Theoretical Computer Science 404 (2008) 156–169

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On the decidability and complexity of the structural congruence for beta-binders☆

A. Romanel*, C. Priami

CoSBi, Piazza Manci 17, I-38100 Povo, Trento, Italy
Dipartimento di Ingegneria e Scienza dell'Informazione, Università degli Studi di Trento, Via Sommarive 14, I-38100 Povo, Trento, Italy

ARTICLE INFO

Keywords:
Process calculi
Structural congruence
Decidability
Complexity

ABSTRACT

Beta-binders is a recent process calculus developed for modelling and simulating biological systems. As usual for process calculi, the semantic definition heavily relies on a structural congruence. The treatment of the structural congruence is essential for implementation. We present a subset of the calculus for which the structural congruence is decidable and a subset for which it is also efficiently solvable. The obtained results are a first step towards implementations.

1. Introduction

Systems Biology studies the behaviour and relationships of the elements composing a particular biological system. Recently, some authors [2] argued that concurrency theory and *process calculi* [3,4] are useful to specify and simulate the behaviour of living matter. As a consequence, a number of process calculi have been adapted or newly developed for applications in systems biology [5–8]. Moreover, there is an increasing interest in new and correct implementation techniques for this kind of process calculi, in order to allow their execution. In particular, the definition of new computational models for stochastic process calculi allows to define new methodologies for the implementation of efficient stochastic simulators for biological processes.

Most of these process calculi are provided with stochastic extensions (i.e. quantitative information about speed of actions is provided with systems specifications) and rely on Gillespie's stochastic simulation [9,10] for analysis, an exact stochastic simulation algorithm for homogeneous, well-mixed chemical reaction systems. An example is the *Biochemical stochastic π-calculus* [5] and its stochastic simulators BioSpi [5] and SPiM [11].

In these implementations each biological entity, composing the system, is seen as a distinct π-calculus process. For example, if we simulate a biological system composed of 10.000 molecules of the species A and 10.000 molecules of the species B, these simulators instantiates 20.000 distinct processes.

The interpretation of each biological entity as a single and distinct process is not the most efficient solution to implement the Gillespie approach [9]. Indeed, each simulation step of the Gillespie's algorithm is computed using the actual propensities of reactions, which are calculated from the reactions rate constants and the multiplicities of the involved species (see [10] for details). As a consequence, the one-to-one correspondence between biological entities and processes causes an explosion of processes due to their multiplicities and not to their semantics. In other words, we will have many copies of the same process to represent the instances of the same biological entities in a given volume.

☆ Revised and extended version of [C. Priami, A. Romanel, The decidability of the structural congruence for beta-binders, Electron. Notes Theor. Comput. Sci. 171 (2) (2007) 155–170].

* Corresponding author at: CoSBi, Piazza Manci 17, I-38100 Povo, Trento, Italy.
E-mail addresses: romanel@cosbi.eu (A. Romanel), priami@cosbi.eu (C. Priami).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.04.007

Table 1
Free names of pi-processes

$fn(nil) = fn(\tau) = \emptyset$	$fn(x(y).P) = (fn(P) \cup \{x\}) \setminus \{y\}$
$fn(P \mid Q) = fn(P) \cup fn(Q)$	$fn(\bar{x}\langle y\rangle.P) = fn(P) \cup \{x, y\}$
$fn((\nu y)P) = fn(P) \setminus \{y\}$	$fn(hide(x).P) = fn(unhide(x).P) = fn(P) \cup \{x\}$
$fn(!P) = fn(P)$	$fn(expose(x, \Gamma).P) = fn(P) \setminus \{x\}$

To overcome the multiple copies problem, it makes sense to instantiate objects that represent species and to maintain for each of this object the information about its multiplicity. We obviously need to establish what a species is and to define an efficient procedure for determining whether or not a biological entity belongs to a species.

This paper focuses on *Beta-binders* [8], a process calculus thought from the beginning for biology and introduced to represent biological interaction mechanisms. In the stochastic extension of Beta-binders, a species is defined as a class of structural congruent beta-processes. For this reason, with the idea of developing a computational model for Beta-binders that considers the species, we decided first to develop on the structural congruence of the calculus, in order to establish a subset for which the structural congruence is decidable and a subset for which its evaluation is also efficiently solvable. For a more detailed description of how to realize a stochastic abstract machine for Beta-binders using this kind of technique, we refer the reader to [12].

The remainder of the paper is structured as follows. In Section 2 a short introduction to Beta-binders is reported, along with the description of some particular normal forms and an overview of the decidability of the structural congruence for the π-calculus. In Section 3 and Section 4 the proof of the decidability of the structural congruence for Beta-binders is presented. In Section 5 a generalization of the proof is given and in Section 6 a subset of Beta-binders with efficiently solvable structural congruence is presented.

All the proofs omitted in this paper can be found in [1].

2. Preliminaries

In this section we briefly review Beta-binders and the most important results regarding the decidability of the structural congruence for the π-calculus.

2.1. Beta-binders

Beta-binders [8,13] is a process algebra developed for representing the interactions between biological entities. The main idea is to encapsulate π-calculus processes into *boxes* with interaction capabilities, also called *beta-processes*. Like the π-calculus also Beta-binders is based on the notion of *naming*. Thus, we assume the existence of a countably infinite set $\mathcal{N}$ of names (ranged over by lower-case letter). The processes wrapped into boxes, also called *pi-processes*, are given by the following context free grammar:

$$P ::= nil \mid \pi.P \mid P|P \mid (\nu y)P \mid !P$$
$$\pi ::= \bar{x}\langle y\rangle \mid x(y) \mid \tau \mid expose(x, \Gamma) \mid hide(x) \mid unhide(x).$$

The syntax of the π-calculus is enriched by the last three options for π to manipulate the interactions *sites* of the boxes. Beta-processes are defined as pi-processes prefixed by specialised binders that represent interaction capabilities. An *elementary beta binder* has the form $\beta(x, \Gamma)$ (active) or $\beta^h(x, \Gamma)$ (hidden) where the name x is the subject of the beta binder and Γ represents the type of x. With $\widehat{\beta}$ we denote either β or β^h. A *well-formed* beta binder (ranged over by $\boldsymbol{B}, \boldsymbol{B}_1, \boldsymbol{B}', \ldots$) is a non-empty string of elementary beta binders where subjects are all distinct. The function $sub(\boldsymbol{B})$ returns the set of all the beta binder subjects in $\boldsymbol{B}$. Moreover, $\boldsymbol{B}^*$ denote either a well-formed beta binder or the empty string. The usual definitions of *free names* (denoted by $fn(-)$) is extended in Table 1 by stipulating that $expose(x, \Gamma).P$ is a binder for x in P. The definitions of *bound names* (denoted by $bn(-)$) and of name substitution are extended consequently.

