on sets: $\biguplus_{i \in I} \mathbb{U}_i = \{(i,u) \mid i \in I \wedge u \in \mathbb{U}_i\}$. This set is ordered by the relation $\sqsubseteq$, defined by $(i,u) \sqsubseteq (j,v) \iff i = j \wedge u \preceq v$, and equipped with a family $\{\iota_i : \mathbb{U}_i \to \biguplus_{j \in I} \mathbb{U}_j\}$ of* canonic insertions *given by $\iota_i(u) = (i,u)$ for all $i \in I$ and $u \in \mathbb{U}_i$.*

Theorem 4 *The disjoint union of a family of prefix orders coincides with the categorical co-product of this family in the category of history preserving maps.*

Finally, combining the two categories in the tradition of process algebra, branching bisimulation using spans turns out to be a congruence with respect to the two constructs discussed above.

Theorem 5 *If there exists a span of history and future preserving surjections between prefix orders $\mathbb{U}$ and $\mathbb{V}$, then for any prefix order $\mathbb{X}$ there exists a span of history and future preserving surjections between $\mathbb{U} \| \mathbb{X}$ and $\mathbb{V} \| \mathbb{X}$, and between $\mathbb{U} \uplus \mathbb{X}$ and $\mathbb{V} \uplus \mathbb{X}$.*

5 Discussion and Concluding remarks

I have shown that dynamical systems can be modeled as a set of executions under their natural prefix ordering, and that history preserving maps represent the refinement of a specification, thus allowing refinements between various types of dynamics in one unified framework. Furthermore, if refinements are complete in the sense that all and only specified behavior is refined, then the corresponding maps are surjective and future preserving.

One of the next steps, is to deal with structured operational semantics in a categorical fashion. Is it possible to create maps from any operation defined using structured operational semantics to the components it depends on? In general, the composition of two systems does not lead to a refinement, so there will not simply be a history preserving map. For example, the system $\mathbb{X} \uplus \mathbb{Y}$ does not have natural maps back to $\mathbb{X}$ and $\mathbb{Y}$. However, there are natural *partial* history preserving maps from $\mathbb{X} \uplus \mathbb{Y}$ to $\mathbb{X}$ and $\mathbb{Y}$. From the point of view of $\mathbb{X}$, the *composition* $\mathbb{X} \uplus \mathbb{Y}$ is a combination of *refinement* and *specification*. The newly specified part is therefore undefined in the map to $\mathbb{X}$, while the refinement is mapped in a history preserving way. For the study of operational semantics in a category theoretic way, I therefore expect that partial history preserving maps may be helpful.

Another possible step, is to add more structure to the notion of prefix order, thus becoming less general but more applicable. Prefix orders really only model the dynamical properties of a system. If one would like to study timing, continuity, energy, or other properties, an observation map (like the one used in section 3) is needed. Incidentally, the map used in section 3 is itself a history preserving map, but other types of maps are conceivable as well. For example, if $\pi : \mathbb{U} \to Q$ and $\pi : \mathbb{V} \to Q$ map the executions of two systems to some (partially ordered) quantity Q, one could define that $\mathbb{U}$ is an *over-approximation* of $\mathbb{V}$ if there is a history and future preserving surjection $f : \mathbb{U} \to \mathbb{V}$ such that $\pi(f(u)) \leq \pi(u)$ for every $u \in \mathbb{U}$. Furthermore, the idea of *prefixing* is intimately coupled with the notion of *concatenation*, since prefixing is also often defined as: $x \preceq z$ iff $\exists_y\, x \cdot y = z$. It seems therefore reasonable to also study which *semigroups* $\langle \mathbb{U}, \cdot \rangle$ admit a natural prefix order. Finally, one could also study probabilistic systems by imposing a measure on the anti-chains of the prefix order, or one could study continuous systems by making using of the natural interval topology on prefix orders, and consider continuous maps between a prefix order and some physical variable.

In conclusion, adding observations in order to study different types of dynamical systems is reminiscent of the definition of executions as functions of time. Looking back, perhaps I did not succeed in eliminating the notion of time from our modeling paradigm after all. In stead, one could say I did succeed in capturing, in an order theoretic way, the notion of a dynamical system as a function of *branching time*.

Acknowledgements go to Harsh Beohar, Erik de Vink and Ruurd Kuiper for their continuing support.

References

[1] J.M. Davoren & P. Tabuada (2007): *On Simulations and Bisimulation of General Flow Systems*. In G. Buttazo A. Bemporad, A. Bicchi, editor: *Hybrid Systems: Computation and Control, 10th International Conference, HSCC 2007*, *Lecture Notes in Computer Science* 4416, Springer-Verlag, pp. 145–158.

[2] R. De Nicola, U. Montanari & F. Vaandrager (1990): *Back and forth bisimulations*. *CONCUR'90 Theories of Concurrency: Unification and Extension*, pp. 152–165.

[3] R.J. van Glabbeek (2001): *Current Trends in Theoretical Computer Science; Entering the 21st Century*, chapter What is Branching Time Semantics and Why to Use It?, pp. 469–479. World Scientific.

[4] J.Y. Halpern & K.R. O'Neill (2008): *Secrecy in Multiagent Systems*. *ACM Trans. Inf. Syst. Secur.* 12, pp. 5:1–5:47, doi:http://doi.acm.org/10.1145/1410234.1410239. Available at `http://doi.acm.org/10.1145/1410234.1410239`.

[5] T. Basten J.C.M. Baeten & M.A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

[6] J.W. Polderman & J.C. Willems (1998): *Introduction to Mathematical Systems Theory: A Behavioural Approach*. *Texts in Applied Mathematics* 26, Springer-Verlag.

[7] G. Winskel & M. Nielsen (1995): *Handbook of logic in computer science (vol. 4)*. chapter Models for concurrency, Oxford University Press, Oxford, UK, pp. 1–148. Available at `http://portal.acm.org/citation.cfm?id=218623.218630`.

Causal Dynamics of Simplicial Complexes: the 2-dimensional case

Pablo Arrighi
Université de Grenoble, LIG, 220 rue de la chimie, 38400 Saint-Martin-d'Hères, France
Université de Lyon, LIP, 46 allée d'Italie, 69008 Lyon, France [*]
parrighi@imag.fr

Simon Martiel
Université Nice-Sophia Antipolis, I3S, 2000 routes des Lucioles, 06900 Sophia Antipolis, France [†‡]
martiel@i3s.unice.fr

We formalize the intuitive idea of a labelled discrete surface which evolves in time, subject to two natural constraints: the evolution does not propagate information too fast; and it acts everywhere the same.

1 Introduction

Various generalizations of cellular automata, such as stochastics [5], asynchronous [10] or non-uniform cellular automata [6], have already been studied. In [1, 2, 3, 4] the authors, together with Dowek and Nesme, generalize Cellular Automata theory to arbitrary, time-varying graphs. I.e. they formalize the intuitive idea of a labelled graph which evolves in time, subject to two natural constraints: the evolution does not propagate information too fast; and it acts everywhere the same. Some fundamental facts of Cellular Automata theory carry through, for instance these "causal graph dynamics" admit a characterization as continuous functions.

The motivation for developing these Causal Graph Dynamics (CGD) was to "free Cellular Automata off the grid", so as to be able to model any situation where agents interact with their neighbours synchronously, leading to a global dynamics in which the states of the agents can change, but also their topology, i.e. the notion of who is next to whom. In [1, 2, 3] two examples of such situations are mentioned. The first example is that of a mobile phone network: mobile phones are modelled as vertices of the graph, in which they appear connected if one of them has the other as a contact. The second example is that of particles lying on a smooth surface and interacting with one another, but whose distribution influences the topology the smooth surface (cf. Heat diffusion in a dilating material, or even discretized General Relativity [11]). CGD seems quite appropriate for modelling the first situation (or at least a stochastic version of it).

Modelling the second situation, however, is not a short-term perspective. One of the several difficulties we face is that having freed Cellular Automata off the grid, we can no longer interpret our graphs as a surface, in general. There are, however, a number of formalisms for describing discretized surfaces which are very close to graphs (Abstract simplicial complexes, CW-complexes…[8]). These work by gluing triangles alongside so as to approximate any smooth surface. Relying upon these formalisms,

*This work was supported by the French National Research Agency, ANR-10-JCJC-0208 CausaQ grant
†This work was supported by the John Templeton Foundation, grant ID 15619
‡This work was supported by the French National Research Agency, ANR-10-JCJC-0208 CausaQ grant

can we formalize the idea of a labelled discrete surface which evolves in time — again subject to the constraints that evolution does not propagate information too fast and acts everywhere the same? Can we achieve this by just modelling each triangle as a vertex, and each gluing of two triangles as an edge, and then evolve the graph according to a CGD?

Notice that one could argue that simplicial complexes are not the simplest objects one could use to represent surfaces: planar graphs may seem more natural to some. Our choice is motivated by two reasons. First, the notion of planar graphs can only be used to represent two-dimensional surfaces, and would be limitating when generalizing to higher dimensions (see further work). Second, in a planar graph, the degree of each vertex is not bounded, and thus such graphs would not fit in our model. In order to change this we would have needed to artificially bound this degree by some constant d and lose the generality of planar graphs.

This paper tackles the question of how to give a rigorous definition of "Causal Dynamics of Simplicial Complexes", focussing on the 2D case for now. It investigates whether CGD can be readily adapted for this purpose, i.e. whether CGD can be "tied up again to discrete 2D surfaces". It will turn out that this can be done at the cost of two additional restrictions, i.e. a CGD must be rotation-commuting and bounded-star preserving in order to be a valid Causal Dynamics of Simplicial Complexes. The first restriction allows us to freely rotate triangles. The second requirement allows us to map geometrical distances into graph distances. Both are decidable.

2 Complexes as graphs

Correspondence. Our aim is to define a Cellular Automata-like model of computation over 2*D simplicial complexes*. For this purpose, it helps to have a more combinatorial representation of these complexes, as graphs. The straightforward way is to map each triangle to a vertex, and each facet of the triangle to an edge. The problem, then, is that we can no longer tell one facet from another, which leads to ambiguities (see Fig. 1 *Top row.*).

A first solution is to consider 2*D coloured simplicial complexes* instead. In these complexes, each of the three facets of a triangle has a different colour amongst $\{a,b,c\}$. Now each triangle is again mapped to a vertex, and each facet of the triangle to an edge, but this edge holds the colours of the facets it connects at its ends (see Fig. 1 *Bottom row.*). We recover [1, 2, 3] the following definition.

Definition 1 (Graph) *A labeled* graph *G is given by*

- *A (at most countable) subset $V(G)$ of V, whose elements are called* vertices *and where V is the uncountable set of all possible vertex name.*
- *A finite set $\pi = \{a,b,c\}$, whose elements are called* ports.
- *A set $E(G)$ of non-intersecting two element subsets of $V(G) : \pi$, whose elements are called edges. Symbol $:$ stands for the cartesian product. An edge $\{u : p, v : q\}$ is to be read "There is an edge linking port p of vertex u and port q of vertex v".*
- *A function $\sigma : V(G) \to \Sigma$ associating to each vertex v some label $\sigma(v)$ in a finite set Σ.*

The set of labeled graphs with labels in Σ is denoted $\mathcal{G}_{\pi,\Sigma}$ and the set of disks of radius r is denoted $\mathcal{D}^r_{\pi,\Sigma}$. We similarly define the set of (unlabelled) graphs and denote it $\mathcal{G}_\pi$.

This notion of coloured simplicial complex is not so common, however. It is more common to consider a version of coloured complexes where triangles can rotate freely, i.e. where we can permute the colours: a for b, b for c, c for a, so that each triangle has a cyclic ordering of its facets but no

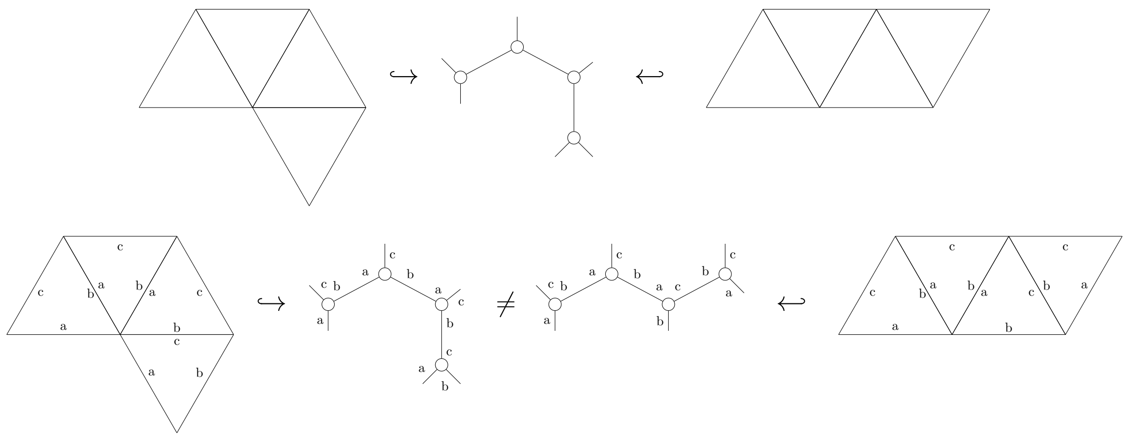


Figure 1: Complexes as graphs. *Top row*. The straightforward way to encode complexes as graphs is ambiguous. *Bottom row*. Encoding coloured complexes instead lifts the ambiguity. However, the fact that the extreme triangles share one point or not, is less obvious in the graph representation.

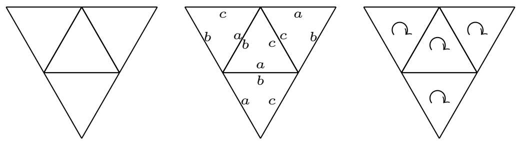


Figure 2: Complexes, Coloured complexes, Oriented Complexes

privileged facet a. The cyclic ordering is then interpreted an orientation: when two facets are glued together in the complex, their orientation must be opposed, so that the two adjacent triangles have the same orientation. This leads to *oriented* $2D$ *simplicial complexes*. Fig. 2 summarizes the three kinds of $2D$ simplicial complexes we have mentioned. Definition 1 captured $2D$ coloured complexes as graphs. How can we capture oriented $2D$ simplicial complexes as graphs?

We need to define rotations of the vertices of the graphs in a way that corresponds to rotating the triangles of coloured complexes. Namely, vertex rotations simply permute the ports of the vertex, whilst preserving the rest of the graph:

Definition 2 (Vertex Rotation) *Let p_{ports} be some cyclic permutation over $\{a,b,c\}$, and p_{labels} be some bijection from Σ to itself such that $p^3_{\text{labels}} = id$. Let G be a graph and $u \in V(G)$ one of its vertices. Then $r_u G = G'$ is such that $V(G') = V(G)$ and:*

- $\{v:i,w:j\} \in E(G) \wedge v \neq u \wedge w \neq u \Leftrightarrow \{v:i,w:j\} \in E(G')$.
- $\{u:i,v:j\} \in E(G) \Leftrightarrow \{u:p_{\text{ports}}(i),v:j\} \in E(G')$.
- $\sigma'(u) = p_{\text{labels}}(\sigma(u))$, *whereas* $\sigma'(v) = \sigma(v)$ *for* $v \neq u$.

(From now on in order to simplify notations we will drop all labels $\sigma(.) \in \Sigma$, though all the results of this paper carry through to labelled graphs.)

Who is next to whom? On the one hand in the world of $2D$ simplicial complexes, two simplices are adjacent if they share a point. On the other hand in the world of graphs, two vertices are adjacent if they share an edge. These two notions do not coincide, as shown in Fig. 1. The Fig. also shows that two triangles share a point if and only if their corresponding vertices are related by a monotonous path:

Definition 3 (Alternating paths) *Let $\Pi = \{a,b,c\}^2$. We say that $u \in \Pi^*$ is a* path *of the graph G if and only if there is a sequence u of pairs of ports $q_i p_i$ such that it is possible to travel in the graph according to this sequence, i.e. there exists $v_0,\dots,v_{|u|} \in V(G)$ such that for all $i \in \{0\dots|u|-1\}$, one has $\{v_i : q_i, v_{i+1} : p_i\} \in E(G)$, with $u_i = q_i p_i$. We say that a path $u = q_0 p_0 \dots q_{|u|} p_{|u|}$ alternates at $i = 0\dots|u|-2$ if either $p_i = q_{i+1}+1$ and $p_{i+1} = q_{i+2}-1$, or $p_i = q_{i+1}-1$ and $p_{i+1} = q_{i+2}+1$. A path is k-*alternating *if and only if it has exactly k alternations. A path is* monotonous *if and only if it does not alternate.*

Thus distance one in complexes is characterized by the existence of a 0-alternating path. More generally, distance $k+1$ in complexes is characterized by the existence of a k-alternating path. Recall that our aim is to define a CA-like model of computation over these complexes. In CA models, each cell must have a bounded number of neighbours (or a bounded "star" in the vocabulary of complexes). This bounded-density of information hypothesis [7] is the first justification for the following restriction upon the graphs we will consider:

Definition 4 (Bounded-star Graphs) *A graph G is* bounded-star *of bound s if and only if is monotonous paths are of length less or equal to s.*

Notice that the property is stable under rotation. A further justification for this restriction will later be given.

3 Causal Graph Dynamics

We now provide the essential definitions of CGD, through their constructive presentation, namely as localizable dynamics. We will not detail nor explain nor motivate these definitions in order to avoid repetitions with [1, 2, 3]. Still, notice that in [1, 2, 3] this constructive presentation is shown equivalent to an axiomatic presentation of CGD, which establishes the full generality of this formalism. The bottom line is that these definitions capture all the graph evolutions which are such that information does not propagate information too fast and which act everywhere the same.

Definition 5 (Local Rule) *A function $f : \mathcal{D}^r_\pi \to \mathcal{G}_\pi$ is called a local rule if there exists some bound b such that:*

- *For all disk D and $v' \in D$, $v' \in V(f(D)) \Rightarrow v' \subseteq V(D).\{\varepsilon, 1, ..., b\}$.*
- *For all graph G and all disks $D_1, D_2 \subset G$, $f(D_1)$ and $f(D_2)$ are consistent.*
- *For all disk D and all isomorphism R, $f(R(D)) = R^*(f(D))$, with $R^*(\{u.i, v.j, ...\}) = \{R(u).i, R(v).j, ...\}$.*

Definition 6 (Localizable Dynamics, a.k.a CGD) *[1, 2, 3] A function F from $\mathcal{G}_\pi$ to $\mathcal{G}_\pi$ is a* localizable dynamics*, or CGD, if and only if there exists r a radius and f a local rule from $\mathcal{D}^r_\pi$ to $\mathcal{G}_\pi$ such that for every graph G in $\mathcal{G}_{\Sigma,r}$,*

$$F(G) = \bigcup_{v \in G} f(G^r_v).$$

CGD act on arbitrary graphs. To compute the image graph, they can make use of the information carried out by the ports of the input graph. Thus, they can readily be interpreted as "Causal Dynamics of Coloured Simplicial Complexes". But what we are really interested in "Causal Dynamics of Bounded-star Oriented Simplicial Complexes", which we will call "Causal Complexes Dynamics" for short.

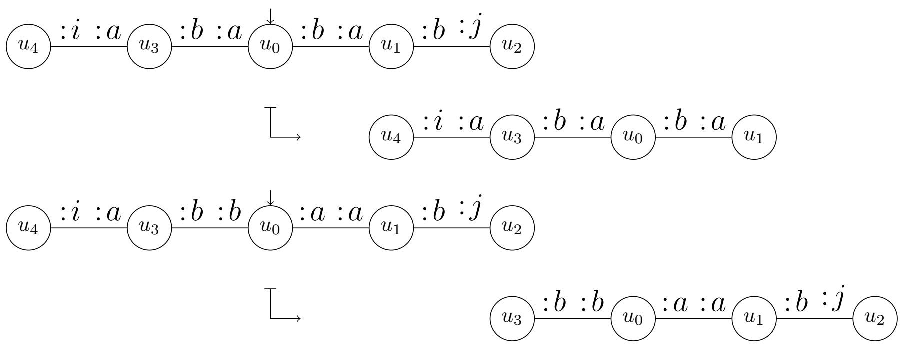


Figure 3: A non-rotation commuting local rule induces a rotation commuting CGD.