Beta-processes (ranged over by $B, B_1, B', \ldots$) are generated by the following context free grammar:

$$B ::= Nil \mid \boldsymbol{B}[P] \mid B \parallel B.$$

The system is either the deadlock beta-process *Nil* or a parallel composition of boxes $\boldsymbol{B}[P]$. We denote by $\mathcal{P}$ and $\mathcal{BB}$, the pi-processes and the beta-processes generated by the grammar, respectively. Moreover, we denote by $\mathcal{T}$ the set of all the Beta-binders types. Free and bound names for beta-processes are defined by specifying $fn(\boldsymbol{B}[P]) = fn(P)\setminus sub(\boldsymbol{B})$.

The structural congruence for Beta-binders is defined through a structural congruence over pi-processes and a structural congruence over beta-processes.

Definition 1. The structural congruence over pi-processes, denoted by $\equiv$, is the smallest relation which satisfies the laws in Fig. 1 (group a) and the structural congruence over beta-processes, denoted by $\equiv$, is the smallest relation which satisfies the laws in Fig. 1 (group b).

Notice that the same symbol is used to denote both congruences. The intended relation is disambiguated by the context of application. Moreover, as usual two pi-processes P and Q are α-equivalent if Q can be obtained from P by renaming one or more bound names in P, and vice versa.

	Group a - **pi-processes**		Group b - **beta-processes**
$a.1)$	$P_1 \equiv P_2$ if P_1 and P_2 are α-equivalent	$b.1)$	$\mathbf{B}[P_1] \equiv \mathbf{B}[P_2]$ if $P_1 \equiv P_2$
$a.2)$	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$	$b.2)$	$B_1 \parallel (B_2 \parallel B_3) \equiv (B_1 \parallel B_2) \parallel B_3$
$a.3)$	$P_1 \mid P_2 \equiv P_2 \mid P_1$	$b.3)$	$B_1 \parallel B_2 \equiv B_2 \parallel B_1$
$a.4)$	$P \mid nil \equiv P$	$b.4)$	$B \parallel Nil \equiv B$
$a.5)$	$(\nu z)(\nu w)P \equiv (\nu w)(\nu z)P$	$b.5)$	$\mathbf{B}_1\mathbf{B}_2[P] \equiv \mathbf{B}_2\mathbf{B}_1[P]$
$a.6)$	$(\nu z)P \equiv P$ if $x \notin fn(P)$	$b.6)$	$\mathbf{B}^*\widehat{\beta}(x:\Gamma)[P] \equiv \mathbf{B}^*\widehat{\beta}(y:\Gamma)[P\{y/x\}]$ with y fresh in P and $y \notin sub(\mathbf{B}^*)$
$a.7)$	$(\nu z)(P_1 \mid P_2) \equiv P_1 \mid (\nu z)P_2$ if $z \notin fn(P_1)$		
$a.8)$	$!P \equiv P \mid !P$		

Fig. 1. Structural laws for Beta-binders.

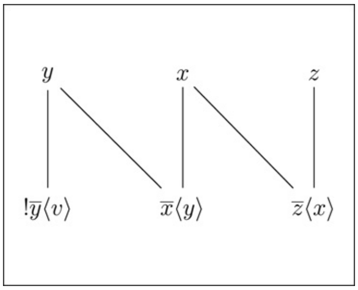

Fig. 2. Bipartite graph $os(P)$ of the process $P = (\nu x)(\nu y)((\overline{x}\langle y\rangle | !\overline{y}\langle v\rangle)|(\nu z)\overline{z}\langle x\rangle)$.

In the stochastic extension of Beta-binders [14] the syntax is enriched in order to allow a Gillespie's stochastic simulation algorithm implementation. The prefix $\pi.P$ is replaced by $(\pi, r).P$, where r is the single parameter defining an exponential distribution that drives the stochastic behaviour of the action corresponding to the prefix π.[1] Moreover, the classical replication $!P$ is replaced by the so called *guarded replication* $!\pi.P$. In order to manage this type of replication, the structural law $!P \equiv P \mid !P$ is replaced by the law $!(\pi, r).P \equiv (\pi, r).(P|!(\pi, r).P)$.

Notice that for the purpose of this paper we are not interested in the semantic of the language. We refer the reader to [8,13,14] for a more detailed description of both the qualitative and quantitative version of Beta-binders.

2.2. Normal forms

In [15] two normal forms for π-calculus processes, called *webform* and *super webform*, are introduced.

With $fn(-)$ and $bn(-)$ we indicate the usual definitions of *free* and *bound* names of π-calculus processes, with *guard* we indicate an action not prefixed by other actions and with $P \equiv_\alpha Q$ we indicate α-equivalent processes.

A process P is *fresh* if $x \notin fn(P)$ whenever (νx) is not in the scope of any guard or replication (called *outer restriction*) in P, and every restriction (νx) occurs at most once as outer restriction in P. For each process P there exists a fresh process P' such that $P' \equiv_\alpha P$. Let P be a fresh process. Let os(P), the *outer subterms* of P, be the set of occurrences of subterm $\pi.Q$ and $!Q$ of P that are not in the scope of any guard or replication. Let or(P), the *outer restrictions* of P, be the set of names x such that (νx) is not in the scope of any guard or replication in P and such that x occurs free in some outer subterm of P. Finally, let og(P), the *outer graph* of P, be the undirected bipartite graph with nodes os$(P) \cup$ or(P) and with an edge between $R \in$ os(P) and $x \in$ or(P) if $x \in fn(R)$. Consider the *fresh* process $P = (\nu x)(\nu y)((\overline{x}\langle y\rangle|!\overline{y}\langle v\rangle)|(\nu z)\overline{z}\langle x\rangle)$. The graph $og(P)$ is shown in Fig. 2.