4 Causal Complexes Dynamics

This section formalizes Causal Complexes Dynamics (CCD).
Rotation-commutating. First, we will restrict CGD so that they may use the information carried out by ports, but only as far as it defines an orientation. Formally, this means restricting to dynamics which commute with graphs rotations.

Definition 7 (Rotation-Commuting function) *A function F from $\mathcal{G}_\pi$ to $\mathcal{G}_\pi$ is rotation-commuting if and only if for all graph G and all sequence of rotations $\overline{r}$ there exists a sequence of rotations $\overline{r}^*$ such that $F(\overline{r}G) = \overline{r}^* F(G)$. Such an $\overline{r}^*$ is called a conjugate of $\overline{r}$. The definition extends naturally to functions from $\mathcal{D}_\pi$ to $\mathcal{G}_\pi$.*

The next question is "When is a CGD rotation-commuting?". More precisely, can we decide, given the local rule f of a CGD F, whether F is rotation-commuting? The difficulty is that being rotation-commuting is a property of the global function F. Indeed, a first guess would be that F is rotation-commuting if and only if f is rotation-commuting, but this turns out to be false.

Example 1 *(identity function) . Consider the local rule of radius* 1 *over graphs of degree* 2 *which acts as the identity in every cases but those given in Fig. 3. Because of these two cases, the local rule makes use the information carried out by the ports around the center of the neighbourhood. It is not rotation-commuting. Yet, the CGD it induces is just the identity, which is trivially rotation-commuting.*

Thus, unfortunately, some rotation-commuting F can be induced by a non-rotation-commuting f. Yet, fortunately, any rotation-commuting F can be induced by a rotation-commuting f.

Theorem 1 *Let F be a localizable dynamics. F is rotation-commuting if and only if there exists a rotation-commuting local rule f which induces F.*

Proposition 1 (Decidability of rotation commutation) *Given a local rule f, it is decidable whether f is rotation-commuting.*

Bounded-star preserving. Second, we will restrict CGD so that they preserve the property of a graph being bounded-star. Indeed, we have explained in Section 2 that the graph distance between two vertices does not correspond to the geometrical distance between the two triangles that they represent. By modelling CCD via CGD, we are guaranteeing that information does not propagate too fast with respect to the graph distance, but not with respect to the geometrical distance. The fact that the geometrical

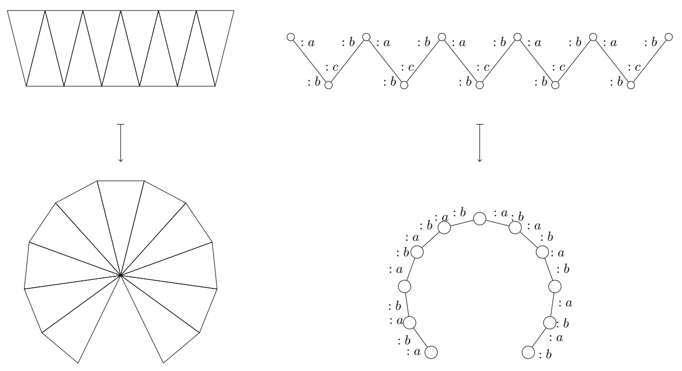


Figure 4: An unwanted evolution: sudden collapse in geometrical distance. *Left column:* in terms of complexes. *Right column.* In terms of graph representation.

distance is less or equal to the graph distance is falsely reassuring: the discrepancy can still lead to an unwanted phenomenon as depicted in Fig. 4.

Of course we may choose not to care about geometrical distance. But if we do care, then we must make the assumption that graphs are bounded-star. This assumption will not only serve to enforce the bounded-density of information hypothesis. It will also relate the geometrical distance and the graph distance by a factor s. As a consequence, the guarantee that information does not propagate too fast with respect to the geometrical distance will be inherited from its counterpart in graph distance. In particular, it will forbid the sudden collapse phenomenon of Fig. 4. All we need to do, then, is to impose that CCD take bounded-star graphs into bounded-star graphs. This can be decided from its local rule.

Definition 8 (Bounded-star preserving) *A CGD F is* bounded-star preserving *if and only if for all bounded-star graph G, $F(G)$ is also bounded-star. A local rule f is bounded-star preserving if and only if it induces bounded-star preserving a global dynamics F.*

Proposition 2 (Decidability of bounded-star preservation) *Given a local rule f and a bound s, it is decidable whether f is bounded-star preserving with bound s.*

Conclusion

Summary. We have obtained that the following definition captures Causal Complexes Dynamics, i.e. evolutions of discrete surfaces such that information does not propagate too fast and that act everywhere the same:

Definition 9 (Causal Complexes Dynamics) *A function F from $\mathcal{G}_\pi$ to $\mathcal{G}_\pi$ is a* Causal Complexes Dynamics, *or CCD, if and only if there exists r a radius and f a rotation-commuting, bounded-star preserving local rule from $\mathcal{D}^r_\pi$ to $\mathcal{G}_\pi$ such that for every graph G in $\mathcal{G}_{\Sigma,r}$, $F(G) = \bigcup_{v\in G} f(G^r_v)$.*

We have also obtained that given a candidate local rule f, it is decidable whether it as the required properties. Since CCD are a specialization of CGD, several results follow as corollary from [1, 2, 3]. For instance, it follows CCD of radius 1 are universal, that CCD are composable, that CCD can be characterized as the set of continuous functions from discrete surfaces to discrete surfaces with respect

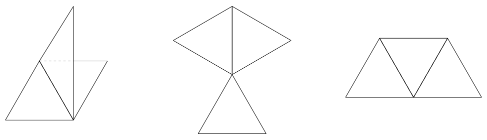


Figure 5: Complexes, pseudomanifolds, combinatorial manifolds.

to the Gromov-Hausdorff-Cantor metric upon isomorphism classes. These results deserve to be made more explicit, but they already are indicators of the generality of the model.

Further work. We went constantly back and forth from graph to simplicial complexes, but we have not formalized this relationship. First: Can every such graph be mapped into a $2D$ oriented simplicial complex? On the one hand, it is intuitive that each vertex represents an oriented triangle, and each edge specifies a unique oriented gluing. On the other hand, we are able to represent a sphere, a cylinder, or a torus with just two vertices, whereas these need many triangles in the simplicial complex formalism. Hence the correspondence is to be found with more economical formalisms such as Δ-complexes [8]. Second: Can any $2D$ oriented simplicial complex be represented by such a graph? We are willingly limiting ourselves to those complexes that arise as discretizations of $2D$ manifolds, i.e. combinatorial manifolds with borders [9]. In the $2D$ case these are just the complex obtained by gluing triangles pairwise only and only along their sides (See Fig. 5). In n-dimensions combinatorial manifolds are harder to characterize, however: the star of every point must be an n-ball. Our bounded-star restriction will then play a crucial role.

Acknowledgements

The authors would like to thank Christian Mercat and Zizhu Wang for helping them enter the world of simplicial complexes.

References

[1] P. Arrighi & G. Dowek (2012): *Causal graph dynamics*. In: *Proceedings of ICALP 2012, Warwick, July 2012, LNCS*, 7392, pp. 54–66.

[2] P. Arrighi & G. Dowek (2012): *Causal graph dynamics (long version)*. *Information & Computation journal, to appear. Pre-print arXiv:1202.1098.*

[3] P. Arrighi & S. Martiel (2012): *Generalized Cayley graphs and cellular automata over them*. In: *Proceedings of GCM 2012, Bremen, September 2012.*, pp. 129–143.

[4] P. Arrighi, S. Martiel & V. Nesme (2013): *Generalized Cayley graphs and cellular automata over them. submitted (long version). Pre-print arXiv:1212.0027.*

[5] P. Arrighi, N. Schabanel & G. Theyssier (2012): *Intrinsic Simulations between Stochastic Cellular Automata*. In: *Proceedings of AUTOMATA & JAC 2012, La Marana, Corsica, September 2012. EPTCS*, 90, pp. 208–224.

[6] Alberto Dennunzio, Enrico Formenti & Julien Provillard (2011): *Non-Uniform Cellular Automata: classes, dynamics, and decidability*. *CoRR* abs/1107.5228. Available at `http://arxiv.org/abs/1107.5228`.

[7] R. Gandy (1980): *Church's thesis and principles for mechanisms*. In: *The Kleene Symposium*, North-Holland Publishing Company, Amsterdam.

[8] Allen Hatcher (2001): *Algebraic Topology*. Http://www.math.cornell.edu/ hatcher/AT/ATpage.html.

[9] WB Raymond Lickorish (1999): *Simplicial moves on complexes and manifolds*. Geometry and Topology Monographs 2(299-320), p. 314.

[10] Luca Manzoni (2012): *Asynchronous cellular automata and dynamical properties*. Natural Computing 11(2), pp. 269–276.

[11] R. Sorkin (1975): *Time-evolution problem in Regge calculus*. Phys. Rev. D. 12(2), pp. 385–396.

Quiz games: a new approach to information hiding based algorithms in scientific computing

Joos Heintz
University of Buenos Aires, CONICET, Argentina and University of Cantabria, Spain

Bernd Bank
Humboldt University, Berlin

Luis Miguel Pardo
University of Cantabria, Spain

Andrés Rojas Paredes
University of Buenos Aires

A major progress in effective algebraic geometry was realised by the introduction of elimination algorithms which solve multivariate polynomial equation systems and are based on the idea of representing polynomials by division–free arithmetic circuits. These algorithms became implemented by the software package "Kronecker" developed by G. Lecerf (Paris–Versailles). This led to the question whether the underlying Kronecker algorithm is already optimal for this kind of elimination problems. The main outcome of this algorithm is the proof that elimination polynomials are "smart to evaluate". This means that they admit circuit representations which are of polynomial size in their degree, whereas their representation by coefficients may grow exponentially in this quantity.

On the other hand one may ask whether Lagrange interpolation of circuit encoded multivariate polynomials admits short circuits for the representation of the output. In the past the authors of this contribution exhibited infinite families of computational examples which show that the Kronecker algorithm is optimal but leads in worst case to exponential complexity and that the Lagrange interpolation problem considered above is unfeasible too in worst case.

The aim of this talk is to present a uniform approach to both complexity results extending their meaning to arbitrary representations of polynomials, beyond the circuit representation. The outcome concerns algorithms which can be implemented applying commonly accepted rules of object oriented software engineering for information hiding based programming with abstract data types. We present two versions of a two party protocol, called "quiz game", which embodies the informal concept of information hiding. The first version corresponds to an exact and the second one to an approximative computation model which is well adapted to numeric calculus.

Submitted to:
DCM 2013

A Calculus of Located Entities

Adriana Compagnoni
Department of Computer Science
Stevens Institute of Technology
New Jersey, USA
Adriana.Compagnoni@stevens.edu

Paola Giannini
Computer Science Institute
DISIT, Univ. Piemonte Orientale
Alessandria, Italy
giannini@di.unipmn.it

Catherine Kim
Department of Computer Science
Stevens Institute of Technology
New Jersey, USA
ckim@stevens.edu

Matthew Milideo
Department of Computer Science
Stevens Institute of Technology
New Jersey, USA
mmiledeo@stevens.edu

Vishakha Sharma
Department of Computer Science
Stevens Institute of Technology
New Jersey, USA
vsharma1@stevens.edu

In this paper we define BioScape^L, a stochastic Pi-calculus in 3D-space with abstract locations. It builds on BioScape by associating with each entity a position in space, which is the barycentre of its associated shape. The position of an entity instance is programmable. This is similar to the origin of the affine transformation in Cardelli-Gardner's 3π. The motivation for such an extension comes from the need to describe the assembly of configurations such as polymers, oligomers, and complexes in space, while keeping a high level description where diffusion and confinement remain part of the semantics of the calculus. In addition to this programmable deterministic translation, we can specify random translation and scaling. Random translation is not present in 3π, and scaling is more complex.

1 Introduction

In this paper we define BioScape^L, a stochastic π-calculus in 3D-space with abstract locations. It builds on BioScape [4] by associating with each entity a geometric transformation of its shape w.r.t. a global reference system. In particular, we specify translation of the barycentre of the entity, and scaling of its shape. Translations may have deterministic and random components, which are resolved at simulation time. Geometric transformations may be programmed when defining the entity behavior. This is similar to the geometric modeling of Cardelli-Gardner's 3π calculus, where processes are equipped with affine transformations. The motivation for such an extension comes from the need to describe systems whose behavior depends on geometric information and dynamic spatial arrangements of their entities, such as in assembly of polymers, oligomers, and complexes. Furthermore, diffusion and confinement remain part of the semantics of the calculus, unlike in 3π, where they are a burden to the programmer.

We introduce the extension of BioScape in two steps. Through an example we show how deterministic translation can be specified, and we present, in Section 2 the formalization of BioScape^L. In Section 3, we introduce the changes needed to the syntax and operational semantics of Section 2, in order to specify random translations and scaling.

A motivating example Microtubules are dynamic tubulin polymers; they are part of the cytoskeleton of eukaryotic cells, and they form roads on which organelles ride on their way to the cell nucleus. Although microtubules are hollow and formed with dimers of α and β tubulin, we simplify their structure in our example. Microtubules are assembled starting from parts, `MTPart`, where a part is an α-β tubulin dimer. Microtubules have a start piece `MTRight` and an end piece `MTLeft`. Between the right and left end pieces there are any number of `MTMiddle` pieces. While the start piece is fixed, microtubules grow and shrink from the end piece. In order to grow, a new `MTPart` becomes the new `MTLeft`, and the old

```
val Cytosol:space = cuboid(50.0, 50.0, 30.0) @ <1.0, 2.0, 24.0>
val step = 0.0, r = 1.0
new MTConstruction@0.116,r:chan(chan ),  MTSend@0.1,r:chan(chan, fl*fl*fl),
       MTSendBack@0.1,r:chan{fl*fl*fl}

let MTPart()@Cytosol, step, sphere(1.0) = (new y@0.27,r:chan
do ?MTConstruction(x); MTLeft(x)_this
or !MTConstruction(y); MTRight(y))_this)

and MTRight(rht:chan)@Cytosol, step, sphere(1.0) =
do delay@1.0; MTRight(rht)_this
or ?rht; MTPart()_this

and MTLeft(lft:chan)@Cytosol, step, sphere(1.0) =
do delay@1.0; MTLeft(lft)_this
or delay@0.27; (new z@0.27,r:chan
     !MTConstruction(z); MTMiddle(lft,z)_this )
or !lft; MTPart()_this
or ?lft;?MTSend(mdl,v);!MTSendBack(this);MTLeft(mdl)_v

and MTMiddle(rht1:chan, lft1:chan)@Cytosol, step, sphere(1.0) =
do delay@1.0; MTMiddle(rht1, lft1)
or !lft1;!MTSend(rht1,this);?MTSendBack(w);MTPart()_w

run (MTPart()_p1 | MTPart()_p2 |  ......  | MTPart()_pN )
```

Figure 1: Microtubules polymerization

MTLeft becomes an MTMiddle. Similarly the end piece can disassemble making the last MTMiddle the new MTLeft, and making the old MTLeft a free MTPart. The construction is done using private channels, similar to the process modeling of actin polymerization of [1], so that only adjacent pieces share channels.

The BioScape^L program for the microtubules example is shown in Fig. 1. We assume an initial concentration of N MTPart's placed in the Cytosol; implemented with a parallel composition of N copies of MTPart with barycentres p1, $\cdots$, pN. See the run command in the last line of the program.

The behavior of the entity MTPart is defined on the right-hand side of the equal sign. For each part a new channel is defined with new y@0.27,r, where y is the name of the channel (clearly any part will have a new name). The stochastic reaction rate of the channel, 0.27, is used in the simulation algorithm to determines the probability and the reaction time of this synchronization. The radius, r, indicates the channel scope, that is how close the entities must be to interact on this channel. MTPart can either do an input on channel MTConstruction, ?MTConstruction(x), or an output on the same channel, !MTConstruction(y).

Consider MTPart()_p1 | MTPart()_p2, representing a MTPart at location p1, and another at p2. If the distance between p1 and p2 is at most r, there can be a synchronization on the MTConstruction channel. The entity MTPart()_p1 sends on channel MTConstruction the channel name y and it becomes MTRight(y)_p1, and MTPart()_p2 receives y on channel MTConstruction, binds y to x and becomes MTLeft(y)_p2 sharing the private channel y with MTRight(y)_p1. The decoration this denotes the barycentre of the MTPart from which MTRight or MTLeft evolve. The metavariable this is a way to get a hold of the runtime position of the generating entity similar to the origin, ✠, of 3π. In general, an entity could be decorated with a position resulting from the evaluation of an expression, such as sum of points, or product by a scalar returning a point. Therefore, we are specifying a translation (which is an affine transformation) of the reference system of the entity (on the right-hand-side of the definition) from the origin of the reference system of the defined entity (on the left-hand-side of the definition).

The entity `MTRight` has the simplest behavior: it can just either stay alive with a delay prefix or do an input action with the (only) `MTLeft` with which it shares the channel `rht` and evolve into a `MTPart` placed in its original position. This corresponds to disassembling a dimer from the microtubule. Notice that, in this case there is no information sent on the channel `rht`.

The entity `MTLeft` has four alternative behaviours. It can

- stay alive with a delay prefix (first line of the definition); or
- interact with a `MTPart` by synchronizing on channel `MTConstruction` (after a delay), and evolve into a `MTMiddle` piece with which it shares the private channel `z`. In other words, the process `MTLeft(y)_p2 | MTPart()_p3` evolves into `MTMiddle(y,z)_p2 | MTLeft(z)_p3`; or
- interact with a `MTRight` by synchronizing on their private channel and disassemble; or
- interact with a `MTMiddle` piece on their private channel, and disassemble.

The last case is the most complex, and we explain it through an example. Assume we have

```
MTMiddle(y,z)_p2 | MTLeft(z)_p3
```

where `z` is private to these two entities. The synchronization on `z` makes `MTMiddle(y,z)_p2` evolve into `!MTSend(y,p2);?MTSendBack(w);MTPart()_w`. This entity now can make an output on channel `MTSend` sending the channel name `y` and its position `p2`. On the other side, with the same synchronization `MTLeft(z)_p3` evolves into `?MTSend(mdl,v);!MTSendBack(p3);MTLeft(mdl)_v`, (note that `mdl` and `v` are input parameter, and `this` is replaced by `p3`, the position of this `MTLeft`). The two entities may interact synchronizing on channel `MTSend`, so that the first evolves into `?MTSendBack(w);MTPart()_w`, and the second into `!MTSendBack(p3);MTLeft(y)_p2` (the input parameter `mdl` has been replaced with the channel name `y`, and `v` with the position `p2`). If the two entities interact again synchronizing on `MTSendBack`, then `MTMiddle(y,z)_p2` evolves into `MTLeft(y)_p2`, becoming the left end of the microtubule , and `MTLeft(z)_p3` evolves into `MTPart()_p3`, becoming a free part. The example shows how positions can be sent on channels, and used to compute the positioning of entities.

2 BioScape^L

The abstract syntax of BioScape^L, which is an extension of BioScape [3], is defined in Fig. 2. This syntax is a minor variation of the concrete syntax of the example in Fig. 1.

We assume a set of *channel names*, denoted by a, b, a set of *variables*, denoted by x, y, and *constants for real numbers*, denoted by $\mathtt{c}$, with subscripts or superscripts, if needed. We use the *floating-point representation* of real numbers. *Points*, denoted by the metavariable p, are triples $(\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3)$ of real numbers.

The empty process is 0. (In the concrete syntax there are integers, booleans, strings, etc.) By $X(\delta)_{\delta'}$ we denote an instance of the entity defined by X. The actual parameter of the instance will be the result of the evaluation of the expression δ. The subscript δ', is an expression whose evaluation will give the position of the barycentre of the shape of the entity X. Triples of floating-point numbers are used to represent points in 3D space. The metavariable `this`, in the syntax clause for expressions, denotes the position of the barycentre of the shape of the entity in which definition the instance of X appears. In the example of Fig. 1, the instances of `MTRight` (and `MTLeft`) on the right-hand-side of the definition of `MTPart()` have the subscript `this` meaning that when `MTPart()` reduces to `MTRight` (or `MTLeft`)

$P,Q ::= 0$	Empty Process
$\quad \mid\ X(\delta)_\delta$	Located Entity Instance
$\quad \mid\ P \mid Q$	Parallel Composition
$\quad \mid\ (\nu a@\mathtt{r},\mathtt{rad} : \mathtt{chan}\{T\}).P$	Restriction
$M ::= \pi.P\ [+\ M]$	Choice of Prefixed Process
$\pi ::= \mathtt{delay}@\mathtt{r}$	Delay
$\quad \mid\ !u(\delta)$	Output
$\quad \mid\ ?u(x)$	Input
$\quad \mid\ \mathtt{mov}$	Move
$u ::= a \mid b \mid \cdots \mid x \mid y \mid \cdots$	Identifiers
$\delta ::= u \mid \mathtt{c} \mid \mathtt{this} \mid \delta_1,\ldots,\delta_n \mid () \mid \delta.i \mid \mathtt{op}(\delta)$	Expressions
$v ::= a \mid b \mid \cdots \mid \mathtt{c} \mid () \mid v_1,\ldots,v_n$	Expression Values
$T ::= \mathtt{chan}\{T\} \mid \mathtt{fl} \mid T_1 * \cdots * T_n \mid \top$	Expression Types
$D ::= \emptyset \mid D, X(x : T) = M^{\xi,\omega,\sigma} \quad \mathtt{FV}(M) \subseteq \bar{x}$	Entity Definitions
$E ::= \emptyset \mid E, a@\mathtt{r},\mathtt{rad} : \mathtt{chan}\{T\}$	Channel Declarations
$\Gamma ::= \emptyset \mid \Gamma, X{:}T \mid \Gamma, u{:}T$	Type Environment

Figure 2: Syntax of BioScape^L

its barycentre will be in the position were the one of `MTPart()` was. The process $P \mid Q$ is the parallel composition of processes P and Q. By $(\nu a@\mathtt{r},\mathtt{rad} : \mathtt{chan}\{T\}).P$ we define the channel name a with two parameters `r` and $\mathtt{rad} \in \mathbb{R}_{\geq 0}$ within process P; the parameter `r` is the stochastic rate for communications through channel a, and `rad` is the communication radius. The radius is the maximum distance between processes in order to communicate through channel a, and the reaction rate determines how long it takes for two processes to react given that they are close enough to communicate. After the colon is the type of a, which is a channel on which values of type T are sent. In the example of Fig. 1 a pair whose first component is a channel name and the second a triple of floating-point numbers (a 3D point) is sent on channel `MTSend`. Note that, in the syntax of BioScape^L(Fig. 2), if no value is sent on the channel, that is the channel is only used for synchronization, the type of the channel is $\mathtt{chan}\{\top\}$, whereas in the concrete syntax it is just `chan`.

The *heterogeneous* choice is denoted by M, where $\pi.P\ [+\ M]$ means $\pi.P \ \mid\ \pi.P + M$. Choices may have reaction branches and movement branches. The reaction branches are probabilistic (stochastic), since reactions are subject to kinetic reaction rates, while the movement branches are non-deterministic, since the movement of instances of entities is always enabled. The prefix π denotes the action that the process $\pi.P$ can perform. The prefix `delay@r` is a spontaneous and unilateral reaction of a single process, where `r` is the stochastic rate. The prefix $!u$ denotes output, and the prefix $?u$ denotes input. Output on a channel specifies an expression whose value will be sent, and input on a channel specifies a variable, that will be bound to the value received. In the concrete syntax of Fig. 1, when no value is sent or received, we just use the channel name, e.g., input and output on channel `left` in the last two lines of the definition of `MTLeft`. The prefix `mov` moves processes in space according to their diffusion rate (ω) (see below). We use standard syntactic abbreviations such as π for $\pi.0$. *Free variables*, `FV`, and *free channel names*, `FN`, of processes and choices can be defined in the usual way. The input prefix $?u(x)$, and the restriction $\nu a@\_$ are binders, and define the scope of the variable x and the channel name a respectively.

Expressions may be channel names, variables, constants, the metavariable `this`, tuples of expressions, including the empty tuple $()$, selection of a component (of a tuple), and operators applied to an actual parameter which is in turn an expression. As already mentioned, the metavariable `this` denotes the barycentre of the entity on the left-hand-side of the definition on which the expression occurs. To define type checking, we assume the existence of a function `typeOf` such that $\texttt{typeOf}(\texttt{op}) = (T_1, T_2)$ says that the operator `op` takes a parameter of type T_1 and returns a value of type T_2.

Expression values, are either channel names or (floating-point) constants, or tuples of values, with $()$ being the tuple with 0 components. (We write constants for floating-point numbers with the standard dot notation.) The BioScape^L types, characterizing these values, are:

- channel types, $\texttt{chan}\{T\}$, specifying the type, T, of the values sent on them,
- the type of real numbers, `fl`
- the type of tuples, $T_1 * \cdots * T_n$, specifying the types, T_i , of the components, and
- $\top$, which is the type of the tuple with 0 components.

We denote by D a global list of definitions. The notation $X(x : T) = M^{\xi,\omega,\sigma}$ defines entity X with formal parameter x of type T to be the choice M with geometry ξ, ω, σ, specifying a movement space ξ, a step ω, and a shape σ. The choice M describes the behavior of X with a choice of prefixed processes. The selection of one of the choices depends not only on the available interactions with other processes, but also on the available space. The movement space ξ is a shape containing the entity shape that defines the limit of the possible movements of the entity, so that, X can move within ξ. The step $\omega \in \mathbb{R}_{\geq 0}$, is the distance that X can stir in a movement, and it corresponds to the diffusion rate of X; σ is the three-dimensional shape (sphere, cube, etc.) of X, having a barycentre. The movement space for the empty process 0 is everywhere, the global space, and its movement step is 0. Each entity variable X can be defined at most once in D, and the *free variables of M*, must be a subset of the variables $\bar{x}$. We also write $X(x) = (\pi.\pi'.P)^{\xi,\omega,\sigma}$ as short for $X(x) = (\pi.Y(x))^{\xi,\omega,\sigma}$ and $Y(x) = (\pi'.P)^{\xi,\omega,\sigma}$.

We use E to range over environments of channel name declarations. A channel name a appears at most once in E, so we consider E up to permutations of channel declarations. The *domain of E* is the set of channel names declared in E.

In Fig. 3 we define the rules for the judgements:

- $\Gamma \vdash \delta : T$, meaning, *in the type environment* Γ *the expressions* δ *has type* T,
- $\Gamma \vdash P \diamond$ ($\Gamma \vdash M \diamond$), meaning, *in the type environment* Γ *the process P (choice M) is well formed*, and
- $\Gamma \vdash D \diamond$, meaning, *in the type environment* Γ *the list of definitions D is well formed.*

The rules for expressions are standard; notice that the type of `this` in rule (Ty.this) is a triple of floating-point numbers representing 3D coordinates. An entity instance $X(\delta)_{\delta'}$ is well formed (rule (Ty.inst)), if the actual parameter δ has the type of X in the context, and if the δ' has the type of a 3D point. In rules (Ty.out) and (Ty.in) the channel identifier u must have a channel type.

Define the *type environment corresponding to channel declarations or entity definitions* as follows

- $\texttt{env}(\emptyset) = \emptyset$

$$\text{(Ty.id)}\ \frac{u{:}T \in \Gamma}{\Gamma \vdash u : T} \qquad \text{(Ty.const)}\ \frac{}{\Gamma \vdash \mathtt{c} : \mathtt{fl}} \qquad \text{(Ty.this)}\ \frac{}{\Gamma \vdash \mathtt{this} : \mathtt{fl} * \mathtt{fl} * \mathtt{fl}}$$

$$\text{(Ty.tuple)}\ \frac{\Gamma \vdash \delta_i : T_i \quad (1 \leq i \leq n)}{\Gamma \vdash \delta_1, \ldots, \delta_n : T_1 * \cdots * T_n} \qquad \text{(Ty.empty)}\ \frac{}{\Gamma \vdash () : \top}$$

$$\text{(Ty.sel)}\ \frac{\Gamma \vdash \delta : T_1 * \cdots * T_n \quad (1 \leq i \leq n)}{\Gamma \vdash \delta.i : T_i} \qquad \text{(Ty.op)}\ \frac{\mathtt{typeOf}(\mathtt{op}) = (T_1, T_2) \quad \Gamma \vdash \delta : T_1}{\Gamma \vdash \mathtt{op}(\delta) : T_2}$$

$$\text{(Ty.nil)}\ \frac{}{\Gamma \vdash 0 \diamond} \qquad \text{(Ty.inst)}\ \frac{X{:}T \in \Gamma \quad \Gamma \vdash \delta : T \quad \Gamma \vdash \delta' : \mathtt{fl} * \mathtt{fl} * \mathtt{fl}}{\Gamma \vdash X(\delta)_{\delta'} \diamond} \qquad \text{(Ty.par)}\ \frac{\Gamma \vdash P \diamond \quad \Gamma \vdash Q \diamond}{\Gamma \vdash P \mid Q \diamond}$$

$$\text{(Ty.restr)}\ \frac{\Gamma, a{:}\mathtt{chan}\{T\} \vdash P \diamond}{\Gamma \vdash (\nu a@\mathtt{r}, \mathtt{rad} : \mathtt{chan}\{T\}).P \diamond} \qquad \text{(Ty.out)}\ \frac{u{:}\mathtt{chan}\{T\} \in \Gamma \quad \Gamma \vdash \delta : T \quad \Gamma \vdash P \diamond}{\Gamma \vdash !u(\delta).P \diamond}$$

$$\text{(Ty.in)}\ \frac{u{:}\mathtt{chan}\{T\} \in \Gamma \quad \Gamma, x{:}T \vdash P \diamond}{\Gamma \vdash ?u(x).P \diamond} \qquad \text{(Ty.pref)}\ \frac{\Gamma \vdash P \diamond}{\begin{array}{l}\Gamma \vdash \mathtt{delay}@\mathtt{r}.P \diamond \\ \Gamma \vdash \mathtt{mov}.P \diamond\end{array}}$$

$$\text{(Ty.choice)}\ \frac{\Gamma \vdash M \diamond \quad \Gamma \vdash M' \diamond}{\Gamma \vdash M + M' \diamond} \qquad \text{(Ty.defs)}\ \frac{\Gamma \vdash D \diamond \quad \Gamma, x{:}T \vdash M \diamond}{\Gamma \vdash D, X(x) = M^{\xi, \omega, \sigma} \diamond}$$

Figure 3: Well typed expressions, processes, and definitions

- $\mathtt{env}(E, a@\mathtt{r}, \mathtt{rad} : \mathtt{chan}\{T\}) = a{:}\mathtt{chan}\{T\}, \mathtt{env}(E)$
- $\mathtt{env}(D, X(x : T) = M^{\xi,\omega,\sigma}) = X{:}T, \mathtt{env}(D)$

A BioScape^L *program* is a well formed sequence of channel and entity declarations followed by an initial process, which is a parallel composition of entity instances, that is

$$E,\ D,\ \ X_1(v_1)_{p_1} \mid \cdots \mid X_n(v_n)_{p_n}$$

such that

$$\mathtt{env}(E), \mathtt{env}(D) \vdash D \diamond \qquad \text{and} \qquad \mathtt{env}(E), \mathtt{env}(D) \vdash\ X_1(v_1)_{p_1} \mid \cdots \mid X_n(v_n)_{p_n} \ \diamond .$$

The *big-step operational semantics* of the expression language is presented in Fig. 4: $\delta \Downarrow v$ means that the evaluation of δ produces the value v. The rules are standard, just notice that selection of the i-th component of a tuple is successful only when the value of the expression to which it is applied has at least i components.

We can prove that: the evaluation of a well typed expression not containing free variables or the metavariable `this` produces a value (of the type of the expression).

To give the semantics of BioScape^L we first define the run-time configurations, their structural equivalence, and then we define the reduction relation, $\rightarrow$, between them. We call $\{X(v)\}_p$ a *located entity*. If σ is the shape of X, and σ' the shape of Y, define

$$\text{(Exp.ch)}\ \frac{}{a \Downarrow a} \qquad \text{(Exp.const)}\ \frac{}{\mathtt{c} \Downarrow \mathtt{c}} \qquad \text{(Exp.tuple)}\ \frac{\delta_1 \Downarrow v_1 \cdots \delta_n \Downarrow v_n}{\delta_1,\dots,\delta_n \Downarrow v_1,\dots,v_n} \qquad \text{(Exp.())}\ \frac{}{() \Downarrow ()}$$

$$\text{(Exp.sel)}\ \frac{\delta \Downarrow v_1,\dots,v_n \quad 1 \le i \le n}{\delta.i \Downarrow v_i} \qquad \text{(Exp.op)}\ \frac{\delta \Downarrow v \quad \mathtt{op}(v) = v'}{\mathtt{op}(\delta) \Downarrow v'}$$

Figure 4: Operational semantics of expressions

$$\text{(S.Loc)}\ \frac{P \equiv Q}{\{P\}_p \equiv \{Q\}_p} \qquad \text{(S.Inst.Ent)}\ \frac{\delta \Downarrow v \quad (\delta'[p/\mathtt{this}]) \Downarrow p'}{\{X(\delta)_{\delta'}\}_p \equiv \{X(v)\}_{p'}}$$

$$\text{(S.Loc.Par)}\ \frac{}{\{P\}_p \mid \{Q\}_p \equiv \{P \mid Q\}_p}$$

$$\text{(S.Loc.Nu)}\ \frac{}{(\nu a@\mathtt{r},\mathtt{rad}{:}\mathtt{chan}\{T\}).\{P\}_p \equiv \{(\nu a@\mathtt{r},\mathtt{rad}{:}\mathtt{chan}\{T\}).P\}_p}$$

$$\text{(S.Nu.Com)}\ \frac{}{\nu_1.\nu_2.A \equiv \nu_2.\nu_1.A}$$

$$\text{(S.Nu.Par)}\ \frac{a \notin \mathtt{fn}(B)}{((\nu a@\mathtt{r},\mathtt{rad}{:}\mathtt{chan}\{T\}).A) \mid B \equiv (\nu a@\mathtt{r},\mathtt{rad}{:}\mathtt{chan}\{T\}).(A \mid B)}$$

Figure 5: Structural Equivalence

- $Ps(p,X) = \{p+q \mid q \in \sigma\}$ to be the *set of point of X positioned at p*, and
- $\mathtt{dis}(\{X(v)\}_p, \{Y(v')\}_{p'})$ for *the distance between two located entities*, as the minimum of the set $\{d(p_1,p_2) \mid p_1 \in Ps(p,\sigma) \wedge p_2 \in Ps(p',\sigma')\}$, where $d(p_1,p_2)$ is the euclidean distance between the points p_1 and p_2.

Spatial configurations, denoted by A, B, … are defined as:

$$A,B ::= \{P\}_p \ \mid\ A \mid B \ \mid\ (\nu a@\mathtt{r},\mathtt{rad}).A \ \mid\ \{X(v)\}_p$$

where P is closed. The spatial configuration $\{P\}_p$ indicates an entity that has its barycentre at p and whose behaviour is described by the process P. Instead $\{X(v)\}_p$ denotes the entity whose behaviour is described by the definition of X and has its barycentre at p. This is different from $\{X(v)_\delta\}_p$ that represents an unspecified entity positioned at p that evolved into the entity X. The position of X is given by the evaluation of the expression δ in which the metavariable `this` is substituted by p.

Structural equivalence on configurations is defined in Fig. 5, where we omit the rules for associativity and commutativity of $|$ and $+$ and reflexivity, symmetry and transitivity. Parallel composition has neutral element $\{0\}_p$ for any p. Rule (S.Loc) uses the standard structural equivalence of Pi-calculus processes. Rule (S.Inst.Ent) places entity $X(\delta)_{\delta'}$ into space by instantiating the entity with the actual parameter resulting from the evaluation of the expression δ, and positioning its barycentre at the point resulting from the evaluation of the subscript expression δ' after substituting the point p (barycentre of the entity from which X evolved) by `this`. The well-formedness of processes ensures that the evaluation of δ', where `this` is substituted by p, produces a 3D point. In rule (S.LOC.PAR) the point p is distributed on the two processes saying that both processes will be located at position p. The rest of the rules deal with (channel name) restriction, and allow us to bring all the restriction outside the process, renaming if needed. The notation ν_i ($i = 1,2$) is an abbreviation for $\nu a_i@\mathtt{r}_i, \mathtt{rad}_i : \mathtt{chan}\{T_i\}$.

We say that a spatial configuration A *is canonical* if it is of the form:

$$\nu_1.\dots.\nu_m.\{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n} \tag{1}$$

The structural equivalence of Fig. 5 allows us to find for any B, a canonical form A such that $A \equiv B$. Note that in a canonical configuration there are no occurrences of $\{X(v)_\delta\}_p$, that is all the entities are located.

A canonical configuration is OK, if all its entities are contained in their respective movement space, and there are no overlapping entities. In the following, we define the OK predicate.

Definition 2.1. *Let A be the canonical configuration* $\nu_1.\dots.\nu_m.\{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n}$. A is OK *if:*

- *for all i,* $1 \leq i \leq n$*, we have that* $Ps(p_i, X_i) \subseteq \xi_i$*, and*
- *for all i, j,* $1 \leq i \neq j \leq n$*, we have that* $Ps(p_i, X_i) \cap Ps(p_j, X_j) = \emptyset$*.*

The operational semantics of BioScape^L is given in Fig. 6, by the reduction relation $\rightarrow$ on *run-time configurations* of the form, $E \vdash \{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n}$, where all the free channel names of $\{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n}$ are in the domain of E. The reduction $\rightarrow$ is defined by the rule (PAR). This rule uses the *auxiliary reductions* $\overset{\mathtt{r}}{\hookrightarrow}$ and $\overset{\mathtt{mv}}{\hookrightarrow}$. Since the spacial configuration B to which A reduces ($E \vdash A \overset{\mathtt{l}}{\hookrightarrow} B$) may not be a parallel composition of located entities, in order to produce a correct run-time configuration, we consider a canonical configuration, $\nu_1.\dots.\nu_m.D$, structurally equivalent to B, in which D is a parallel of located entities. In the configuration resulting from the reduction, all the channel definitions corresponding to the restrictions $\nu_1.\dots.\nu_m$ are moved into the channel environment. In so doing, we assume renaming of the names in the restriction to avoid clashes with channel names already in the domain of E. In this rule we also check that the configuration produced: $D \mid C$ is OK, that is, it is space consistent. The system is *stuck* if there is no auxiliary rule that can be applied, and after the reduction the configuration obtained is OK. Therefore, the *evolution of systems in BioScape^L produces configurations in which entities do not overlap, and are confined in their movement space.*

We denote the reflexive and transitive closure of $\rightarrow$ with $\rightarrow^*$. The rules of the auxiliary reductions involve entities, $X(v)$, and entities evolve according to one of the choices in their definitions in D. In the rules (DELAY), (COM), and (MOVE) there is no check of whether the entities of the resulting process overlap or whether they are contained in their movement space. These checks are done, as previously said, in the reduction rule (PAR) where in the premise `l` is either `r` or `mv`.

In the two stochastic rules, (DELAY), and (COM), `r` is the rate of the synchronization that determines probability and duration of the reduction. Rule (DELAY) makes the entity X evolve into the process P with a stochastic rate `r`. In rule (COM) the entity $X(v_x)$ sends on channel a the value v_a to the entity $Y(v_y)$, and evolves into process P located at p_x. The entity $Y(v_y)$ receives v_a and evolves into Q in which v_a substitutes the variable z located at p_y. This communication happens on the common channel a if the located entities $\{X(v_x)\}_{p_x}$ and $\{Y(v_y)\}_{p_y}$ are close enough. In particular, to interact on channel $a@\mathtt{r},\mathtt{rad}$, must be that $\mathtt{dis}(\{X(v)\}_p, \{Y(v')\}_{p'}) \leq \mathtt{rad}$. For instance $\mathtt{rad} = 0$ means that the two entities must be in contact to react.