A process $P = (\nu x_1)\ldots(\nu x_k)(P_1 \mid \cdots \mid P_m)$ with $k \geq 0$ and $m \geq 1$ is a *web* if: (1) every process P_i is a replication $!Q$ or a guarded process $\pi.Q$; (2) $x_1, \ldots, x_k$ are all distinct (P is fresh); (3) for each x_j there exists a process P_i such that $x_j \in fn(P_i)$; (4) $og(P)$ is connected. Every replication $!P$ and every guarded process $\pi.P$ is a web (with $k = 0$ and $m = 1$). No web is congruent to the inactive process *nil*. A web should be denoted with the set $\{x_1, \ldots, x_k, P_1, \ldots, P_m\}$ which lists the names of the outer restrictions and the outer subterms. A *webform* of a fresh process P, denoted with $wf(P)$, is the composition of all the webs $(\nu x_1)\ldots(\nu x_k)(P_1 \mid \cdots \mid P_m)$ such that $\{x_1, \ldots, x_k, P_1, \ldots, P_m\}$ is a connected component of $og(P)$. If $og(P)$ is the empty graph, then $wf(P) = nil$. In [15] (Lemma 3.8) an inductive decidable computation of $wf(P)$ is presented and here reported in Fig. 3.

[1] An exponential distribution with rate r is a function $F(t) = 1 - e^{-rt}$, where t is the time parameter. The parameter r determines the shape of the curve. The greater the r parameter, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time t is $F(t) = 1 - e^{-rt}$, so r determines the time t needed to obtain a probability near to 1. The exponential density function is $f(t) = re^{-rt}$.

(1) $wf(nil) = nil$, $wf(\pi.P) = \pi.P$, $wf(!P) = !P$;

(2) if $wf(P) = P_1 \mid ... \mid P_m$ e $wf(Q) = Q_1 \mid ... \mid Q_n$ with P_i and Q_j web, then $wf(P \mid Q) \equiv^{\mathbf{min}} P_1 \mid ... \mid P_m \mid Q_1 \mid ... \mid Q_n$;

(3) if $x \notin fn(P)$, then $wf((\nu x)P) \equiv^{\mathbf{min}} wf(P)$;

(4) if $x \in fn(P)$, then $wf((\nu x)P) \equiv^{\mathbf{min}} Q \mid R_1 \mid ... \mid R_n$ where,

(a) $wf(P) = Q_1 \mid ... \mid Q_m \mid R_1 \mid ... \mid R_n$ with Q_i and R_j web e $(\nu x)(Q_1 \mid ... \mid Q_m)$ fresh;

(b) $x \in fn(Q_i)$ for all i, and $x \notin fn(R_j)$ for all j;

(c) Q is a web with $Q \equiv^{\mathbf{!fr}} (\nu x)(Q_1 \mid ... \mid Q_m)$ and precisely, if $Q_i = (\nu x_{i,1})...(\nu x_{i,l_i})(Q_{i,1} \mid ... \mid Q_{i,n_i})$, then $Q = (\nu x)(\nu x_{i,1})...(\nu x_{i,l_i})(Q_{1,1} \mid ... \mid Q_{m,n_m})$

Fig. 3. Inductive computation of the *webform*.

The *super webform* of a fresh process P, denoted with $swf(P)$, is inductively defined in the following way: $swf(P) = wf(subwf(P))$ where, by definition, $subwf(P)$ is obtained from P by replacing every outer subterm $\pi.Q$ of P with $\pi.swf(Q)$ and every outer subterm $!Q$ with $!swf(Q)$. See [15] for a more detailed description.

2.3. The decidability of the structural congruence for the π-calculus

The most important results for the decidability of the structural congruence for the π-calculus are those presented by Engelfriet in [16] and by Engelfriet and Gelsema in [17–19,15]. They consider the syntax of the *small π-calculus* (presented in [20]) and the congruences over the set of processes generated by a subcollection of the structural laws presented in Fig. 4 (where, for our purpose, we add the congruence $\equiv^{\mathbf{min}}$). The standard structural congruence, defined in [16,17] and denoted with $\equiv^{\mathbf{std}}$, is determined by the laws (α), (1.1), (1.2), (1.3), (2.1), (2.2), (2.3) and (3.1). In [18], the middle congruence, denoted with $\equiv^{\mathbf{md}}$, was introduced to give a different view of the treatment of replication. The decidability of the middle congruence was shown in [19]. They reduce it to the decidability of extended structural congruence, denoted with $\equiv^{\mathbf{ext}}$, that was shown in [17]. In [15], instead, was shown the decidability of the replication free congruence, denoted with $\equiv^{\mathbf{!fr}}$, and the decidability of the standard congruence for the subclass of *replication restricted* processes. Formally, a process P is *replication restricted* if for every subterm $!R$ of P and every (νx) that covers $!R$ in P, if $x \in fn(R)$, then $x \in fn(S)$ for every component S of R where with component we mean a *web*. The decidability of the structural congruence for this subclass of processes is reduced to the problem of solving certain systems of linear equations with coefficients in $\mathbb{N}$.

3. Structural congruence over beta-processes

The structural laws for Beta-binders, presented in Fig. 1, are divided in two groups: the laws for pi-processes (*group a*) and the laws for beta-processes (*group b*). From law $b.1$ it turns out that the decidability of the structural congruence over pi-processes is a necessary condition for the decidability of the structural congruence over beta-processes.

The congruences that we consider in this paper are $\equiv_{bb}^{\mathbf{min}}$ and $\equiv_{bb}^{\mathbf{std}}$. Congruence $\equiv_{bb}^{\mathbf{min}}$ is generated by the structural laws of *group a* and the laws $b.1$, $b.5$ and $b.6$. Congruence $\equiv_{bb}^{\mathbf{std}}$ is generated by all the structural laws of *group a* and *group b*.

First, we prove the decidability of the congruence $\equiv_{bb}^{\mathbf{min}}$ making some assumptions: (1) we restrict the *well-formedness* definition by assuming that a *well-formed* beta binder (ranged over by $\boldsymbol{B}, \boldsymbol{B}_1, \boldsymbol{B}', \ldots$) is a non-empty string of elementary beta binders where subjects and types are all distinct; (2) we assume that the structural congruence over pi-processes is decidable, and therefore we assume that there exists a function $PI_{std} : \mathcal{P} \times \mathcal{P} \to \{true, false\}$ that accepts two pi-processes as parameters and returns *true* if the pi-processes are structural congruent, and returns *false* otherwise; (3) we assume that the types of the beta binders are defined over algebraic structures with decidable and efficiently solvable equality relation, and therefore we assume that there exists a function $Equal : \Gamma \times \Delta \to \{true, false\}$ that accepts two types as parameters and returns *true* if the types are equal, and returns *false* otherwise. We then prove the decidability of the congruence $\equiv_{bb}^{\mathbf{std}}$ always under the previous assumptions. Finally, we will analyze in detail the decidability of the structural congruence over pi-processes.