The non-stochastic rule (MOVE) defines movement (Fig. 6). In this rule, $\mathtt{rand}(\omega)$ returns *a random point whose distance from* $\langle 0,0,0 \rangle$ *is* ω. This prefix `mov` says that the entity is subject to brownian motion. The located entity is moved randomly a distance ω from its original position.

$$\text{(PAR)}\ \frac{E \vdash A \overset{\mathtt{l}}{\hookrightarrow} B \qquad B \equiv \nu_1.\dots.\nu_m.D \text{ canonical} \qquad D \mid C \text{ OK}}{E \vdash A \mid C \to E, a_1@\mathtt{r}_1, \mathtt{rad}_1 : \mathtt{chan}\{T_1\}, \dots, a_m@\mathtt{r}_m, \mathtt{rad}_m : \mathtt{chan}\{T_m\} \vdash D \mid C}$$

$$\text{(DELAY)}\ \frac{X(x) = (delay@\mathtt{r}.P\ [+M])^{\xi,\omega,\sigma} \in D}{E \vdash \{X(v)\}_p \overset{\mathtt{r}}{\hookrightarrow} \{P[v/x]\}_p}$$

$$\text{(COM)}\ \frac{\begin{array}{l} X(x) = M_x^{\xi,\omega,\sigma} \in D \quad M_x[v_x/x] = (!a(\delta_a).P\ [+M]) \quad \delta_a \Downarrow v_a \\ Y(y) = M_y^{\xi',\omega',\sigma'} \in D \quad M_y[v_y/y] = (?a(z).Q\ [+N]) \\ \mathtt{dis}(\{X(v_x)\}_{p_x}, \{Y(v_y)\}_{p_y}) \leq \mathtt{rad} \end{array}}{E, a@\mathtt{r}, \mathtt{rad} : \mathtt{chan}\{T\} \vdash \{X(v_x)\}_{p_x} \mid \{Y(v_y)\}_{p_y} \overset{\mathtt{r}}{\hookrightarrow} \{P\}_{p_x} \mid \{Q[v_a/z]\}_{p_y}}$$

$$\text{(MOVE)}\ \frac{p' = p + \mathtt{rand}(\omega) \quad X(x) = (\mathtt{mov}.P\ [+M])^{\xi,\omega,\sigma} \in D}{E \vdash \{X(v)\}_p \overset{\mathtt{mv}}{\hookrightarrow} \{P[v/x]\}_{p'}}$$

Figure 6: Reduction Relation

Given a BioScape^L program $E,\ D,\ X_1(v_1)_{p_1} \mid \cdots \mid X_n(v_n)_{p_n}$, the *initial configuration* of this program is:

$$E \vdash \{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n}$$

If $\{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n}$ is OK, then we can prove that for all E' and A such that $E \vdash \{X_1(v_1)\}_{p_1} \mid \cdots \mid \{X_n(v_n)\}_{p_n} \to^* E' \vdash A$, we have that A is OK.

3 Random Translation and Scaling

Consider the case of a bacterium that secretes a hydronium ion (`HIon`). The language extension discussed so far will allow us to describe where to locate the `HIon`, but it will be at a specified location with respect to the position of the bacterium. Instead we would like to be able to say that it should be at a distance equal to the sum of the radius of the bacterium and the ion, but in a random direction. To this end, we decorate entity instances with expressions evaluating to pairs, whose first component is, as before, a translation point, and the second a number which specifies, as in the rule (MOVE) of Fig 6, a distance from which we generate a random point. In the fragment of code Fig. 7(a) the barycentre of the instances of `Bac` will be in the position of the `Bac` they evolve from, whereas the the barycentre of the instances of `HIon` will be in a random position that is at a distance equal to the sum of the radius of the bacterium and the ion, from the barycentre of the `Bac` it evolves from.

As far as the definition of the syntax of this extension, we have to change the typing rule for entity instances, see rule (TY.INST) of Fig. 3 specifying that the type of δ' must be $(\mathtt{fl} * \mathtt{fl} * \mathtt{fl}) * \mathtt{fl}$. Regarding the semantics we have to change the rule of structural equivalence (S.INST.ENT) that places entities into space, as follows:

$$\text{(S.INST.ENT.R)}\ \frac{\delta \Downarrow v \quad (\delta'[p/\mathtt{this}]) \Downarrow (p', \mathtt{c}) \quad p'' = p' + \mathtt{rand}(\mathtt{c})}{\{X(\delta)_{\delta'}\}_p \equiv \{X(v)\}_{p''}}$$

We now consider the shape of entities. As it is now, we have a specific shape and always the same dimension. In order to represent a change in scale a new entity with a smaller or bigger shape would have

```
Bac() =
do mov.Bac()_(this,0)
or delay@0.005.(Bac()_(this,0) | HIon()_(this,rB+rH))
or ...
```

(a)

```
Bac() = ... max-size ...
do mov.Bac()_(this.1,0,1.1)
or delay@0.005.(Bac()_(this.1,0,1) | HIon()_(this.1,rB+rH,1))
or delay@0.2.(Bac()_(this.1,rB,0.5) | Bac()_(this.1,rB,0.5))
or .....
```

(b)

Figure 7: (a) Random translation and (b) scaling

to be defined. Alternatively, we would like to be able to change the size of the entity using scaling directives. For instance, consider adding to the previous example of the bacterium the fact that bacteria grow and divide. So in Fig. 7(b) we add these behaviour, specifying that the bacterium may spontaneously divide into two bacteria of half the size of the original one, and moved apart in random directions a distance equal to the radius of the shape of the original bacterium, and that movement is associated with a growth of 10%. We can see that, now to get the barycentre of the enclosing entity, since located entities will be labelled with a point and a scaling factor we have to select the first component of `this`. For instance, in `mov.Bac()_(this.1,0,1.1)`, `this`.1 refers to the barycentre of the entity it evolved from. We could also access the scaling factor of this entity with `this`.2. In the definition of the entity `Bact` we fix a maximum growth, `max-size`.

As far as the syntax of the language is concerned we modify the typing rule for instance of entities, (Ty.inst) of Fig. 3, specifying that the type of δ' must be $(\mathtt{fl}*\mathtt{fl}*\mathtt{fl})*\mathtt{fl}*\mathtt{fl}$, and the type for `this` is $(\mathtt{fl}*\mathtt{fl}*\mathtt{fl})*\mathtt{fl}$. Moreover the definition of entities becomes: $X(x:T) = M^{\xi,\omega,\sigma,\mu}$ where μ record the *maximum scaling factor for the entity X*.

As already mentioned, the syntax of spatial configurations records in addition to the barycentre of the shape of the entity also its scaling factor s, that is

$$A,B ::= \{P\}_{(p,s)} \;\mid\; A \mid B \;\mid\; (\nu a@\mathtt{r},\mathtt{rad}).A \;\mid\; \{X(v)\}_{(p,s)}$$

In placing entities into space, rule (S.Inst.Ent.R), and in the rule for movement (Move) we have to scale the random quantity added to the translated point since this quantity refers to the initial (standard) dimension of entity X. In rule (S.Inst.Ent.R) the scaling factor of the located entity is obtained by multiplying the scaling factor of the entity from which X evolved by the one specified for X. The new rules replacing (S.Inst.Ent) and (Move) are shown in Fig. 8.

Scaling affects the dimension of the shape of entities, and therefore the space occupied by them. Consequently the definition of OK configuration, and the distance between entities will have to take into account this fact. In particular, let $X(x) = M^{\xi,\omega,\sigma,\mu}$ define $Sc(s,X) = \{s \times p \mid p \in \sigma\}$, where $\times$ is scalar product. The new definitions are as follows.

Definition 3.1. *(i)* $\mathtt{dis}(\{X(v)\}_{(p,s)}, \{Y(v')\}_{(p',s')})$ *is the minimum of* $\{d(p_1,p_2) \mid p_1 \in Ps(p,Sc(s,X)) \wedge p_2 \in Ps(p',Sc(s',Y))\}$.

(ii) Let A be the canonical configuration $\nu_1.\dots.\nu_m.\{X_1(v_1)\}_{(p_1,s_1)} \mid \cdots \mid \{X_n(v_n)\}_{(p_n,s_n)}$. A is OK *if:*

$$\text{(S.INST.ENT.S)}\ \frac{\delta \Downarrow v \quad (\delta'[(p,s)/\mathtt{this}]) \Downarrow (p', \mathtt{c}_r, \mathtt{c}_s)}{\{X(\delta)_{\delta'}\}_{(p,s)} \equiv \{X(v)\}_{(p'+\mathtt{rand}(\mathtt{c}_r \times s), s\times \mathtt{c}_s)}}$$

$$\text{(MOVE.S)}\ \frac{X(x) = (\mathtt{mov}.P\ [+\ M])^{\xi,\omega,\sigma} \in D}{\{X(v)\}_{(p,s)} \overset{\mathtt{mv}}{\hookrightarrow} \{P[v/x]\}_{(p+\mathtt{rand}(s\times\omega),s)}}$$

Figure 8: Modified rules for random translation and scaling

- *for all i,* $1 \leq i \leq n$, *we have that* $Ps(p_i, Sc(s_i, X_i)) \subseteq \xi_i$,
- *for all i,* $1 \leq i \leq n$, *we have that* $s_i \leq \mu_i$, *and*
- *for all i, j,* $1 \leq i \neq j \leq n$ *we have that* $Ps(p_i, Sc(s_i, X_i)) \cap Ps(p_j, Sc(s_j, X_j)) = \emptyset$.

We can still prove that, starting from an initial configuration we get OK configurations.

4 Related Work and Conclusions

BioScape is a modeling language based on process algebras such as SPiM [7], Kappa [5], Petri Nets [8], etc. However, these languages lack programmable spatial information. In [1] an extension of SPiM for displaying geometric information is introduced. However, this is a rather *ad hoc* extension motivated by the description of the biological process modelled that is actin polymerization. As already mentioned in the introduction, the calculus that is closer to BioScape^L is 3π([2]), a geometric process algebra in which processes are equipped with affine transformations. There are two main differences between BioScape^L and 3π. First, in BioScape^L we do not consider affine transformations, but just a uniform scaling in all directions maintaining the barycentre of the entity in its original position, and in addition to standard translation also a random translation, which is not an affine transformation. Second, and more important, is the fact that 3π is a low level language that gives absolute control of spatial attributes to the programmer, while in BioScape^L the programmer specifies species at a higher level, and it has been designed to program biological and biomaterial processes and their interactions. In [4], it is presented a fully abstract translation from BioScape to 3π, that could be extended to BioScape^L.

In collaboration with materials scientist Matthew Libera, from Stevens, we are working on the computationally assisted development of antibacterial surfaces. Traditionally biomaterials development consists of designing a surface and testing its properties experimentally. This trial-and-error approach is limited, because of the resources and time needed to sample a representative number of configurations in a combinatorially complex scenario. In many cases the design is also aided by computational models tailored to a specific application. In these cases, there have been successful attempts to identify biomaterials with optimal properties [6, 9, 10]. However, developing such dedicated software frameworks is time consuming, and small modifications in the understanding of the application can lead to significant and time consuming software changes.

Our proposal consists of designing antibacterial biomaterials from first principles. Using the antibacterial effect of individual components, we will computationally design optimally antibacterial surfaces, which simultaneously promote the growth of healthy tissue. Our model will stochastically assemble surface blocks whose connectivity will be determined by their antibacterial properties, as well as their ability to encourage tissue growth, in the same way a child assembles building blocks. These designed surfaces will then be tested in virtual experiments in the same platform. In order to test these surfaces we will use BioScape^L, where surfaces will be described by a collection of located entities generated by the surface design process.

The emerging surface patterns with maximal antibacterial effect will be used to design tiling patterns, which will motivate the design of new biomaterials that will then be tested in wet lab experiments.

References

[1] Luca Cardelli, Emmanuelle Caron, Philippa Gardner, Ozan Kahramanogullari & Andrew Phillips (2009): *A Process Model of Actin Polymerisation*. Electr. Notes Theor. Comput. Sci. 229(1), pp. 127–144.

[2] Luca Cardelli & Philippa Gardner (2012): *Processes in Space*. Theor. Comput. Sci. 431, pp. 40–55.

[3] Adriana Compagnoni, Mariangiola Dezani-Ciancaglini, Paola Giannini, Karin Sauer, Vishakha Sharma & Angelo Troina (2012): *Parallel BioScape: A Stochastic and Parallel Language for Mobile and Spatial Interactions*. In: Proceedings 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi, Newcastle, UK, 8th September 2012, Electronic Proceedings in Theoretical Computer Science 100, pp. 101–106.

[4] Adriana Compagnoni, Vishakha Sharma, Yifei Bao, Matthew Libera, Svetlana Sukhishvili, Philippe Bidinger, Livio Bioglio & Eduardo Bonelli (2013): *BioScape: A Modeling and Simulation Language for Bacteria-Materials Interactions*. Electronic Notes in Theoretical Computer Science 293(0), pp. 35 – 49. Proceedings of the Third International Workshop on Interactions Between Computer Science and Biology (CS2Bio'12).

[5] Jérôme Feret, Vincent Danos, Jean Krivine, Russ Harmer & Walter Fontana (2009): *Internal Coarse-graining of Molecular Systems*. Proceedings of the National Academy of Sciences 106(16), pp. 6453–6458.

[6] D. Lacroix, J. A. Planell & P. J. Prendergast (2009): *Computer-aided design and finite element modelling of biomaterial scaffolds for bone tisse engineering*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 367(1895), pp. 1993–2009.

[7] Andrew Phillips & Luca Cardelli (2007): *Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus*. In: CMSB'07, LNCS 4695, Springer, pp. 184–199.

[8] John W. Pinney, David R. Westhead & Glenn A. McConkey (2003): *Petri Net representations in Systems Biology*. Biochem. Soc. Trans. 31(6), pp. 1513–1515.

[9] Jack R. Smith, Agnieszka Seyda, Norbert Weber, Doyle Knight, Sascha Abramson & Joachim Kohn (2004): *Integration of Combinatorial Synthesis, Rapid Screening, and Computational Modeling in Biomaterials Development*. Macromolecular Rapid Communications 25(1), pp. 127–140.

[10] Kyriacos Zygourakis & Pauline A. Markenscoff (1996): *Computer-aided design of bioerodible devices with optimal release characteristics: a cellular automata approach*. Biomaterials 17(2), pp. 125 – 135.

Towards Formal Interaction-based Models of Grid Computing Infrastructures

Carlos Alberto Ramírez Restrepo
EISC – Universidad del Valle, Cali, Colombia

Jorge A. Pérez
CITI/DI – FCT - Universidade Nova de Lisboa, Portugal

Jesús Aranda
EISC – Universidad del Valle, Cali, Colombia

Juan Francisco Díaz Frias
EISC – Universidad del Valle, Cali, Colombia

Grid computing (GC) systems are increasingly used as large-scale machines built on top of a massive pool of resources (processing time, storage, software) which often span multiple distributed domains. Users may interact with the grid by concurrently adding new tasks; the grid is expected to assign resources to tasks in a fair and trustworthy way. These distinctive features of GC systems make their formal specification and verification a challenging issue. Although prior works have proposed formal approaches to the specification of GC systems, a precise account of the interaction model which underlies grid resource sharing has not been yet proposed. In this paper, we describe ongoing work aimed at filling this gap. Our formal approach relies on *(higher-order) process calculi*: these core programming languages for concurrency offer a simple, compositional framework in which the main grid components can be precisely expressed and potentially reasoned about.

1 Introduction

Context. Grid computing (GC) systems comprise a large pool of computational resources, which are made available by multiple institutions (*administrative domains*) to users wishing to execute tasks which would be hard (or even impossible) to perform in a single administrative domain. This is in sharp contrast with conventional distributed systems, in which resources are typically owned and controlled by a single institution. That is, while in distributed systems there is a clear correspondence between system users and valid resource users, in GC systems an analogous correspondence is less explicit, as resources may belong to multiple administrative domains. Moreover, a grid user may not correspond to an actual user in the administrative domains or resources. Yet another point of contrast concerns transparency and security requirements: while in conventional distributed systems users typically know a priori the resources that they need use to execute their tasks, in GC systems users may execute tasks without knowing (or being aware of) the internal structure in the system. GC systems differ also from emerging cloud computing infrastructures, which focus on offering economies of scale for exploiting virtually unlimited resources, based on the Software as a Service (SaaS) paradigm. In fact, differently from clouds, GC systems aim at the efficient execution of computationally intensive tasks, subject to constraints determined by limitations on resource availability/access. Other notable differences between clouds and grids concern failure management, resource ownership, and infrastructure transparency [8].

Our Approach. Here we are concerned with specification and verification techniques that support the design and construction of GC systems. As discussed above, a critical aspect here is that of appropriately assigning resources to a potentially huge number of user tasks running concurrently. In fact, given the scale and complexity of GC systems, this task appears a challenging issue from several perspectives. For this reason, we are interested in formal frameworks that may offer a sound basis for the analysis of the concurrent, interactive behavior that is intrinsic to GC systems. We observe a discrepancy between the practical adoption of grid infrastructures and the formal methods available to appropriately describe and support such infrastructures. In particular, little work has addressed the peculiarities of the interaction-based behavior of GC systems. In this paper, we describe ongoing work aimed at understanding how *process calculi* can shed light in this discrepancy. Based on a small set of operators—atomic interaction, sequencing, parallel composition, scoping, recursion—process calculi such as CCS [11] and the π-calculus [12] have been studied within the concurrency theory community as formal computational models for describing communicating systems. Their compositional character provides a suitable basis for developing proof techniques over formal specifications (e.g. behavioral equivalences and type systems) and for investigating new programming abstractions based on communication.

Our approach to GC systems is based on *higher-order process calculi*. Roughly speaking, these are core programming models for concurrency in which the communication of processes (i.e. full programs) is allowed. This is in contrast to the forms of communication available in calculi such as CCS and the π-calculus, in which only *first-order* values (such as booleans, integers, references, or communication channels) can be exchanged. From a foundational level, higher-order process calculi can be seen as concurrent variants of the λ-calculus. In fact, reduction for process exchange in higher-order process calculi is reminiscent of usual β-reduction in functional calculi. In our setting, higher-order communication naturally models the fact that user (sub)tasks —which typically describe complex sequences of computational behaviors— are exchanged among different grid components in order to achieve their execution. We rely on the higher-order π-calculus (HOπ) [15]: this is a well-known calculi which enhances the π-calculus with the ability of communicating processes. Hence, HOπ specifications can abstract forms of code mobility, therefore allowing for flexibility in descriptions of concurrent communications. Moreover, proof techniques based on behavioral equivalence are well-understood for HOπ (see, for instance, [16]).

The main contribution of this work is then a formal model of GC infrastructures, with a focus on the resource assignment facility that is central to them. Our model distills the main GC features informally discussed in the literature (see, e.g., [6]) and our own exchanges with GC experts. The model is roughly in two parts: a static component and a dynamic component. As its name suggests, the former defines the static properties of GC systems: using first-order logic, we identify and formalize a series of pre-conditions and basic invariants that should hold for GC components. Such a static specification, however, is quite limited to represent and analyze the concurrent (interleaved) execution sequences due to possibly many grid users running simultaneously. To remedy this, the dynamic component of our model, expressed as HOπ processes, describes the dynamics in GC systems by defining the interactions among its components. Building upon the elements of the static part of the model, the dynamic component explicitly represents the main components present in real infrastructures, such as users, tasks, administative domains, virtual organizations, resources and processes. Moreover, in order to describe explicitly certain details of GC systems, we also consider user and resource proxies.

While simple, our current model already provides a good basis for specifying more realistic GC systems and for reasoning about GC correctness properties, by appealing to well-known proof techniques associated to process calculi. Examples of such properties are authentication and authorization guarantees: they are intended to ensure that users only access and use the administrative domains and resources for which they can prove their identity/permissions. For example, a user only can access the administrative domains where she is a member; she may only use resources belonging to such domains. The use of higher-order process calculi with cryptographic elements [9] may be useful to reason about such properties. Also, we would like to reason about task termination and resource delivery in the grid setting. We see the model presented here is an initial step to such analyses, exploiting known results on process termination and reachability properties for calculi such as CCS [3] and also for variants of HOπ [10, 7].