We consider two beta-processes $\boldsymbol{B}[P]$ and $\boldsymbol{B}'[P']$. We notice that the laws of *group b* related to the congruence $\equiv_{bb}^{\mathbf{min}}$ only refers to the structure of the beta binders lists $\boldsymbol{B}$ and $\boldsymbol{B}'$. In fact, the two lists are considered congruent only if they are equal (law $b.1$), or if $\boldsymbol{B}$ is a permutation of $\boldsymbol{B}'$ that satisfies the laws $b.5$ and $b.6$.

For this reason the decidability of the congruence $\equiv_{bb}^{\mathbf{min}}$ can be described through a function $BB_{\min} : \boldsymbol{B}[P] \times \boldsymbol{B}[P] \to \{true, false\}$ defined by induction on the structure of beta-processes (Table 2).

	rule	$\equiv^{\mathbf{min}}$	$\equiv^{\nu\mathbf{fr}}$	$\equiv^{\mathbf{!fr}}$	$\equiv^{\mathbf{std}}$	$\equiv^{\mathbf{md}}$	$\equiv^{\mathbf{ext}}$
(α)	$P_1 \equiv P_2$ if P_1 and P_2 are α-equivalent	+	+	+	+	+	+
(1.1)	$P \mid nil \equiv P$		+	+	+	+	+
(1.2)	$P_1 \mid P_2 \equiv P_2 \mid P_1$	+	+	+	+	+	+
(1.3)	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$	+	+	+	+	+	+
(2.1)	$(\nu z)(\nu w)P \equiv (\nu w)(\nu z)P$	+	+	+	+	+	+
(2.2)	$(\nu z)P \equiv P$			+	+	+	+
	if $x \notin fn(P)$						
(2.3)	$(\nu z)(P_1 \mid P_2) \equiv P_1 \mid (\nu z)P_2$			+	+	+	+
	if $z \notin fn(P_1)$						
(3.1)	$!P \equiv P \mid !P$		+		+	(+)	+
(3.6)	$!(P \mid Q) \equiv !(\ P \mid Q) \mid P$					+	(+)
(3.2)	$!(P \mid Q) \equiv !P \mid !Q$						+
(3.3)	$!!P \equiv !P$						+
(3.4)	$!nil \equiv nil$						+
(2.4)	$(\nu x)\pi.P \equiv \pi.(\nu x)P$						+
	if $x \notin n(\pi)$						+

Fig. 4. Structural laws for the π-calculus.

Table 2
Definition of function BB_{min}

$$BB_{\text{min}}(\epsilon[P], \epsilon[P']) = PI_{std}(P, P')$$
$$BB_{\text{min}}(\epsilon[P], \boldsymbol{B}'[P']) = BB_{\text{min}}(\boldsymbol{B}[P], \epsilon[P']) = false$$
$$BB_{\text{min}}(\widehat{\beta}(x : \Gamma)\boldsymbol{B}^*[P], \boldsymbol{B}'[P']) = \begin{cases} BB_{\text{min}}(\boldsymbol{B}^*[P\{z/x\}], \boldsymbol{B}_1^*\boldsymbol{B}_2^*[P'\{z/y\}]) & \text{if } (1) \\ BB_{\text{min}}(\boldsymbol{B}^*[P], \boldsymbol{B}_1^*\boldsymbol{B}_2^*[P']) & \text{if } (2) \\ false & o.w. \end{cases}$$

(1) $\boldsymbol{B}' = \boldsymbol{B}_1^*\widehat{\beta}(y : \Delta)\boldsymbol{B}_2^*$ *with* ($Equal(\Gamma, \Delta) = true$) *and* $(x \neq y)$ *and* $z \notin (fn(P) \cup fn(P') \cup sub(\boldsymbol{B}^*) \cup sub(\boldsymbol{B}_1^*\boldsymbol{B}_2^*))$

(2) $\boldsymbol{B}' = \boldsymbol{B}_1^*\widehat{\beta}(x : \Delta)\boldsymbol{B}_2^*$ *with* $Equal(\Gamma, \Delta) = true$

If the lists $\boldsymbol{B}$ and $\boldsymbol{B}'$ are not empty, then there are three different cases: (1) if a type correspondence between the first beta binders $\widehat{\beta}(x : \Gamma)$ of $\boldsymbol{B}$ and one beta binder $\widehat{\beta}(y : \Delta)$ of $\boldsymbol{B}'$ such that $(x \neq y)$ exists, then the function BB_{min} is recursively invoked on the beta-processes $\boldsymbol{B}_1[P\{z/x\}]$ and $\boldsymbol{B}_2[P'\{z/y\}]$, where $z \notin fn(P) \cup fn(P') \cup sub(\boldsymbol{B}_1) \cup sub(\boldsymbol{B}_2)$, $\boldsymbol{B}_1$ is obtained from $\boldsymbol{B}$ deleting the beta binder $\widehat{\beta}(x : \Gamma)$ and $\boldsymbol{B}_2$ is obtained from $\boldsymbol{B}'$ deleting the beta binder $\widehat{\beta}(y : \Delta)$; (2) if the first beta binder of the list $\boldsymbol{B}$ is equal to one beta binder of the list $\boldsymbol{B}'$, then the function BB_{min} is recursively invoked on the beta-processes $\boldsymbol{B}_1[P]$ and $\boldsymbol{B}_2[P']$, where $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ are respectively obtained from $\boldsymbol{B}$ and $\boldsymbol{B}'$ deleting the equal beta binders; (3) if no correspondence between the first beta binder of $\boldsymbol{B}$ and one beta binder of $\boldsymbol{B}'$ exists, then the function returns *false*.

If only one of the beta binders lists $\boldsymbol{B}$ and $\boldsymbol{B}'$ is empty, then the function returns *false*.

If both $\boldsymbol{B}$ and $\boldsymbol{B}'$ are empty, then the function PI_{std} is invoked on the pi-processes P and P'. In this case the function BB_{min} returns the result of $PI_{std}(P, P')$.

We notice that the decidability of the structural congruence over pi-processes is not only necessary condition but also sufficient condition for the decidability of the congruence $\equiv_{bb}^{\mathbf{min}}$.