Related Work. We believe that our ongoing work already improves on previous attempts for grid formalization. We briefly review such works and contrast them with out approach. The π-calculus has been used in [18, 20, 19] for modeling and analyzing the specific aspects of grid services composition and workflow. These approaches only model the GC components related to grid services such as resources and tasks—other components and other aspects of GC dynamics are not taken into account. In contrast, our model adopts a more comprehensive view of GC systems, including, for instance, key interaction patterns related to user intervention and the role of user and resource proxies in resource assignment and task execution. In [13], Abstract State Machines (ASM) are used to give a declarative characterization of GC systems, offering a formal description of the fundamental attributes that a GC system must support. The main GC elements are modeled as universes (sets) and the grid behavior is represented using rules over them. Processes are the only agents considered in this approach; they correspond to the tasks and user and resource mapping agents. These agents execute the rules over the defined universes. However, as evolution of GC components is described declaratively, concurrent interactions among GC components is not explicitly represented. Also, the model in [13] does not consider the key concept of virtual organizations and the role of user and resource proxies. Finally, in [1, 4], High Level and Coloured Petri Nets have been used to describe and analyze the grid architecture and grid workflow. They model a 3-layer grid architecture and the interaction between GC components in these layers, but this approach does not consider the role of the virtual organizations, administrative domains, and security requirements—all of these being central to resource assignment.

Organization. The rest of this paper is organized as follows. Section 2 briefly recalls the main features of GC systems. In Section 3 we present the syntax and semantics of the higher-order π-calculus. Section 4 gives a brief description of our GC formalization, and Section 5 illustrates it via a small example. Finally, future work is discussed in Section 6. A full description of our process model is available in [14].

2 Grid Computing: A Brief Overview

Grid computing broadly refers to the coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [6]. Distinctive features of GC systems include requirements of interoperability and support for heterogeneous and dynamic environments. Other typical requirements are decentralized control, security, access transparency, scalability, availability, and reconfigurability [6].

Sharing in GC systems not only refers to data and information but also to direct access to all kinds of resources (computing power, storage, software, data, network), which may be required for executing complex tasks. Each *administrative domain* (AD in the following) establishes what resources are shared, who is allowed to use them, and what are the usage policies. A *virtual organization* (VO in the following) is a set of administrative domains defined by such policies. The participants in a VO share resources in a controlled way in order to cooperate in executing a specific task. VOs vary in their purpose, scope, size, duration, structure, community, and sociology [6].

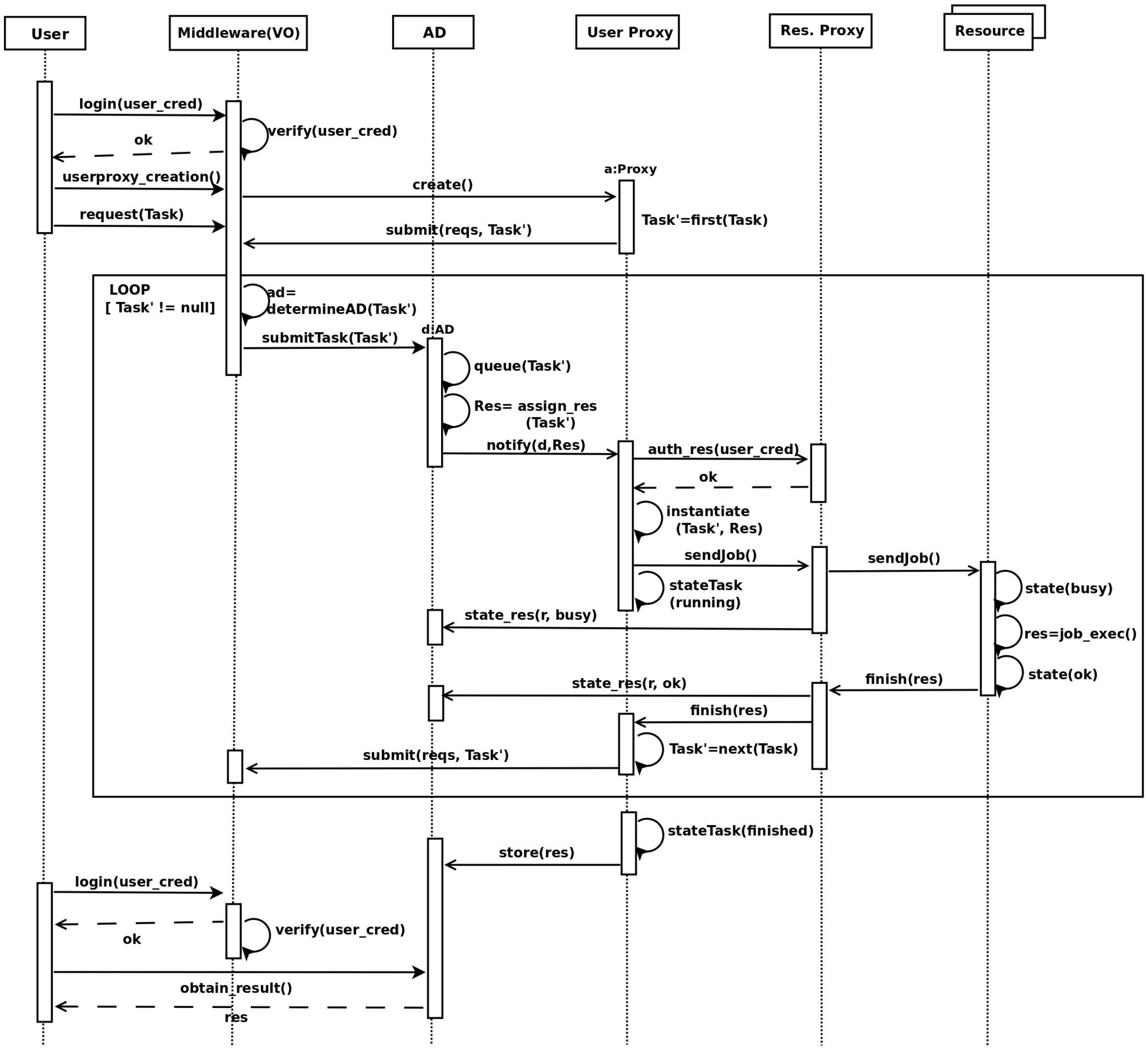
In GC systems, users can share or have direct access to resources in a transparent manner—they do not need to know (or be aware of) what resources they are using, where such resources are physically located or that they may have failed and have recovered. This transparent access to distributed resources is achieved through the use of the so-called *grid middleware*: a software layer that implements the *protocols* and *services* that enable the sharing of heterogeneous resources transparently and provides the most important functionality required for allowing task execution and establishing of VOs [17]. In this way, the middleware provides the mechanisms that allow users to use resources in a transparent way, while satisfying security requirements such as authentication, authorization, delegation and single sign-on. To this end, the middleware includes user and resource *proxies* [5]. While a user proxy is an entity that is given permission to act on behalf of a user for a fixed period of time, a resource proxy serves as interface between the middleware and a resource, thus simplifying (i) the authentication between user proxy and the resource and (ii) the mapping between grid users and the local users which are valid in the resource.

2.1 Grid Resource Assignment Protocol

As our interest is in an interaction-based approach to GC systems, next we present a protocol which describes the interaction sequence among the main grid components—namely, users, ADs, VOs, resources and proxies. The protocol is also described as a sequence diagram in Figure 1; it formalizes requirements and mechanisms which have been described in the literature only informally. Our process calculi model is then intended to give a precise account of this protocol.

1. A user sends its credentials to a grid node in order to authenticate. In the figure, this step is represented by the message $login(user\_cred)$ from User to VO.
2. If the authentication is successful then the user is granted to access the grid. Otherwise, the user must revise its credentials. In the figure, this step is represented by message *ok* from VO to User.
3. The authenticated user sends a proxy creation request, and a task with its requirements to the grid node. The task may be a complex object; in particular, it may be structured in terms of subtasks which follow some process logic. In the figure, these steps are represented by messages $userproxy_creation()$ (from User to VO), $create()$ (from VO to User Proxy), and $request(Task)$ (from User to VO).
4. The user proxy sends to the grid node the requirements of each subtask. In the figure, this step is represented by the message $submit(reqs, Task')$ from User Proxy to VO.
5. The grid node selects an AD in the VO with available resources to satisfy the requirements for the subtask. This subtask is assigned and sent to such an AD. In the Figure, this step is represented by messages $determineAd(Task')$ (inside VO), $submitTask(Task')$ (from VO to AD), and $queue(Task')$ (inside AD).
6. The AD assigns appropriate resources for this subtask according to some scheduling strategy. In the figure, this step is represented by the message $assignRes(Task')$ inside AD.
7. The user proxy authenticates into the resource proxies of assigned resources. If the authentication is successful then the subtask is executed. Otherwise, the subtask is sent back to the grid node. In the figure, these steps are represented by messages $auth\_res(user\_cred)$ (from User Proxy to Resource Proxy), *ok* (from Resource Proxy to User Proxy), $sendJob()$ (from Resource Proxy to Resource), and $job_exec()$ (inside Resource).
8. When the subtask has finished (message $finish(res)$, from Resource to its Resource Proxy), it is detected if there are more subtasks (condition $Task' \neq null$ in the loop). If yes then the result of the previous subtask is transmitted to the next subtask and the previous subprotocol is executed again (message $submit(reqs, Task')$). Otherwise, if the just executed subtask is the last one, then the result is stored and the protocol finishes (message $store(res)$ from User Proxy to AD).

Figure 1 Grid Interaction Protocol



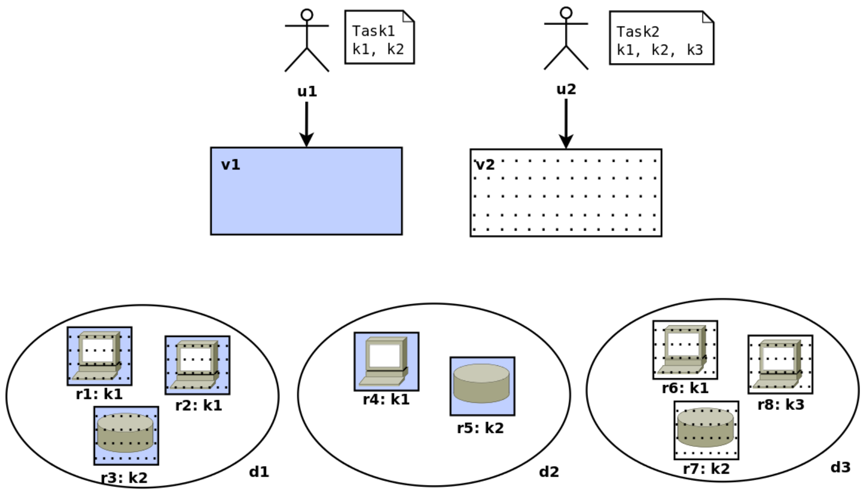
2.2 A Representative GC Scenario

We now describe a small but representative example of a GC system. Depicted in Figure 2, our scenario draws inspiration from the one discussed in [6]. The scenario contains three actual organizations (administrative domains) (denoted d_1, d_2, and d_3) and two VOs (denoted v_1 and v_2); in the figure, they are depicted as ovals and rectangles, respectively. VO v_1 (blue rectangle) groups participants in an aerospace design consortium and v_2 (lined rectangle) links participants for sharing spare computing cycles. AD d_1 is member of both v_1 and v_2. Also, ADs d_1 and d_2 participate in v_1 and AD d_3 participates in v_2.

In our scenario, we consider two users, denoted u_1 and u_2. While u_1 belongs to v_1, user u_2 belongs to u_2. Both u_1 and u_2 have a task to execute in the grid, denoted Task1 and Task2 in the figure, respectively. To perform, Task1 requires one resource of type (or descriptor) k_1 and one resource of type k_2. Similarly, Task2 requires three resources, distinguished by types k_1, k_2, and k_3.

Resources are located in appropriate ADs: AD d_1 owns three resources: r_1 (type k_1), r_2 (type k_1), and r_3 (type k_1); AD d_2 owns two resources: r_4 (type k_1) and r_5 (type k_2); and AD d_3 owns three resources:

Figure 2 A Representative Grid Scenario: Two users (u_1, u_2), two VOs (v_1,v_2), three ADs (d_1, d_2, d_3)



r_6 (type k_1), r_7 (type k_2), and r_8 (type k_8). While resources of d_1 are shared by v_1 and v_2, resources of d_2 are available only to v_1, and resources of d_3 are available only to v_2.

3 The Process Model: Syntax and Semantics

In this section, we briefly present the syntax and semantics of the higher-order π calculus [15], written HOπ in the following. In HOπ, both names (channels) and processes may be passed around by synchronization on names; communication can be thus loosely assimilated to β-reduction in the λ-calculus. We assume a set of names/channels ranged over $x,y,z,\dots$ and a set of process variables ranged over $X,Y,Z,\dots$. We write $\widetilde{o}$ to denote a finite tuple of elements $o_1,\dots,o_k$.

Definition 3.1. *The language of HOπ processes is given by the following syntax:*

$$\begin{array}{rcl}\alpha & ::= & x(\widetilde{U}) \;\big|\; \overline{x}\langle\widetilde{K}\rangle \\ P & ::= & \sum_{i\in I}\alpha_i.P_i \;\big|\; P_1 \mid P_2 \;\big|\; (\nu\, x)P \;\big|\; \texttt{if } [x=y] \texttt{ then } P_1 \texttt{ else } P_2 \;\big|\; \mathsf{D}\langle\widetilde{K}\rangle \;\big|\; X\langle\widetilde{K}\rangle\end{array}$$

We have two *prefixes*, ranged over $\alpha,\alpha',\dots$. An input prefix $x(\widetilde{U})$ (resp. output prefix $\overline{x}\langle\widetilde{K}\rangle$) denotes an atomic input action (resp. output action) on a name x. Above, $\widetilde{K}$ and $\widetilde{U}$ denote tuples of names and processes, and of names and variables, respectively. Process $\sum_{i\in I}\alpha_i.P_i$ represents the *non-deterministic choice* among prefixed processes $\alpha_i.P_i$. The operational semantics ensures that only one of them will be executed, discarding the rest. When $I=\emptyset$ we write $\mathbf{0}$; when $I=|2|$ we write $\alpha_1.P_1+\alpha_2.P_2$. Also, we simply write α to refer the process $\alpha.\mathbf{0}$. Process $P_1 \mid P_2$ stands for the *parallel composition* of processes P_1 and P_2. We write $\prod_{j\in J}P_j$ as a shorthand notation for process $P_1 \mid \dots \mid P_{|J|}$. Process $(\nu x)P$ declares the name x private to process P. That is, the scope of x is P; this scope may be enlarged by communication to other processes (*scope extrusion*). The conditional `if` $[x=y]$ `then` P_1 `else` P_2 is based on equality of names x and y: if $x=y$ then the process continues as P_1; otherwise it continues as P_2. By taking inputs

Figure 3 Reduction semantics for HOπ

$$\text{(COM)} \quad (\ldots + x(\widetilde{U}).P) \mid (\ldots + \overline{x}\langle \widetilde{K} \rangle.Q) \longrightarrow P\{\widetilde{K}/\widetilde{U}\} \mid Q$$

$$\text{(PAR)}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{(RES)}\ \frac{P \longrightarrow Q}{(\nu x)P \longrightarrow (\nu x)Q} \qquad \text{(STR)}\ \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

and restriction as binders, notions of free and bound names/variables arise as expected. We identify process up to consistent renaming of bound names/variables, writing $\equiv_\alpha$ for this congruence.

One way of specifying infinite process behavior is via *parametric definitions*. We write $\mathsf{D}\langle \tilde{K} \rangle$ to denote the application of a constant identifier D with parameters $\widetilde{K}$. We assume each identifier D has a unique definition $\mathsf{D}(\tilde{U}) \overset{def}{=} P$, where $\widetilde{U}$ is composed of all free names or variables in P, i.e. names or variable which occur out the scope of any binding. Similarly, $X\langle \tilde{K} \rangle$ denotes the application of parameters $\widetilde{K}$ to process variable X.

We endow our process language with a *reduction semantics*. Intuitively, a reduction $P \longrightarrow Q$ denotes a single evolution step from process P to Q, without interaction from its surrounding environment.

Definition 3.2. *Reduction, written $P \longrightarrow Q$, is the binary relation on HOπ processes defined by the rules in Figure 3.*

The rules in Figure 3 formalize process communication and reduction under parallel composition and restriction. In rule (COM), notation $P\{\widetilde{K}/\widetilde{U}\}$ stands for process P in which all free occurrences of names/variables in $\widetilde{U}$ have been substituted by names/processes in $\widetilde{K}$. We assume arity in communications is consistent, i.e., the length of $\widetilde{U}$ must be equal to the length of $\widetilde{K}$, with one-to-one correspondence among elements of both tuples. Observe that by means of rule (STR) reduction is closed under a *structural congruence* relation, written $\equiv$, which is used to promote process interactions. It is defined as follows:

Definition 3.3. *Structural congruence, written $P \equiv Q$, is the smallest process congruence such that*

$$\begin{array}{ll}
P \mid \mathbf{0} \equiv P & P \equiv_\alpha Q \Rightarrow P \equiv Q \\
P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\
(\nu x)\mathbf{0} \equiv \mathbf{0} & x \notin fn(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) \\
D(\widetilde{U}) \overset{def}{=} P \;\Rightarrow\; D\langle \widetilde{K} \rangle \;\equiv\; P\{\widetilde{K}/\widetilde{U}\} & (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \\
\texttt{if } [x=y] \texttt{ then } P_1 \texttt{ else } P_2 \;\equiv\; P_1 \ (\textit{if } x=y) & \texttt{if } [x=y] \texttt{ then } P_1 \texttt{ else } P_2 \;\equiv\; P_2 \ (\textit{if } x \neq y) \\
\sum_{i\in I} \alpha_i.P_i \;\equiv\; \sum_{j\in J} \alpha_j.P_j \ (\textit{if } J \textit{ is a permutation of } I) &
\end{array}$$

4 A Formal Model of Grid Interaction

We now present our formal model of grid computing systems. As discussed in the Introduction, it is composed of static and dynamic components. While the static component is given in terms of invariants, the dynamic part is specified using HOπ processes.

Figure 4 Base sets for GC components

GC Component	Base set
Users	$u \in U$
VOs	$v \in V$
ADs	$d \in D$
Tasks	$\mathtt{T} \in T$
User Tasks	$\mathtt{S} \in S$
Resources	$r \in R$
Nodes	$n \in N$
User Proxies	$a \in A$
Resource Proxies	$x \in X$
Resource Descriptors	$k \in K$
Logs	$l \in L$

Static Component: Base Sets and Invariants

To formalize GC systems and their components, we first relate such components to base reference sets. Then, we state associated invariant properties by defining static predicates over elements of such sets. Table 4 summarizes our notation for these base sets. The intuitive meaning of most of such sets should be clear from the description given in Section 2; we distinguish between the definition of tasks (represented as the base set T) and concrete instances of such tasks which are submitted by the users (called user tasks in base set S. For each VO we consider a group of access points (nodes) associated to it; these nodes are represented by the base set N. As for the invariants, based on informal descriptions in the literature [6], we have identified the elements that we consider essential to GC systems. Using first-order logic, we formalize such elements in terms of conditions/predicates over the elements of the reference sets. Examples of such invariants are the following:

- *Each user is member of a VO.* Using predicate $member(u,v)$, which holds if user $u \in U$ is member of VO $v \in V$, we then may write this invariant as: $\forall_{u \in U} \exists_{v \in V}\ member(u,v)$.
- *Each user has associated one task to execute in the GC system.* Using predicate $task(u,\mathtt{S})$, which holds if user $u \in U$ is the owner of task $\mathtt{S} \in S$, we then write this invariant as: $\forall_{u \in U}, \exists_{\mathtt{S} \in S}.\ task(u,\mathtt{S})$.
- *Each resource belongs to an AD.* Using predicate $belongsTo(r,d)$, which holds if resource $r \in R$ belongs to AD $d \in D$, we then may write this invariant as: $\forall_{r \in R}, \exists_{d \in D}.\ belongsTo(r,d)$.
- *Every AD can participate in one or more VOs.* Using predicate $participate(d,v)$, which holds if AD $d \in D$ participates into VO $v \in V$, we then may write this invariant as: $\forall_{d \in D}, \exists_{v_1,\ldots,v_z \in V}.\ participate(d,v_i)$.

There are also invariants concerning access points (nodes), resource descriptors, task states, resource states, and task logs; they are given in terms of appropriate base sets, and are omitted here for the sake of space. Finally, we also assume tasks can be built using the next grammar:

$$\mathtt{T} ::= \mathtt{J}\langle r_1,\ldots,r_m\rangle \;\big|\; \mathtt{T.T} \;\big|\; \mathtt{T} \oplus \mathtt{T} \;\big|\; \mathtt{T} \parallel \mathtt{T} \;\big|\; \mathtt{end}$$

Above, $\mathtt{J}\langle r_1,\ldots,r_m\rangle$ denotes a basic job J with resources $r_1,\ldots,r_m$ ($m \geq 1$). More complex tasks can be defined via sequential and parallel composition, and via non-deterministic choice. We also assume a finalization task, denoted `end`. For simplicity, we assume that each user is associated to a single task. This is not a limitation, for tasks may involve several subtasks in parallel and sequential composition.

See [14] for a more detailed description of the static component of our model.

Figure 5 Dynamic model: Correspondence among GC components and processes (full details in [14])

GC Component	**HOπ Process**	**Intuitive Description**
Grid system	$\mathsf{Grid}^{\omega,\mu}_{\delta,\eta}$	Represents the whole GC system
User ($u \in U$)	$[\![u,\mathtt{S}]\!]^{\widetilde{c},y} = \mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y)$	Models the behavior of u to authenticate and submit its task $\mathtt{S}$
	$\hookrightarrow \mathsf{UsrMonitor}(\widetilde{c}, g, a, y, \mathsf{P})$	Monitors tasks submitted by u
Node ($n \in N$)	$[\![n]\!]^{v,y,\widetilde{d}} = \mathsf{AP}(y, \widetilde{d})$	Models the interaction of n with users to authenticate
	$\hookrightarrow \mathsf{AP\text{-}UsrHandler}(ch_1, ch_2, ce)$	Models the user proxy creation and task submission
	$\hookrightarrow \mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$	Represents the interaction with the user task
VO ($v \in V$)	Composition of instances of $\mathsf{AP}(y, \widetilde{d})$	A collection of nodes
AD ($d \in D$)	$[\![d]\!] = \mathsf{AD}(d)$	Models the AD with its resources, proxy resources and management elements
	$\hookrightarrow \mathsf{AD\text{-}RecReq}(b, d)$	Receives the tasks assigned to the AD and puts them in the queue
	$\hookrightarrow \mathsf{AD\text{-}AsgRes}(b, d, ch)$	Dequeues tasks and assigns appropriate resources to them
	$\hookrightarrow \mathsf{AD\text{-}LRM}(\widetilde{s}, \widetilde{x}, \widetilde{w}, ch, d)$	Supervises the state of resources, and determines the available resources for a task
Resource ($r \in R$)	$[\![r]\!]^{r,q} = \mathsf{AD\text{-}Resource}(r, q)$	Models a resource's behavior when is used by a task
User Proxy ($a \in A$)	$[\![a]\!]^{ce,p,t,g} = \mathsf{AP\text{-}UserProxy}(ce, p, t, g)$	Models the task management, the request of execution of subtasks and the authentication with resource proxies
Res. Proxy ($x \in X$)	$[\![x]\!]^{x,q,r,w} = \mathsf{AD\text{-}ResourceProxy}(x, q, r, w)$	Acts as a mediator between GC components and a resource
Log ($l \in L$)	$\mathsf{AP\text{-}Log}(g_r, g_w, st, \widetilde{z})$	Interacts with GC components to register the changes in the task state and result
Task ($\mathtt{T} \in T$)	$\lceil \mathtt{T} \rceil^{t,e}$ definition	Represents the behavior of a task
User Task ($\mathtt{S} \in S$)	$[\![\mathtt{S}]\!] = \lceil \mathtt{T} \rceil^{t,e}$	Models a task instance corresponding to a user task
Descriptors ($k \in K$)	Names $\mathtt{k}_1, \ldots, \mathtt{k}_K$	Models the different types of resources

Dynamic Component: Model in HOπ calculus

In addition to a static specification of its components, we should provide a dynamic specification that unambigously shows how the GC system may evolve as a result of the interaction of its components. In the light of the protocol outlined in Sect. 2, such interactions may follow intricate patterns and must adhere to basic correctness and trustworthiness criteria. We would like formal mechanisms to ensure that models indeed satisfy such criteria. As we wish to describe GC systems in a compositional way, clearly specifying the interacting mechanisms and their relationships, first-order logic is not the most appropriate formalism for this task. We then appeal to specifications expressed as HOπ processes: they offer a basis on which interaction features in GC systems can be succinctly represented, and potentially verified using proof techniques associated to HOπ (behavioral equivalences, type systems, etc.). We thus extend the static description overviewed above so as to define in HOπ the behavior of GC components and their interactions according to the invariants and predicates of the static representation. Figure 5 summarizes the correspondence between the elements in the static description and their respective process representation in the dynamic component of the model. In the figure, we use the symbol $\hookrightarrow$ to represent sub-processes which are triggered as part of the execution of a main process. A complete description of all the processes mentioned the figure can be found in [14].

In addition to the high-level correspondence in Figure 5, our process model features a correspondence between some elements in the GC scenario and names in the HOπ calculus. For example, for each resource $r \in R$ there is a name r that corresponds to the access channel to the resource. The same name also allows to refer directly the resource through such a name in its process representation. The following table presents the correspondence among GC components in the static description and the access channels in their process representation:

GC Component	**Access Channel**
$n_i \in N$	Name y_i
$d_l \in D$	Name d_l
$r_j \in R$	Name r_j
$x_j \in X$	Name x_j

Next we briefly describe process representations for some grid components (users, middleware, ADs) mentioned in Figure 5. We use ω, μ, δ, and η to denote, respectively, the number of VOs, users, ADs, and nodes (access points) in the system. Also, we rely on standard process representations of queues (and associated operations). It is worth highlighting that the HOπ representations of the GC components are related to the invariants and other elements of the static component of the model. This means that process interactions do not concern arbitrary elements of the base sets, but rather elements which may be subject to invariants. For example, a user process representation only can interact with the process representation of a node that corresponds to a VO where such user is member. Often, elements of the process language are useful to enforce invariants and/or to prevent interactions not valid by static predicates—an example is the use of fresh channels and scope extrusion in name exchanges. We are currently exploring how to formally state a correspondence between the invariants in the static description and the reductions of the dynamic representation (based on HOπ processes).

Grid system. A grid system is modeled as the parallel composition of processes representation of users, ADs, and access points. These are denoted $[\![u,\mathtt{S}]\!]^{\widetilde{c},y}$, $[\![d]\!]$, and $[\![n]\!]^{v,y,\widetilde{d}}$ corresponding to the processes $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$, $\mathsf{AD}(d)$, and $\mathsf{AP}(y,\widetilde{d})$, respectively. This structure promotes interaction: while user processes interact with access point processes through private channels $y_1,\ldots,y_\eta$, AD processes communicate with access point processes in private channels $d_1,\ldots,d_\delta$. This way, our process model of a GC system, parametric on ω, μ, δ, and η, is the following:

$$\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta} \stackrel{\text{def}}{=} (\nu\, y_{n_1},\ldots,y_{n_\eta})\big(\prod_{i\in I}[\![u_i,\mathtt{S_i}]\!]^{\widetilde{c_i},y_{node(i)}} \mid (\nu\, d_1,\ldots,d_{|D|})(\prod_{h\in H}[\![n_h]\!]^{vo(n_h),y_{n_h},\widetilde{d_h}} \mid \prod_{l\in L}[\![d_l]\!])\big)$$

where $I=\{1,\ldots,\mu\}, H=\{1,\ldots,\eta\}$, and $L=\{1,\ldots,\delta\}$ are index sets over users, access points, and ADs, respectively. In process $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$ (defined below), $[\![\mathtt{S}]\!]^{t,e}$ is a process representation of task $\mathtt{T}$ ($\mathtt{S}$ is a instance of $\mathtt{T}$) that depends on names t and e: while t is used to send subtasks requirements to the appropriate user proxy, e is used to signal task completion. Name d in $\mathsf{AD}(d)$ is used for interaction between the AD and access point processes. In $\mathsf{AP}(y,\widetilde{d})$, name y is used to interact with user processes, while $\widetilde{d}$ stands for a tuple with the access channel of the ADs in the VO associated to the access point.

Users. The process model for users, denoted $\mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y)$, is parametric on a tuple of user credentials $\widetilde{c}$, a task process $[\![\mathtt{S}]\!]^{t,e}$ (explained above), and a name y, which is used to access a grid node (an instance of process $\mathsf{AP}(y, \widetilde{d})$). Process $\mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y)$ interacts with node process $\mathsf{AP}\langle y, \widetilde{d}\rangle$ in order to authenticate to the grid, create a user proxy, and submit/monitor her task. More precisely, we have:

$$\begin{aligned}\mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y) \stackrel{\text{def}}{=}\ & (\nu\, u)(\overline{y}\,\langle \widetilde{c}, u\rangle\,.u(ch_1, -, m).\\ & \quad \mathtt{if}\ [m = \mathtt{ok}]\ \mathtt{then}\ \overline{ch_1}.ch_1(a).\overline{ch_1}\,\left\langle [\![\mathtt{S}]\!]^{t,e}\right\rangle.ch_1(g).\mathsf{UsrMonitor}\langle \widetilde{c}, g, a, y, \mathsf{P}\rangle\\ & \quad \mathtt{else}\ \mathbf{0})\end{aligned}$$

The first communication on y represents an authentication request against an authentication service deployed at $\mathsf{AP}(y, \widetilde{d})$. This service returns name $\mathtt{ok}$ (resp. $\mathtt{denied}$) if the authentication is successful (resp. failed). We write $u(ch_1, -, m)$ to denote a reception of three arguments along u, in which the second one is not relevant. Name ch_1 is used for user proxy creation and task submission: proxy creation is requested by an output signal on ch_1; then, a name a (to be used to access the user proxy) is received on ch_1; subsequently, the task can be sent: this is represented by the (higher-order) output prefix $\overline{ch_1}\langle [\![\mathtt{S}]\!]^{t,e}\rangle$. Once the task has been sent, a channel associated to the log of the submitted task is received on ch_1, and process $\mathsf{UsrMonitor}(\widetilde{c}, g, a, y, \mathsf{P})$ is launched: it abstracts the user interaction with her access point for monitoring the task just submitted. The last parameter for $\mathsf{UsrMonitor}$, process P, specifies the user behavior to be executed upon reception of the outcome of her task.

Middleware. The middleware is represented as the composition of access point processes $\mathsf{AP}(y, \widetilde{d})$. For each VO in the grid, there are some instances of access point process associated to it. An instance of $\mathsf{AP}(y, \widetilde{d})$ interacts with an instance of $\mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y)$ for authentication purposes, user proxy creation, and task submission/monitoring. Then, process $\mathsf{AP}(y, \widetilde{d})$ launches a process $\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$, discussed below, which interacts with the user proxy process.

$$\begin{aligned}\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d}) \stackrel{\text{def}}{=}\ & ce(\widetilde{k}, m, a, g).(\mathsf{AP\text{-}ProxyHandler}\langle ce, \widetilde{d}\rangle \mid\\ & \quad (\nu c, b, f)(\prod_{d_i \in \widetilde{d}} \mathsf{AP\text{-}Search}^{\widetilde{k}}\langle c, f, d_i\rangle \mid \mathsf{AP\text{-}Acc}\langle c, f, b\rangle \mid\\ & \quad b(d_1, \ldots, d_\sigma).\sum_{j \in 1\ldots\sigma} \overline{d_j}\left\langle \widetilde{k}, m, a, g\right\rangle))\end{aligned}$$

Process $\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$ is parametric on (i) name ce, which is used to receive the task requirements from the user proxy process; and (ii) tuple $\widetilde{d}$, which contains the names associated to the ADs of the VO of the access point. Once $\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$ has received on ce the tuple $\widetilde{k}$ which represents the descriptors of the required grid resources, it selects the appropriate ADs for the requested resources. We abstract this selection by processes $\mathsf{AP\text{-}Search}^K$ and $\mathsf{AP\text{-}Acc}$. Given a tuple/set of resources descriptors K, each instance of process $\mathsf{AP\text{-}Search}^K$ searches among the resources shared by an AD with resources satisfying the requirements in K. Once a suitable AD has been found, $\mathsf{AP\text{-}Search}^K$ sends the access channel of that AD to $\mathsf{AP\text{-}Acc}$, which records all such access channels. Once all instances of $\mathsf{AP\text{-}Search}^K$ have completed the search, $\mathsf{AP\text{-}Acc}$ sends such ADs to process $\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$ along name b. Then, $\mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$ non-deterministically selects an AD.

Administrative domains. As mentioned above, an AD is represented as process $\mathsf{AD}(d)$, which consists of the parallel composition of processes in charge of receiving, queuing, and attending task execution requests. Also, $\mathsf{AD}(d)$ comprises process models of resources and resource proxies (see below). For the sake of space, we only present the process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$, which is in charge of assigning the appropriate resources for the subtasks assigned to the AD. Process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ is parametric on channels b,d, and ch: it extracts a request of the queue through channels b and c, and proceeds to attend it. Then, $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ interacts with the local resource manager process through channels ch, ans_1, and ans_2 in order to determine the resources for the request. If there are available and appropriate resources for the request then $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ receives in ans_1 the access channels of the resource proxies and forwards them to the user proxy through name p. Otherwise, if there are no resources then $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ receives an input in ans_2 and sends back the request to the queue.

$$
\begin{array}{lcl}
\mathsf{AD\text{-}AsgRes}(b,d,ch) & \stackrel{\text{def}}{=} & (\nu\, n,c)(\overline{b}\langle n,c\rangle.(c(k_1,\ldots,k_\zeta,m,p,g,b'). \\
& & \quad (\nu\, o,ans_1,ans_2) \\
& & \quad\; (\overline{ch}\left\langle k_1,\ldots,k_\zeta,ans_1,ans_2\right\rangle. \\
& & \qquad (ans_1(cr_1,\ldots,cr_\zeta). \\
& & \qquad\quad \overline{p}\left\langle k_1,cr_1,...,k_\zeta,cr_\zeta,m,o\right\rangle.\mathsf{AD\text{-}AsgRes}\langle b',d,ch\rangle \\
& & \qquad + \\
& & \qquad ans_2.\overline{d}\left\langle k_1,\ldots,k_\zeta,m,p,g\right\rangle.\mathsf{AD\text{-}AsgRes}\langle b',d,ch\rangle) \\
& & \quad\; \mid o(X).X) \\
& & \quad + n.\mathsf{AD\text{-}AsgRes}\langle b,d,ch\rangle))
\end{array}
$$

Observe how also $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ features higher-order communication in its interaction with task process $[\![\mathsf{S}]\!]^{t,e}$. In fact, using a higher-order output on name o (not shown), the task process $[\![\mathsf{S}]\!]^{t,e}$ is expected to send a job to $\mathsf{AD\text{-}AsgRes}(b,d,ch)$—which is denoted by process variable X. As soon as the reception on o takes place, process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ will execute the involved job.

User and Resource Proxies. We represent user proxies as instances of a process which receives the requirements of the subtasks of the user task process $[\![\mathsf{S}]\!]^{t,e}$ and submits such requirements to an access point process. Moreover, a user proxy process interacts with process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ which sends it the channels of the resource proxies of assigned resources. Finally, the user proxy process communicates with resources proxies process in order to authenticate and obtain the direct access to resources. Resource proxies are abstracted as a process which interacts with its associated resource process and instances of user proxy process. The interaction with its associated resource process allows the resource proxy to keep track of the state of the resource, as a resource notifies its proxy when a task has been completed.

Other components. In addition to the components described above, our process models also includes representations for other components in the GC system, namely logs processes, resource processes, and queue processes. There is a log process for each user task: it is in charge of registering the current state and the result of a task. Middleware processes (access points) interact to read the log when the user process requests it. In fact, processes $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ and $\mathsf{AP\text{-}UserProxy}(ce,p,t,g)$ interact with the log process to register a new state and/or the final result. Resource processes abstract the behavior of actual grid resources. They interact with resource proxy process and task process $[\![\mathsf{S}]\!]^{t,e}$. Finally, the queue process is a process representation of a queue structure. There is a queue process for each AD, which is used to store the subtasks requests of resources assigned to the AD.

5 Revisiting The Example

We now illustrate our formal model for the illustrative scenario presented in Section 2 (see also Figure 2). The following table summarizes the base sets for this scenario:

Base Set	Description
$D \;= \{d_1, d_2, d_3\}$	Administrative domains
$U \;= \{u_1, u_2\}$	Users
$V \;= \{v_1, v_2\}$	Virtual organizations
$N \;= \{n_1, n_2\}$	Grid nodes
$R \;= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$	Resources
$K \;= \{k_1, k_2, k_3\}$	Resource descriptors
$T \;= \{\mathtt{T}_1, \mathtt{T}_2\}$	Task
$S \;= \{\mathsf{S}_1, \mathsf{S}_2\}$	User tasks
$A \;= \{a_1, a_2\}$	User proxies
$X \;= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$	Resource proxies
$L \;= \{l_1, l_2\}$	Logs
$M \;= \{free, notAvailable, busy\}$	Resource states
$W \;= \{inactive, submitted, queued, running, finished\}$	Task states

For the sake of space, we omit the full description of the static component of the model. Still, the description of the scenario given in Section 2 should provide an intuitive idea of the key valid relationships between the main grid components. We only highlight the fact that user tasks $\mathtt{S}_1$ and $\mathtt{S}_2$ are instances of tasks $\mathtt{T}_1$ and $\mathtt{T}_2$, respectively.

As for the dynamic component of the model, we would have the following HOπ process:

$$\begin{aligned}\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta} \;=\; & (\nu\, y_1, y_2)\big(\llbracket u_1, \mathsf{S}_1 \rrbracket^{\widetilde{c_1}, y_1} \;\mid\; \llbracket u_2, \mathsf{S}_2 \rrbracket^{\widetilde{c_2}, y_2} \;\mid \\ & \qquad (\nu\, d_1, d_2, d_3)(\llbracket n_1 \rrbracket^{v_1, y_1, d_1, d_2} \;\mid\; \llbracket n_2 \rrbracket^{v_2, y_2, d_2, d_3} \;\mid\; \llbracket d_1 \rrbracket \;\mid\; \llbracket d_2 \rrbracket \;\mid\; \llbracket d_3 \rrbracket)\big)\end{aligned}$$

where $\omega = 2$, $\mu = 2$, $\delta = 2$ and $\eta = 2$. By expanding the definitions of $\llbracket u, \mathsf{S} \rrbracket^{\widetilde{c}, y}$, $\llbracket n \rrbracket^{v, y, \widetilde{d}}$, and $\llbracket d_1 \rrbracket$, the above process can be equivalently stated as follows:

$$\begin{aligned}\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta} \;=\; & (\nu\, y_1, y_2)\big((\nu\, t_1, e_1)\; \mathsf{User}\langle \widetilde{c_1}, \llbracket \mathsf{S}_1 \rrbracket^{t_1, e_1}, y_1 \rangle \;\mid\; (\nu\, t_2, e_2)\; \mathsf{User}\langle \widetilde{c_2}, \llbracket \mathsf{S}_2 \rrbracket^{t_2, e_2}, y_2 \rangle \;\mid \\ & \qquad (\nu\, d_1, d_2, d_3)(\mathsf{AP}^{v_1}\langle y_1, d_1, d_2 \rangle \;\mid\; \mathsf{AP}^{v_2}\langle y_2, d_2, d_3 \rangle \mid \mathsf{AD}\langle d_1 \rangle \mid \mathsf{AD}\langle d_2 \rangle \mid \mathsf{AD}\langle d_3 \rangle)\big)\end{aligned}$$

To illustrate process evolution, we now illustrate a particular reduction sequences that originates from $\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta}$. Precisely, we show the interactions that occur when the user u_1 accesses the grid for executing its task $\mathtt{S}_1$. For this reason, we restrict to discuss the reductions for the interaction among process representation of u_1.