Now we analyze the congruence $\equiv_{bb}^{\mathbf{std}}$. The law $b.2$ regards parallelization with the inactive beta-process *Nil* and the laws $b.3$ and $b.4$ are associativity and commutativity rules. The decidability of the congruence $\equiv_{bb}^{\mathbf{std}}$ can be described through a function $BB_{std} : B \times B \to \{true, false\}$ defined by induction on the structure of beta-processes in Table 3, where if $B' = B_1 || \cdots || B_n$ and $n = 1$ then B' is a box or the inactive beta-process *Nil*. The function $Remove : \boldsymbol{B}[P] \times B \to B$ is defined in Table 4.

If B and B' are composed of a different number of boxes, then they are not congruent and the function BB_{std} returns *false*. If there exists a bijection between the boxes $\boldsymbol{B}_i[P_i]$ of B and the boxes $\boldsymbol{B}_j'[P_j']$ of B' such that for each correspondence it is $\boldsymbol{B}_i[P_i] \equiv_{bb}^{\mathbf{min}} \boldsymbol{B}_j'[P_j']$, then the two beta-processes are congruent and the function returns *true*. Otherwise the function returns *false*.

Lemma 2. *The decidability of the structural congruence over pi-processes is a necessary and sufficient condition for the decidability of the structural congruence over beta-processes.*

Table 3
Definition of function BB_{std}

$$BB_{std}(Nil, B') = \begin{cases} false & \mathit{if}\ (1) \\ true & o.w. \end{cases}$$

$$BB_{std}(\boldsymbol{B}_1[P_1], B') = \begin{cases} BB_{std}(Nil, Remove(\boldsymbol{B}''[P''], B')) & \mathit{if}\ (2) \\ false & o.w. \end{cases}$$

$$BB_{std}(\boldsymbol{B}_1[P_1] \mid\mid B, B') = \begin{cases} BB_{std}(B, Remove(\boldsymbol{B}''[P''], B')) & \mathit{if}\ (2) \\ false & o.w. \end{cases}$$

$$BB_{std}(Nil \mid\mid B, B') = BB_{std}(B, B')$$

(1) $\exists\, j, n \in \mathbb{N}^+$ *with* $(B' = B_1 || \cdots || B_n)$ *and* $(j \leq n)$ *and* $(B_j = \boldsymbol{B}''[P''])$

(2) $\exists\, j, n \in \mathbb{N}^+$ *with* $(B' = B_1 || \cdots || B_n)$ *and* $(j \leq n)$ *and* $(B_j = \boldsymbol{B}''[P''])$ *and* $(BB_{\min}(\boldsymbol{B}_1[P_1], \boldsymbol{B}''[P'']) = true)$

Table 4
Definition of function *Remove*

$$Remove(\boldsymbol{B}[P], Nil) = Nil$$

$$Remove(\boldsymbol{B}[P], \boldsymbol{B}'[P']) = \begin{cases} Nil & \mathit{if}\ \boldsymbol{B}'[P'] = \boldsymbol{B}[P] \\ \boldsymbol{B}'[P'] & o.w. \end{cases}$$

$$Remove(\boldsymbol{B}[P], B_1 || B') = \begin{cases} B' & \mathit{if}\ (1) \\ B_1 \mid\mid Remove(\boldsymbol{B}[P], B') & o.w. \end{cases}$$

(1) $(B_1 = \boldsymbol{B}''[P''])$ *and* $(\boldsymbol{B}''[P''] = \boldsymbol{B}[P])$

	rule	$\equiv^{\mathbf{min}}$	$\equiv^{\mathbf{!fr}}$	$\equiv^{\mathbf{std}}$
(α)	$P_1 \equiv P_2$ if P_1 and P_2 are α-equivalent	+	+	+
(1.1)	$P \mid nil \equiv P$		+	+
(1.2)	$P_1 \mid P_2 \equiv P_2 \mid P_1$	+	+	+
(1.3)	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$	+	+	+
(2.1)	$(\nu z)(\nu w)P \equiv (\nu w)(\nu z)P$	+	+	+
(2.2)	$(\nu z)P \equiv P$ if $x \notin fn(P)$		+	+
(2.3)	$(\nu z)(P_1 \mid P_2) \equiv P_1 \mid (\nu z)P_2$ if $z \notin fn(P_1)$		+	+
(3.1)	$!\pi.P \equiv \pi.(P \mid !\pi.P)$			+

Fig. 5. Structural laws for the small π-calculus with guarded replication.

4. Structural congruence over pi-processes

The results on which we base part of our work are those obtained from Engelfriet and Gelsema in [15] and reported in Section 2.3. In fact, the decidability of the structural congruence over beta-processes strongly depends on the structural congruence over pi-processes. Moreover, the pi-processes are *small pi-Calculus* processes with an extended set of actions, and the structural laws for the structural congruence over pi-processes are the same ones for the structural congruence over small pi-Calculus processes. Thereafter, the results presented in [15] for the standard congruence $\equiv^{\mathbf{std}}$ and the replication free congruence $\equiv^{\mathbf{!fr}}$ can also be used in this context because they do not depend on the specific types of actions contained in the processes.

Lemma 3. *The congruences $\equiv^{\mathbf{std}}_{bb}$ and $\equiv^{\mathbf{min}}_{bb}$ are decidable for the subclass of beta-processes with replication restricted pi-processes.*

We notice that this result is valid for the qualitative version of Beta-binders. Now consider the stochastic extension of Beta-binders. The classical replication is replaced with the guarded replication and hence the syntax and the structural laws for pi-processes are modified substituting respectively $!P$ with $!\pi.P$ and $!P \equiv P \mid !P$ with $!\pi.P \equiv \pi.(P \mid !\pi.P)$.[2] The Fig. 5 shows the congruences over guarded replication pi-processes that we will consider in the remainder of the paper.

A process that only uses guarded replication is, by definition, replication restricted. Therefore, the standard structural congruence over guarded replication pi-processes is decidable. More precisely, this result is valid if we consider the replication structural law $!P \equiv P \mid !P$, whereas it must be proved if we consider the replication structural law $!\pi.P \equiv \pi.(P \mid !\pi.P)$.

[2] For simplicity in the remainder of the paper we omit the rate r in the prefixes because not important for our purpose.

Table 5
Definition of function *Impl*

$Impl(nil) = nil$	$Impl(P_0 \mid P_1) = Impl(P_0) \mid Impl(P_1)$
$Impl(!\pi.P') = !Impl(\pi.P')$	$Impl(\pi.P') = \begin{cases} !\pi.Q & \text{if } (1) \\ \pi.Impl(P') & o.w. \end{cases}$
$Impl((\nu x)P') = (\nu x)Impl(P')$	
(1) $\exists\, j, n \in \mathbb{N}^+$ *s.t.* $Impl(P') = P_1 \mid \cdots \mid P_n$ *and* $1 \le j \le n$ *and* $P_j = !\pi'.R$ *and* $Q = RemovePI(P_j, Impl(P'))$ *and* $\pi.Q \equiv^{!\mathbf{fr}} \pi'.R$	

Table 6
Definition of function *RemovePI*

$$RemovePI(P, P') = \begin{cases} P_1 & \text{if } (P' = P_0 \mid P_1) \wedge (P_0 = P) \\ P_0 \mid RemovePI(P, P_1) & \text{if } (P' = P_0 \mid P_1) \wedge (P_0 \neq P) \\ nil & \text{if } (1) \\ P' & o.w. \end{cases}$$

(1) $((P' = nil)$ *or* $(P' = \pi.R)$ *or* $(P' = !\pi.R)$ *or* $(P' = (\nu x)R))$ *and* $(P = P')$

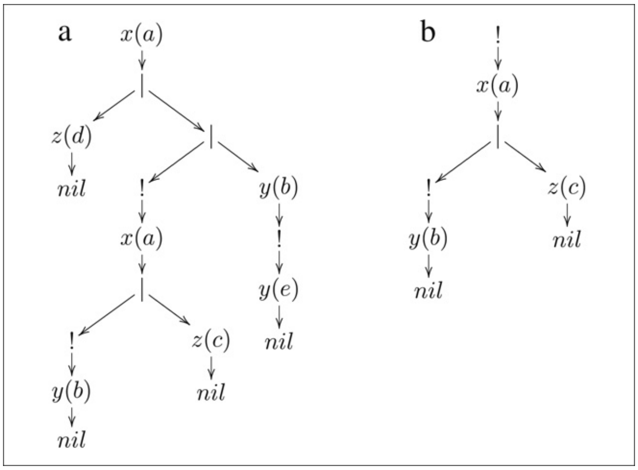

Fig. 6. Example of application of the function *Impl*: (a) Syntax tree of a pi-process $P = x(a).(z(d).nil|!x(a).(!y(b).nil|z(c).nil)|y(b).!y(e).nil)$; (b) Syntax tree of the pi-process $Impl(P)$.

In this paper we want to face the problem of decidability of structural congruence for guarded replication pi-processes from another point of view. In particular, we will consider the structure of pi-processes that only use guarded replication. In [15], the main difficulty in showing the decidability of $\equiv^{\mathbf{std}}$ for replication restricted processes is the treatment of replication, which allows a process to grow indefinitely and without particular structure in its number of subterms. A process that uses guarded replication, instead, allows a process to grow indefinitely in its number of subterms maintaining structure.
Given a generic pi-process P, this characteristic allows us to define a function that recognizes and eliminates all the expanded replication in P.

This function, that we call *Impl*, is defined by induction in Table 5, where if $Impl(P') = P_1 \mid \cdots \mid P_n$ and $n = 1$ then $Impl(P')$ is in the form nil, $\pi.R$, $!\pi.R$, or $(\nu x)R$. The function $RemovePI : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ is defined in Table 6. Since the processes have finite length the function *Impl* ends.

Let P be a pi-process. *Impl(P)* propagates *Impl* recursively to all the subterms of P. For each subterm of the form $\pi.P'$, the function controls if the recursive invocation $Impl(P')$ results in a pi-process of the form $P_1 \mid \cdots \mid P_n$ such that there exists $j \in \{1, \ldots, n\}$ where $P_j = !\pi'.R$, $Q = RemovePI(P_j, Impl(P'))$ and $\pi.Q \equiv^{!\mathbf{fr}} \pi'.R$. If it is the case, this means that the subterm $\pi.P'$ corresponds to a replication expansion and hence $\pi.P'$ can be substituted with the imploded pi-process $\pi.Q$. Obviously, the complexity of the control depends on the number of parallel components of *Impl(P')* and on the complexity of $\equiv_e^{!\mathbf{fr}}$. In particular, note that if the congruence $\equiv_e^{!\mathbf{fr}}$ is efficiently solvable, then also the function *Impl* is efficiently solvable, i.e., the complexity is polynomial in the size of the passed pi-process.

An example of how the function *Impl* works is presented in Fig. 6. In particular, given the process (Fig. 6a)

$$P = x(a).(z(d).nil|!x(a).(!y(b).nil|z(c).nil)|y(b).!y(e).nil),$$

the function recognizes that the subprocess $y(b).!y(e).nil$ is a one level expansion of the pi-process $!y(e).nil$ and compresses it. Then, the function recognizes that the whole pi-process is a one level expansion of the pi-process $!x(a).(!y(b).nil|z(c).nil)$ and returns this final pi-process (Fig. 6b), that does not contain expanded guarded replication.

Lemma 4. *Let P be a pi-process that only uses guarded replication. Then $Impl(P) \equiv^{\mathbf{std}} P$.*

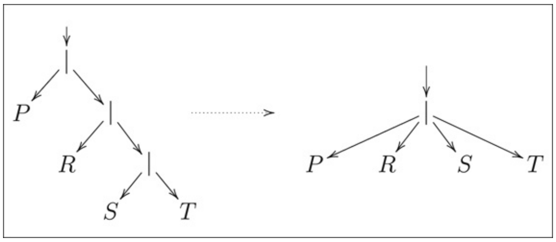

Fig. 7. Binary parallel composition transformation.

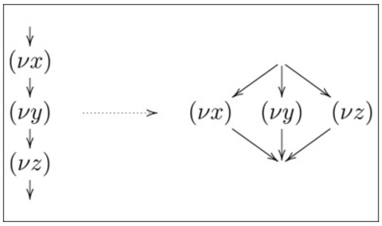

Fig. 8. Restriction sequences transformation.

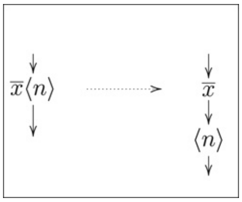

Fig. 9. Output node transformation.

Now consider the subclass of guarded replication pi-processes that does not contain expanded replications. We call this subclass $\mathcal{P}_{rp}$.