First, we have a reduction $\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta} \longrightarrow^{*} \mathsf{GRID}^1$, where we abstract reductions in which process $\mathsf{User}\langle \widetilde{c_1}, \llbracket \mathsf{S}_1 \rrbracket^{t_1, e_1}, y_1 \rangle$ interacts with process $\mathsf{AP}^{v_1}\langle y_1, d_1, d_2 \rangle$ to realize steps of user authentication, proxy creation, and submission of task $\mathtt{S}_1$, as stipulated in the protocol. Process GRID^1 is as follows:

$$\begin{aligned}\mathsf{GRID}^1 \;\equiv\; & (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle \widetilde{c_1}, g_{r1}, a_1, y_1, \mathsf{P} \rangle \;\mid\; (\nu\, t_2, e_2)\mathsf{User}\langle \widetilde{c_2}, \llbracket \mathsf{S}_2 \rrbracket^{t_2, e_2}, y_2 \rangle \;\mid \\ & \qquad (\nu\, d_1, d_2, d_3)(\mathsf{AP}^{v_1}\langle y_1, d_1, d_2 \rangle \;\mid\; \mathsf{AP}^1(\mathtt{S}_1) \;\mid\; \mathsf{AP}^{v_2}\langle y_2, d_2, d_3 \rangle \;\mid \\ & \qquad\qquad \mathsf{AD}\langle d_1 \rangle \;\mid\; \mathsf{AD}\langle d_2 \rangle \;\mid\; \mathsf{AD}\langle d_3 \rangle)\big)\end{aligned} \qquad (1)$$

This way, after the above mentioned steps are executed, we obtain residual processes for the user $\mathsf{UsrMonitor}\langle \widetilde{c_1}, g_{w1}, a_1, y_1, \mathsf{P} \rangle$ and the access point $\mathsf{AP}^1(\mathtt{S}_1)$. In particular, we write $\mathsf{AP}^1(\mathtt{S}_1)$ to denote the composition of process $\llbracket \mathtt{S}_1 \rrbracket^{t_1, e_1}$, the log process for this task, user proxy, as well as process

$\mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle$, which interacts with user proxy process. More in detail, the structure of $\mathsf{AP}^1(\mathtt{S}_1)$ is as follows:

$$\begin{aligned}\mathsf{AP}^1(\mathtt{S}_1) \;\equiv\; & [\![\mathtt{S}_1]\!]^{t_1,e_1} \mid (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{submitted}, \mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state}, \mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result}, \widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle)\end{aligned}$$

Private name g_{w1} is used by processes ($\mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle$) and $\mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle$ to register the changes in the state of task $\mathtt{S}_1$. Private name ce_1 stands for the channel on which processes $\mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle$ and $\mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle$ may interact. At this point, we have the reduction sequence: $\mathsf{GRID}^1 \longrightarrow^* \mathsf{GRID}^2$, which represents some reductions corresponding to the AD selection in the VO and the task submission to such AD. In this case, we assume the AD d_1 is selected for the execution of the task. Process GRID^2 is as follows:

$$\begin{aligned}\mathsf{GRID}^2 \;\equiv\; & (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle \widetilde{c_1}, g_{r1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle \widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^{v_1}\langle y_1, d_1, d_2\rangle \mid \mathsf{AP}^2(\mathtt{S}_1) \mid \mathsf{AP}^{v_2}\langle y_2, d_2, d_3\rangle \mid \\ & \mathsf{AD}^1\langle d_1\rangle \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle)\big)\end{aligned} \tag{2}$$

Above, process $\mathsf{AP}^2(\mathtt{S}_1)$ is the residual process for the access point, and process $\mathsf{AD}^1\langle d_1\rangle$ is the residual process for the representation of AD d_1. In this process, the interaction between the task process and the user proxy process has evolved to $[\![\mathtt{S}_1^1]\!]$ and $\mathsf{AP\text{-}UserProxy}_1$, respectively. Process $\mathsf{AP}^2(\mathtt{S}_1)$ is as follows:

$$\begin{aligned}\mathsf{AP}^2(\mathtt{S}_1) \;\equiv\; & [\![\mathtt{S}_1^1]\!] \mid (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{queued}, \mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state}, \mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result}, \widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}_1 \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle)\end{aligned} \tag{3}$$

Then, we have the reduction sequence $\mathsf{GRID}^2 \longrightarrow^* \mathsf{GRID}^3$,representing some reductions corresponding to resource selection and task execution. Process GRID^3 is as follows:

$$\begin{aligned}\mathsf{GRID}^3 \;\equiv\; & (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle \widetilde{c_1}, g_{w1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle \widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^{v_1}\langle y_1, d_1, d_2\rangle \mid \mathsf{AP}^3(\mathtt{S}_1) \mid \mathsf{AP}^{v_2}\langle y_2, d_2, d_3\rangle \mid \\ & \mathsf{AD}^2(\mathtt{S}_1) \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle)\big)\end{aligned} \tag{4}$$

In GRID^3, process $\mathsf{AP}^3(\mathtt{S}_1)$ corresponds to residual process for the access point; process $\mathsf{AD}^2(\mathtt{S}_1)$ is its analogous for the representation of AD d_1. Process $\mathsf{AP}^3(\mathtt{S}_1)$ and $[\![\mathtt{S}_1^2]\!]$ are as follows:

$$\begin{aligned}\mathsf{AP}^3(\mathtt{S}_1) \;\equiv\; & [\![\mathtt{S}_1^2]\!] \mid (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{running}, \mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state}, \mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result}, \widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle\rangle \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle) \\ [\![\mathtt{S}_1^2]\!] \;\equiv\; & \overline{e_1}\langle \widetilde{res_1}\rangle\end{aligned}$$

Process $[\![\mathtt{S}_1^2]\!]$ stands for the residual process for the task process of user u_1; it notifies its completion through channel e_1. We obtain the reduction sequence $\mathsf{GRID}^3 \longrightarrow^* \mathsf{GRID}^4$ after some reductions corresponding to task finalization and log registering. Process GRID^4 is as follows:

$$\begin{aligned}\mathsf{GRID}^4 \;\equiv\; & (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle \widetilde{c_1}, g_{w1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle \widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^{v_1}\langle y_1, d_1, d_2\rangle \mid \mathsf{AP}^4(\mathtt{S}_1) \mid \mathsf{AP}^{v_2}\langle y_2, d_2, d_3\rangle \mid \\ & \mathsf{AD}^3(\mathtt{S}_1) \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle)\big)\end{aligned} \tag{5}$$

where $\mathsf{AP}^4(\mathtt{S}_1)$ is as follows:

$$\begin{aligned}\mathsf{AP}^4(\mathtt{S}_1) \;\equiv\; & (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{finished}, \widetilde{res_1}\rangle \mid \\ & \mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle\rangle \mid \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle)\end{aligned}$$

At last, we infer the reduction sequence $\mathsf{GRID}^4 \longrightarrow^* \mathsf{GRID}^5$,with process GRID^5 defined as follows:

$$\begin{aligned}\mathsf{GRID}^5 \;\equiv\; & (\nu\, y_1, y_2)\big(\mathsf{P}\langle\widetilde{res_1}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle\widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^4(\mathtt{S}_1) \mid \mathsf{AP}^{v_2}\langle y_2, d_2, d_3\rangle \mid \mathsf{AD}^3(\mathtt{S}_1) \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle)\big)\end{aligned} \tag{6}$$

where process $\mathsf{P}\langle\widetilde{res_1}\rangle$ denotes an unspecified, parameterized process that is to be executed by the user monitor with the task result $\widetilde{res_1}$. Such a process may correspond to, for instance, a query that stores $\widetilde{res_1}$ in some remote database.

6 Future Work

The process model of GC systems presented here describes basic interactions among grid main components, abstracting essential static and dynamic properties of such systems. There are some aspects, such as time, which are typical of GC infrastructures and that our process models does not yet take into account. Still, we think our current model is already a good basis for extensions: the inherent compositionality of process specifications should ease incremental extensions. In this sense, as future work, we will refine the model to represent locations (i.e., distinguished computation sites) and process failures. To this end, an initial approach would be using a calculus of *adaptable processes* [2], which enables to incorporate forms of runtime process adaptation over located, interacting processes.

A strong motivation for pursuing a process calculi model of GC systems is that of exploiting the proof techniques over processes (behavioral equivalences, type systems) so as to reason about grid systems. That is, we would like to explore how our process model allows us to reason about correctness properties of GC systems. This involves, for instance, exploiting our model's compositionality and well-established theories of behavioral equivalence to reason about arbitrary behaviors in the grid setting. Also, we would like to reason about task termination and resource delivery in the grid setting. These properties are intrinsically related to reachability problems, and deadlock- and cycle-detection problems. We believe that a process calculi model offers a suitable basis also for investigating such problems.

References

[1] Carmen Bratosin, Wil Aalst, Natalia Sidorova & Nikola Trčka (2008): *A Reference Model for Grid Architectures and Its Analysis*. In: *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, Springer-Verlag, Berlin, Heidelberg, pp. 898–913, doi:10.1007/978-3-540-88871-0_63. Available at http://dx.doi.org/10.1007/978-3-540-88871-0_63.

[2] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2012): *Adaptable processes*. *Logical Methods in Computer Science* 8(4). Available at http://dx.doi.org/10.2168/LMCS-8(4:13)2012.

[3] Nadia Busi, Maurizio Gabbrielli & Gianluigi Zavattaro (2009): *On the expressive power of recursion, replication and iteration in process calculi*. *Mathematical Structures in Computer Science* 19(6), pp. 1191–1222. Available at http://dx.doi.org/10.1017/S096012950999017X.

[4] Y. Du, C. Jiang & Y. Guo (2006): *Towards a formal model for grid architecture via Petri Nets*. Information Technology Journal 5(5), pp. 833–841. Available at `http://www.scopus.com/inward/record.url?eid=2-s2.0-33750162414&partnerID=40&md5=c85e98c9215a1644490b3e690a929941`.

[5] Ian Foster, Carl Kesselman, Gene Tsudik & Steven Tuecke (1998): *A security architecture for computational grids*. In: Proce. of CCS'98, CCS '98, ACM, New York, NY, USA, pp. 83–92, doi:10.1145/288090.288111. Available at `http://doi.acm.org/10.1145/288090.288111`.

[6] Ian Foster, Carl Kesselman & Steven Tuecke (2001): *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Int. J. High Perform. Comput. Appl. 15(3), pp. 200–222. Available at `http://dx.doi.org/10.1177/109434200101500302`.

[7] Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2009): *On the Expressiveness of Forwarding in Higher-Order Communication*. In Martin Leucker & Carroll Morgan, editors: ICTAC, Lecture Notes in Computer Science 5684, Springer, pp. 155–169. Available at `http://dx.doi.org/10.1007/978-3-642-03466-4_10`.

[8] Seyyed Mohsen Hashemi & Amid Khatibi Bardsiri (2012): *Cloud Computing Vs. Grid Computing*. ARPN Journal of Systems and Software 2(5). Available at `http://dx.doi.org/10.2168/LMCS-8(4:13)2012`.

[9] Vasileios Koutavas & Matthew Hennessy (2011): *A Testing Theory for a Higher-Order Cryptographic Language - (Extended Abstract)*. In Gilles Barthe, editor: ESOP, Lecture Notes in Computer Science 6602, Springer, pp. 358–377. Available at `http://dx.doi.org/10.1007/978-3-642-19718-5_19`.

[10] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi & Alan Schmitt (2011): *On the expressiveness and decidability of higher-order process calculi*. Inf. Comput. 209(2), pp. 198–226. Available at `http://dx.doi.org/10.1016/j.ic.2010.10.001`.

[11] Robin Milner (1989): *Comunication and Concurrency*. Prentice Hall.

[12] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, I*. Inf. Comput. 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4. Available at `http://dx.doi.org/10.1016/0890-5401(92)90008-4`.

[13] Zsolt Németh & Vaidy S. Sunderam (2003): *Characterizing Grids: Attributes, Definitions, and Formalisms*. J. Grid Comput. 1(1), pp. 9–23. Available at `http://dx.doi.org/10.1023/A:1024011025052`.

[14] Carlos A. Ramírez, Jorge A. Pérez, Jesús Aranda & Juan Francisco Díaz Frias (2013): *Towards Formal Interaction-based Models of Grid Computing Infrastructures (Extended Version)*. Available at `http://tinyurl.com/kkcadba`.

[15] Davide Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci.

[16] Davide Sangiorgi (1996): *Bisimulation for Higher-Order Process Calculi*. Inf. Comput. 131(2), pp. 141–178. Available at `http://dx.doi.org/10.1006/inco.1996.0096`.

[17] K. Stanoevska-Slabeva, T. Wozniak & S. Ristol (2009): *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Springer. Available at `http://books.google.com/books?id=TxOJI24idPYC`.

[18] Chuliang Weng, Xinda Lu & Qianni Deng (2003): *Formalizing Service Publication and Discovery in Grid Computing Systems*. In Minglu Li, Xian-He Sun, Qianni Deng & Jun Ni, editors: GCC (1), Lecture Notes in Computer Science 3032, Springer, pp. 669–676. Available at `http://dx.doi.org/10.1007/978-3-540-24679-4_118`.

[19] Li Zhan-jun, Huang Yong-zhong & Guo Shao-zhong (2009): *Using Pi-Calculus to Formalize Grid Workflow Parallel Computing Patterns*. In: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, ITNG '09, IEEE Computer Society, Washington, DC, USA, pp. 1568–1571, doi:10.1109/ITNG.2009.63. Available at `http://dx.doi.org/10.1109/ITNG.2009.63`.

[20] Jing Zhou & Guosun Zeng (2009): *A mechanism for grid service composition behavior specification and verification*. Future Generation Comp. Syst. 25(3), pp. 378–383. Available at `http://dx.doi.org/10.1016/j.future.2008.02.013`.

Using HMM in Strategic Games

Mario Benevides Isaque Lima Rafael Nader Pedro Rougemont

Systems and Computer Engineering Program and Computer Science Department

Federal University of Rio de Janeiro, Brazil

In this paper we describe an approach to resolve strategic games in which players can assume different types along the game. Our goal is to infer which type the opponent is adopting at each moment so that we can increase the players odds. To achieve that we use Markov games combined with hidden Markov model. We discuss a hypothetical example of a tennis game whose solution can be applied to any game with similar characteristics.

1 Introduction

Game theory is broadly used in several real world applications to solve situations involving conflicting interests, such as sports competitions, economics, social studies, etc.

In this paper we propose a model which maps opponent players behavior as a set of states (types), each state having a pre-defined payoff table, in order to infer opponents next move. We address the problem in which the states cannot be directly observed, but instead, they can be estimated from the observations of the players actions.

The rest of this paper is organized as follows. In section 3, we present the necessary background in Hidden Markov Model. In section 4 we introduce our model and in section 5 we illustrate it using a tennis game example, whose solution can be applied to any game with similar characteristics. In appendix A, we provide a Game Theory tool kit for the reader not familiar with this subject.

2 Related Work

In this paper we try to solve a particular case of a problem known as Repeated Game with Incomplete Information, first introduced by Aumann and Maschler in 1960 [1], described below:

The game G is a Repeated Game where in the first round the state of Nature is chosen by a probability p and only player 1 knows this state. Player 2 knows the possible states, but doesnt know the actual state. After each round, both players know the action of each one, and play again.

In [4] it is studied Markov Chain Games with Lack of Information. In this game, instead of a probability associated to an initial state of Nature, we have a Markov Chain Transition Probability Distribution that changes the state through time. Both players know the action of each other at each round, but only one player knows the actual and past states and the payoff associated with those actions. The other player knows the Transition Probability Distribution. This game is a particular case of Markov Chain Game or Stochastic Game [6], where both players know the actual state (or the last state and the transition probability distribution).

In [4] it is presented some properties and solutions for this class of games, using the recurrence property of a Markov Chain and the Belief Matrix B.

Submitted to:
DCM 2013

Our problem is a particular case of Renaults game, where the lack of information is worse and the unaware player doesnt know the Transition Probability Distribution of the Markov Chain Game. With that, we can describe formally our problem as below:

The game G is a Repeated Game between two players where the Player 1 has a type that changes through the time. Each type has a probability distribution to other types that is independent of past states and actions. Each type introduces a different game with different payoffs. Player 1 knows his actual type, player 2 knows the types that player 1 can be, but doesnt know the actual type neither the transition between types. Both players know the action of each round.

As said before, our game is a particular case of a Markov Chain Game, so the transition between types of player 1 follows the Markov Property. But as the player 2 doesnt know the state of player 1 we cant solve this as a Markov Chain Game (or Stochastic Game), and as the player 2 doesnt know the transition between the types we cant use the Markov Chain Game with Lack of Information.

To solve this problem we propose a model to this game and a solution that involves Hidden Markov Model. We compare our results with other ways to solve a problem like that.

3 Hidden Markov Model (HMM)

A Hidden Markov Model [10, 9] is composed by hidden states, associated to random variables that describe the transitions between states. It has the Markov Property, i.e., the transitions are memoryless, independent of the past states.

The hidden chain is associated with observations. Each hidden state is described by a random variable whose possible values are the observations.

Definition 1 *A Hidden Markov Model* $\Lambda = \langle A, B, \Pi \rangle$ *consists of*

- *a finite set of states* $Q = \{q_1, ..., q_n\}$*;*
- *a transition array A, storing the probability of the current state be* q_i *given that the previous state was* q_j*;*
- *a finite set of observation states* $O = \{o_1, ..., o_k\}$*;*
- *a transition array B, storing the probability of the observation state* o_i *be produced from state* q_j*;*
- *a* $1 \times n$ *initial array* Π*, storing the initial probability of the model be in state* q_i*.*

Based on a given set of observations y_i, and a predefined set of transitions b_{ij}, the Hidden Markov Models framework allows us to estimate the hidden transitions, labeled a_{ij} in the figure above.

4 Hidden Markov Game (HMG)

In this section, we describe the Hidden Markov Game (HMG), which is inspired by the works discussed in section 2. We first define it for many player and then restrict it to two players.

Definition 2 *A Hidden Markov Game G consists of*

- *A finite set of players* $P = \{1, ..., I\}$*;*
- *Strategy sets* $S_1, S_2, ...S_I$*, one for each player;*
- *Type sets* $T_1, T_2, ...T_I$*, one for each player;*

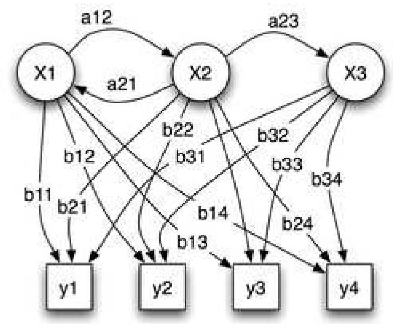


Figure 1: HMM Example

- *A transition functions $\mathscr{T}_i : T_i \times S_1 \times ... \times S_I \mapsto \mathscr{PD}(T_i)$, one for each player;*
- *A finite set of observation state variables $\mathbf{O} = \{\mathscr{O}_1, ..., \mathscr{O}_k\}$;*
- *Payoff functions $u_i : S_1 \times S_2 \times ... \times S_I \times T_i \mapsto \Re$, one for each player;*
- *Probability Distribution $\pi_{i,j}$, representing the player i prior belief about the type of his opponent j, for each player j.*

Where $\mathscr{PD}(T_i)$ is a probability distribution over T_i.