Lemma 5. *Let P and Q be pi-processes belonging to $\mathcal{P}_{rp}$. Then $P \equiv^{\mathbf{std}} Q$ iff $P \equiv^{!\mathbf{fr}} Q$.*

Lemma 6. *Let P and Q be guarded replication pi-processes. Then $P \equiv^{\mathbf{std}} Q$ iff $Impl(P) \equiv^{!\mathbf{fr}} Impl(Q)$.*

We notice that the function *Impl* is intrinsically based on the congruence relation $\equiv^{!\mathbf{fr}}$. So, we can assert that there exists a procedure that allows to verify the standard congruence over guarded replication pi-processes using only the laws of the replication free congruence. Therefore, this procedure is effectively decidable only if the replication free congruence is decidable. In [15] (Theorem 3.10) Engelfriet proves that

$$P \equiv^{!\mathbf{fr}} Q \Longleftrightarrow swf(P) \equiv_{\alpha} swf(Q)$$

where, due to some initial conventions, with $\equiv_{\alpha}$ he means $\equiv^{\mathbf{min}}$. To show in a more intuitive way that $\equiv^{!\mathbf{fr}}$ is decidable, we prove that the problem $P \equiv^{\mathbf{min}} Q$ is equivalent to an isomorphism problem over labelled directed acyclic graphs (lDAGs), that we know to be a decidable problem.

Let P be a pi-process. We define a procedure that permits to construct the lDAG, denoted with $GS(P)$, that we will use in the next proof.

Definition 7. Let P be a pi-process. The graph $GS(P)$ is built from the syntax tree of P applying the following transformations:

(1) the multiple composition of binary parallels are replaced with a unique n-ary parallel (Fig. 7);
(2) the restriction sequences are transformed as shown in Fig. 8;
(3) the output nodes, that have label $\overline{x}\langle n\rangle$, are replaced with a sequence of two nodes where the first has label $\overline{x}$ and the second has label $\langle n\rangle$ (Fig. 9);
(4) An edge is added from each node that contains a binding occurrence for a name to all the nodes that contain names bound to this occurrence (Fig. 10);
(5) Every name that binds something is replaced with 0 and every bound name is replaced with 1.

Without loss of generality we assume that 0 and 1 do not belong to the set of names $\mathcal{N}$.

The *GS* graph can be built in polynomial time and is essential for the treatment of the α-conversion and the commutativity of restrictions. Let $P = (\nu x)(\nu y)(a(x).nil \mid y(z).\overline{b}\langle z\rangle.\overline{x}\langle m\rangle.nil)$. Fig. 11 shows the building procedure of the graph $GS(P)$. With $\cong$ with denote the classical isomorphism relation between lDAGs, where the isomorphism is a bijection of nodes that maintains labels and adjacency properties.

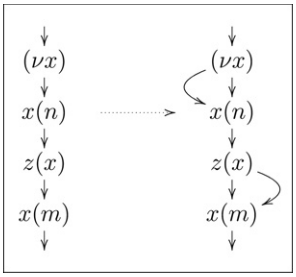

Fig. 10. Edge addition. Notice that there is no edge between the restriction (νx) and the node $x(m)$.

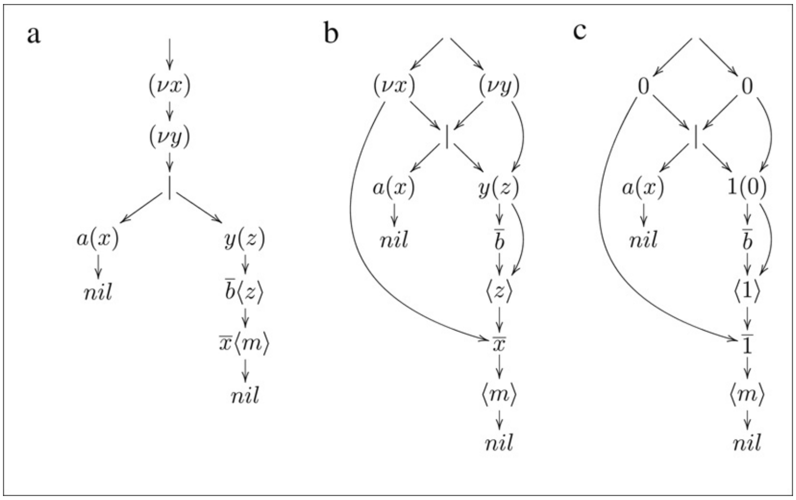

Fig. 11. Transformation of the syntax tree of the pi-process $P = (\nu x)(\nu y)(a(x).nil \mid y(z).\bar{b}\langle z\rangle.\bar{x}\langle m\rangle.nil)$ in $GS(P)$. In (a) it is shown the syntax tree of P. In (b) it is shown the application of the transformations 1, 2, 3 and 4. In (c) the transformation is completed.

Lemma 8. *Let P and Q be pi-processes. Then $P \equiv^{\mathbf{min}} Q$ iff $GS(P) \cong GS(Q)$.*