We are interested in a particular case of the HMG where we have two players and players 1 does not know his transitions function $\mathscr{T}_i$. And the set of observation state variables $\mathbf{O} = S_2$

Problem: Given a sequence of observations on time of the observation state variables $O^{t_0}, ..., O^{t_m}$, we want to know the probability, for player i, that $O^{t_{m+1}} = \mathscr{O}_j$, for $1 \leq j \leq k$.

In fact, a Hidden Markov Game can be seem as a Markov Game where a player does not know the probability distributions of the transitions between types. Instead, he knows a sequence of observation on time of the observable behavior of each opponent.

Due to this particularity, we cannot use the standard Markov Game tools to solve the game. The aim of our model is to provide a solution for the game partitioning the problem. In order to achieve that we infer the Markov chain at each turn of the game and then play in accordance to this Markov Game. In fact, if we had a one turn game, then we would have a Bayesian Game and we could calculate the Bayesian Nash Equilibrium and play according to it. This is what has been done in [11].

The inference of the transitions of the Markov chain is accomplished using a Hidden Markov Model HMM. The probability distribution associating each state of the Markov chain to the observable states are given by the Mixed Equilibrium of each matrix in each type. A Baum Welch Algorithm is used to infer the HMM.

Our Solution: Given a Hidden Markov Game G and a sequence of observations on time of the observation state variables $O^{t_0}, ..., O^{t_m}$, we want to calculate the probability, for player i, that $O^{t_{m+1}} = \mathscr{O}_j$, for $1 \leq j \leq k$. This is accomplished following the steps below:

1. Represent the Hidden Markov Game G as a Hidden Markov Model each for player i as follows:
   (a) *the finite set of states $Q = \{q_1, ..., q_n\}$* is the set the set $T_i \times S_1 \times ... \times S_I$;

(b) *the transition array A, storing the probability of the current state be q_j given that the previous state was q_l* is what we want to infer;

(c) *the finite set of observation states* $O = \{o_1, ..., o_k\}$ is the set of observation state variables $\mathbf{O} = \{\mathscr{O}_1, ..., \mathscr{O}_k\}$;

(d) *the transition array B, storing the probability of the observation state $\mathscr{O}_h$ be produced from state* $q_j =< T_j, s_1, ..., s_I >$ is obtained calculating the probability of player i using strategy $< s_1, ..., s_I >$ in the Mixed Nash Equilibrium of game T_j and adding the probabilities, for each profile $\vec{s}^{-i} \in f_i(\mathscr{O}_h)$;

(e) *the* $1 \times n$ *initial array* Π, *storing the initial probability of the model be in state* q_j is the $\pi_{i,j}$.

2. Use the Baum Welsh algorithm to infer the matrix A;
3. Solve the underline Markov Game and find the must probable type each opponent is playing;
4. If it is one move game then choose the observable state with the greatest probability. Else, play according to the Mixed Nash Equilibrium.

5 Application and Test

In order to illustrate our frame work we present an example of a tennis game. In this example, there are three hidden states, i.e., players types (aggressive, moderate and defensive) and two possible observations (open and center).

We assume that the transitions between the hidden states and the observations are fixed, i.e., they are previously computed using the payoff matrix of each profile. We do that to reduce the number of loose variables. We compare ours results with the ones obtained by Bayesian game using the same trained HMM to compute the Bayesian output. We also compare our results with some more naive approachs like Tit-for-Tat (that always repeat the last action made by the opponent), Random Choice and More Frequently (choose the action that is more frequently used by the opponent).

We used the following payoff matrixes, with two players: player 1 (column) and player 2 (row).

	Open	Center
Open	0.65,0.35	0.89,0.11
Center	0.98,0.02	0.15,0.85

Table 1: Payoff Matrix (Aggressive Profile)

	Open	Center
Open	0.15,0.85	0.80,0.20
Center	0.90,0.10	0.15,0.85

Table 2: Payoff Matrix (Moderate Profile)

	Open	Center
Open	0.10,0.90	0.55,0.45
Center	0.85,0.15	0.05,0.95

Table 3: Payoff Matrix (Defensive Profile)

We compute the mixed strategy to each profile, and consequentially compute the values of B.

Next we present some scenarios that we use.

5.1 Scenarios

In this section we present some of the scenarios that we use.

The original HMM is only used to generate the observations of the game. We generate 10.000 observations, for each scenario, and after every 200 observations we compute the result of the game.

We used the metric proposed in [9] to compare the similarity of two Markov models.

5.1.1 Scenario 1 Aggressive Player

In order to illustrate the behavior of an aggressive player we model a HMM that has more probability to stay in the aggressive state.

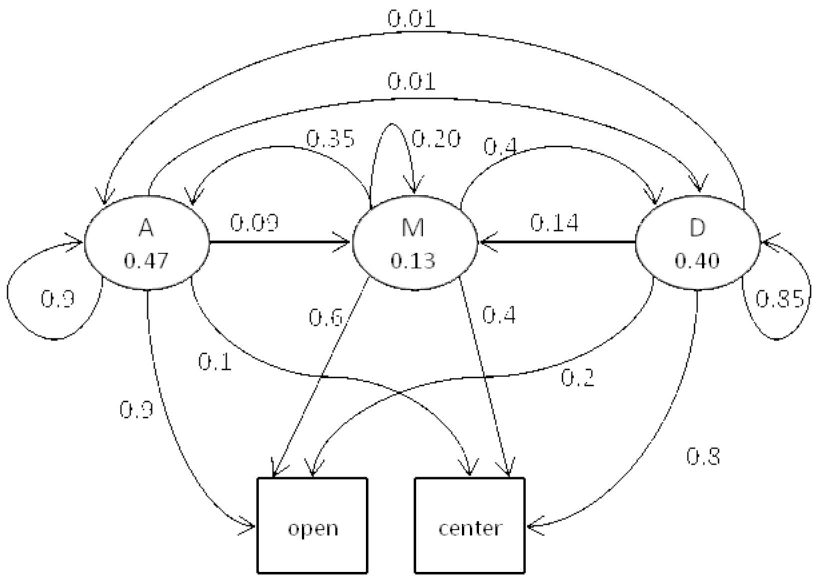


Figure 2: Original HMM (Aggressive Player)

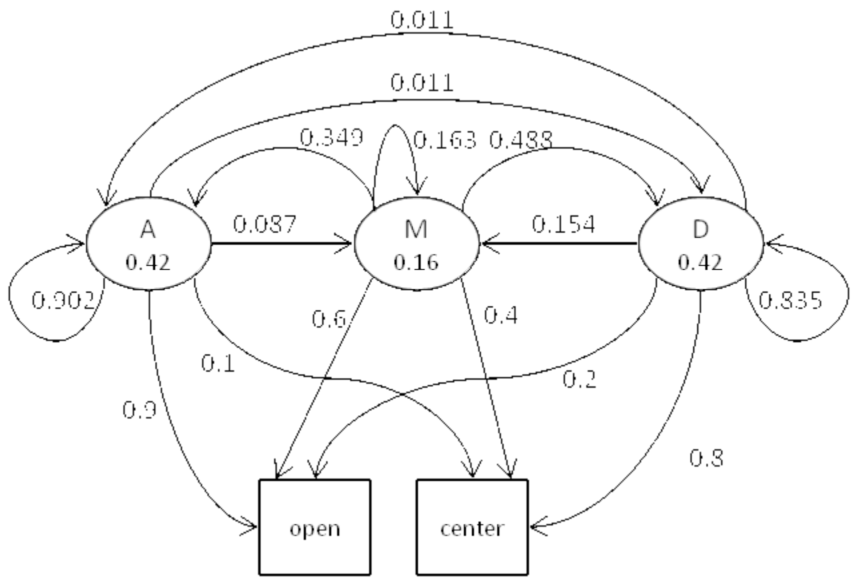


Figure 3: Trained HMM (Aggressive Player)

The difference between the original HMM and de trained HMM was 0,0033, i.e, the two HMM are very close, as we can empiric see in the picture below. As we can see on the table below, the proposed algorithm increase the player odds.

	Proposed Model	Bayesian Game	Random	More Frequently	Tit-for-Tat
hit rate	0,78	0,58	0,496	0,582	0,532

Table 4: Agressive Scenario Hit Rate

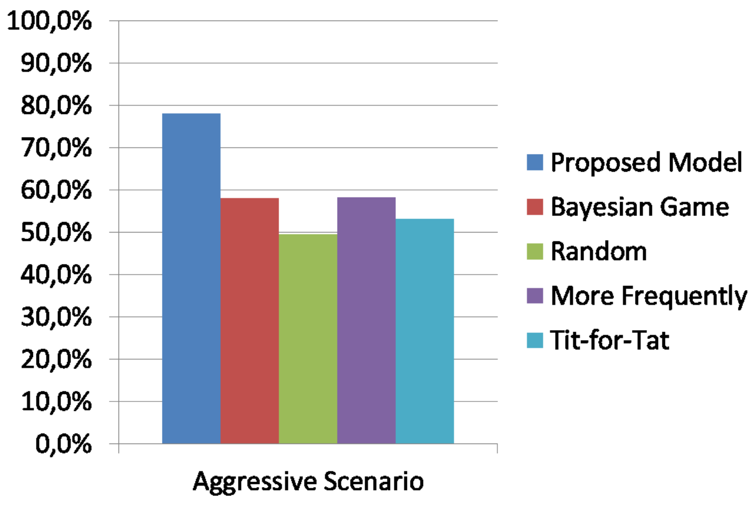


Figure 4: Agressive Scenario Graphic

5.1.2 Scenario 2 Defensive Player

In order to illustrate the behavior of a defensive player we model a HMM that has more probability to stay in the defensive state.

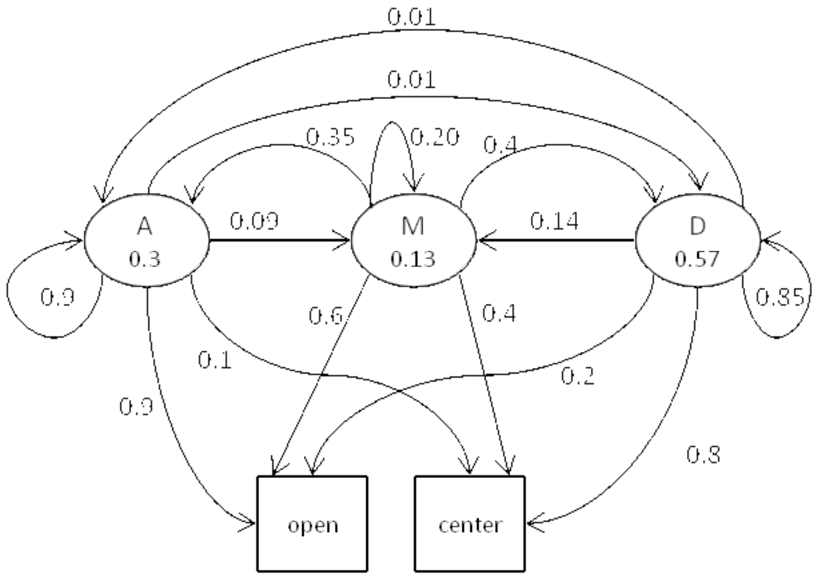


Figure 5: Original HMM (Defensive Player)

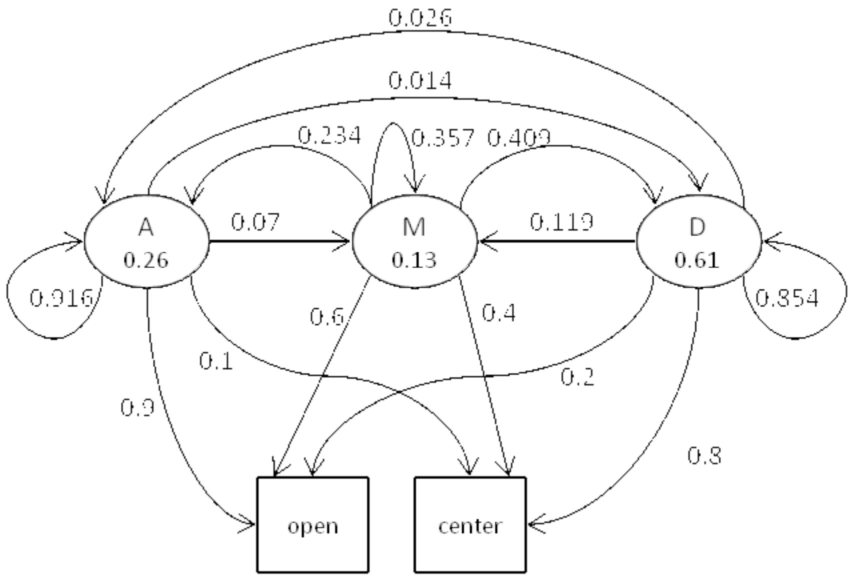


Figure 6: Trained HMM (Defensive Player)

Once again the proposed algorithm increase the player odds and the trained HMM is very close to the original HMM.

	Proposed Model	Bayesian Game	Random	More Frequently	Tit-for-Tat
hit rate	0,71	0,50	0,492	0,53	0,498

Table 5: Defensive Scenario Hit Rate

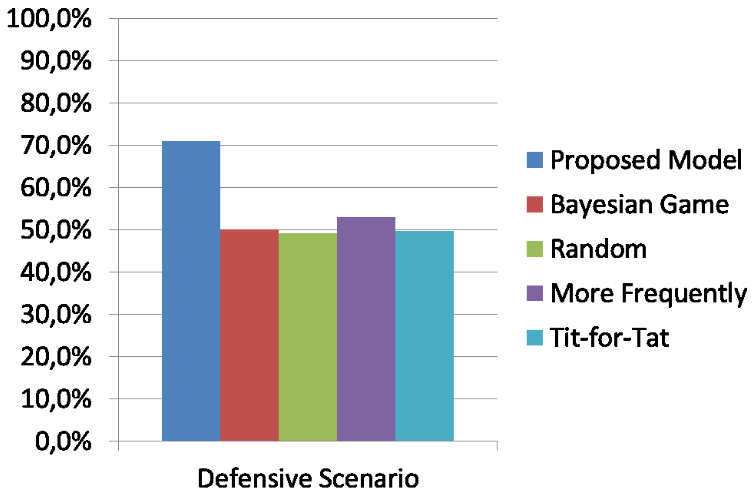


Figure 7: Defensive Player Graphic

The gain in these two scenarios are due to the fact that the proposed algorithm take into account the transitions between hidden states, so we have more information that help us to choose a better strategy.

6 Conclusions

In this work we have introduced a novel class of games called Hidden Markov Games, which can be thought as Markov Game where a player does not know the probability distributions of the transitions between types. Instead, he knows a sequence of observation on time of the observable behavior of each opponent. We propose a solution to our game representing it as a Hidden Markov Model, We use Baum Welsh algorithm to infer the probability distributions of the transitions between types. Finally, we solve the underline Markov Game.

In order to illustrate our approach, we present a tennis game example and solve it using our method. The experimental results indicates that our solution is quite good.

References

[1] R. J. Aumann and M. Maschler, *Repeated games with incomplete information*, with the collaboration of R. Stearns, Cambridge, MA: MIT Press, 1995 .

[2] M. J. Osborne and A. Rubinstein *A Couser in Game Theory*. The MIT Press, 1994.

[3] R. Gibbons. *A Prime in Game Theory*. Prentice Hall, 1992.

[4] J. Renault. The value of Markov chain games with lack of information on one side *Mathematics of Operations Research,*. vol. 31 pp. 490-512, 2006.

[5] W. He and J. Gao. A Finitely Repeated Bayesian Game with Hidden Markovian States *Third International Joint Conference on Computational Science and Optimization*. IEEE, 2010.

[6] J. Van Der Wal. Stochastic dynamic programming *In Mathematical Centre Tracts, 139*. M. Kaufmann, 1981.

[7] M. Owen. *Game Theory*. Academic Press, Orlando, Florida, Second Edition, 1982.

[8] M. L. Littman. Markov Games as a Framework for Multi-Agent Reinforcement Learning *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994.

[9] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applic. *Speech Recognition*. IEEE, 1989.

[10] L. R. Rabiner. A Probabilistic Distance Measure for Hidden Markov Models. *AT&T Tech. J.*. vol. 64, no 2, p. 391-408, 1985.

[11] E. Waghabi. Applying HMM in Mixed Strategy Game. *MSC. Th., PESC*. Federal Univ. Rio de Janeiro, 2009.

A Game Theory Tool Kit

In this section, we present the necessary background on Game theory. First we introduce the concepts of Normal Form Strategic Game and Pure and Mixed Nash Equilibrium. Finally, we define Bayesian Games and Markov Games.

Definition 3 *A Normal (Strategic) Form game G consists of*

- *A finite set of players* $P = \{1,...,I\}$*;*
- *Strategy sets* $S_1, S_2, ...S_I$*, one for each player;*
- *Payoff functions* $u_i : S_1 \times S_2 \times ... \times S_I \mapsto \Re$*, one for each player.*

A strategy profile is a tuple $s = s_1, s_2, ..., s_I$ *such that* $s \in \mathscr{S}$*, where* $\mathscr{S} = S_1 \times S_2 \times ... \times S_I$*. We denote* s_{-i} *as the profile obtained from s removing* s_i*, i.e.,* $s_{-i} = s_1, s_2, ..., s_{i-1}, s_{i+1}, ..., s_I$

The following game is an example of normal (strategic) form game with two players Rose (Row) and Collin (Column) and both have two strategies s_1 and s_2.

	s_1	s_2
s_1	3,3	2,5
s_2	5,2	1,1

Table 6: Normal Form Strategic Game

Definition 4 *A strategy profile* s^* *is a pure strategy Nash equilibrium of G if and only if*

$$u_i(s^*) \geq u_i(s_i, s_{-i})$$

for all players i and all strategy $s_i \in S_i$*, where u is the payoff function.*

Intuitively, a strategy profile is a Nash Equilibrium if for each player, he cannot improve his payoff changing his strategy alone.

The game presented in table 6 has two Nash Equilibrium (s_1, s_2) and (s_2, s_1). Which one they should play? A possible answer could be to assign probabilities to the strategies, i.e., Rose could play s_1 with probability p and s_2 with $1-p$, and Collin could play s_1 with probability q and s_2 with $1-q$. This game has a mixed Nash equilibrium that is $p = 1/3$ and $q = 1/3$. We can calculate the expected payoff for both player and check that they cannot improve their payoff changing their strategy alone, i.e., this a Nash Equilibrium.

Other class of static games are strategic games of incomplete information also called Bayesian Games [2, 3]. The intuition behind this kind of game is that in some situations the player knows his payoff function but is not sure about his opponents function. He knows that his opponent is playing according to one type in a finite set of types with some probability.

Definition 5 *A Static Bayesian game G consists of*

- *A finite set of players $P = \{1, ..., I\}$;*
- *Strategy sets $S_1, S_2, ...S_I$, one for each player;*
- *Type sets $T_1, T_2, ...T_I$, one for each player;*
- *Payoff functions $u_i : S_1 \times S_2 \times ... \times S_I \times T_i \mapsto \Re$, one for each player.*
- *Probability Distribution $P(t_{-i} \mid t_i)$, denoting the player i belief about his opponents types t_{-i}, given that his type is t_i.*
- *Probability Distribution π_i, representing the player i prior belief about the types of his opponents.*

Markov Games [6, 7, 8] can be though as a natural extension of Bayesian games, where we have transitions between types and a probability distributions over these transitions.

A Markov game can be defined as follows

Definition 6 *A Markov game G consists of*

- *A finite set of players $P = \{1, ..., I\}$;*
- *Strategy sets $S_1, S_2, ...S_I$, one for each player;*
- *Type sets $T_1, T_2, ...T_I$, one for each player;*
- *A transition functions $\mathscr{T}_i \ : \ T_i \times S_1 \times ... \times S_I \mapsto \mathscr{P}\mathscr{D}(T_i)$, one for each player;*
- *Payoff functions $u_i : S_1 \times S_2 \times ... \times S_I \times T_i \mapsto \Re$, one for each player;*
- *Probability Distribution π_i, representing the player i prior belief about the types of his opponents.*

Where $\mathscr{P}\mathscr{D}(T_i)$ is a probability distribution over T_i.