Proof. Let R be a pi-process. Then the nodes of the graph $GS(R) = (V_R, E_R)$ are enumerated with a pre-order starting from the root of the cover tree of the graph, without considering the added edges (Fig. 12). **(⇒)** We assume by hypothesis that $P \equiv^{\mathbf{min}} Q$. This means that P is obtainable from Q (and vice versa) by applying, in Q, a sequence $r_1, \ldots, r_n$ of structural laws. We denote with Q_i the pi-processes obtained from $Q = Q_0$ by applying the rules $r_1, \ldots, r_i$. Moreover, we assume that r_i supplies the information about where to apply the law in Q_{i-1}. The construction of an isomorphism ϕ_i between $GS(Q_{i-1})$ and $GS(Q_i)$ depends on the structural law r_i applied. We have three cases: **(1)** Suppose that Q_i is obtained from Q_{i-1} by applying the law (2.1) on a subterm $(\nu x)(\nu y)Q'$ of Q_{i-1}. Therefore, the only difference between Q_{i-1} and Q_i is that in Q_i the subterm $(\nu x)(\nu y)Q'$ appears in the form $(\nu y)(\nu x)Q'$. Let n_1 and n_2 be the nodes in $GS(Q_{i-1})$ that represent the restrictions (νx) and (νy), respectively, of the subterm $(\nu x)(\nu y)Q'$. In the graph Q_i the representation is inverted. In fact, n_1 represents (νy) while n_2 represents (νx). Let ϕ_i be the mapping between the nodes of $GS(Q_{i-1})$ and $GS(Q_i)$ such that for each node $n \in V_{Q_{i-1}}$ with $n \notin \{n_1, n_2\}$ is $\phi_i(n) = n$ and such that $\phi_i(n_1) = n_2$ and $\phi_i(n_2) = n_1$. ϕ_i is an isomorphism because, for the GS construction, the nodes $n \in V_{Q_{i-1}}$ and $\phi_i(n) \in V_{Q_i}$ have the same labels and for each edge $(n, n') \in E_{Q_{i-1}}$ it is $(\phi_i(n), \phi_i(n')) \in E_{Q_i}$.
(2) Suppose that Q_i is obtained from Q_{i-1} by applying the law (1.2) on a subterm $Q'|Q''$ of Q_{i-1}. Thereafter, the only difference between Q_{i-1} and Q_i is that in Q_i the subterm $Q'|Q''$ appears in the form $Q''|Q'$. Let n_0 and n_1 be the nodes in $GS(Q_{i-1})$ that represent the root node of the subgraph $GS(Q')$ and the root node of the subgraph $GS(Q'')$, respectively. In $GS(Q_i)$ the representation is inverted. In fact, n_1 represents the root node of the subgraph $GS(Q'')$ while n_2 represents the root node of the subgraph $GS(Q')$. Let ϕ_i be the mapping between the nodes of $GS(Q_{i-1})$ and $GS(Q_i)$ such that for each node $n \in V_{Q_{i-1}}$, with $n \notin \{GS(Q'), GS(Q'')\}$, it is $\phi_i(n) = n$ and such that for each node $n_1 + k$, with $k \geq 0$ and $n_1 + k \in GS(Q')$, and for each node $n_2 + j$, with $j \geq 0$ and $n_2 + j \in GS(Q'')$, it is $\phi_i(n_1 + k) = n_2 + k$ and $\phi_i(n_2 + j) = n_1 + j$. Also in this case ϕ_i is an isomorphism because, for the GS construction, the nodes $n \in V_{Q_{i-1}}$ and $\phi_i(n) \in V_{Q_i}$ have the same labels and for each edge $(n, n') \in E_{Q_{i-1}}$ it is $(\phi_i(n), \phi_i(n')) \in E_{Q_i}$.
(3) If Q_i is obtained from Q_{i-1} by applying α-conversion or the law (1.3) then the isomorphism ϕ_i is the identity *id* because, for the GS construction, the graphs $GS(Q_{i-1})$ and $GS(Q_i)$ are equal.

The composition $\phi_1 \circ \cdots \circ \phi_n$ is an isomorphism because the isomorphism relation is closed under composition and precisely it is the isomorphism between $GS(Q)$ and $GS(P)$ we wanted.
(⇐) Let P and Q pi-processes such that $GS(P) \cong GS(Q)$. We prove the implication by contradiction assuming that $P \not\equiv^{\mathbf{min}} Q$. The proof is by induction on the structure of the processes P and Q.

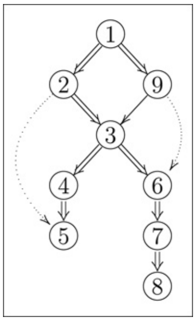

Fig. 12. Example of graph node enumeration. The double lined arrows show the cover tree of the graph. The dotted arrows represent the added edges that we do not consider.

(Induction base) Let $P = nil$. Since $P \not\equiv^{\mathbf{min}} Q$ then $Q \neq nil$ and obviously $GS(P) \ncong GS(Q)$. **(Case** $P = x(y).R$**)** if $Q \neq x(y).S$ then $GS(P) \ncong GS(Q)$ because in Q, by the graph GS construction, does not exists a node with the label and adjacency properties of the node that represent $x(y)$ in P. Otherwise, if $Q = x(y).S$ we have that $R \not\equiv^{\mathbf{min}} S$. By inductive hypothesis we obtain that $GS(R) \ncong GS(S)$ and since for each isomorphism the node that represent $x(y)$ in P should be mapped into the node that represent $x(y)$ in Q, it turns out that a total mapping does not exists and hence $GS(P) \ncong GS(Q)$. **(Case** $P = \bar{x}\langle y\rangle.R$ **and** $P = !\pi.R$**)** Similar to the previous case. **(Case** $P = R_1 \mid \cdots \mid R_n$**)** Let $P = R_1 \mid \cdots \mid R_n$ (we intend all the processes in a form like $(\cdots((R_1 \mid R_2) \mid R_3) \mid \cdots \mid R_n))$ such that R_i is not a parallel composition. If $Q \neq S_1 \mid \cdots \mid S_n$ (with S_i be not a parallel composition) then, by the graph GS construction, $GS(P) \ncong GS(Q)$. Otherwise, we have that $\exists R_i$ such that $\forall S_j$ it is $R_i \not\equiv^{\mathbf{min}} S_j$ and therefore, by inductive hypothesis, $\forall S_j$ it is $GS(R_i) \ncong GS(S_j)$. Since all the subgraphs R_i in P and S_j in Q are disjunct we obtain that $GS(P) \ncong GS(Q)$. **(Case** $P = (\nu x_1)\cdots(\nu x_n)R$**)** Let $P = (\nu x_1)\cdots(\nu x_n)R$ (with R not in the form $(\nu x)R'$). if $Q \neq (\nu y_1)\cdots(\nu y_n)S$ (with S not in the form $(\nu y)S'$) then, by the graph GS construction, $GS(P) \ncong GS(Q)$. Otherwise, we have that for each permutation of restrictions $(\nu y_1)\cdots(\nu y_n)$ and α-conversion it is $Q = (\nu x_1)\cdots(\nu x_n)T$ with $T \not\equiv^{\mathbf{min}} R$ and thus, by inductive hypothesis, $GS(R) \ncong GS(T)$. Since, by the graph GS construction, the nodes that represents $(\nu x_1)\cdots(\nu x_n)$ should be mapped into the nodes that represents $(\nu y_1)\cdots(\nu y_n)$ we have that $GS(P) \ncong GS(Q)$.

This contradict the assumption that $GS(P) \cong GS(Q)$ and therefore the implication is valid. □

The lDAG isomorphism problem [21,22] is placed in the complexity class **GI**, which contains all the problems equivalent to the general graph isomorphism problem. The class **GI** is a particular complexity class. In fact, no polynomially resolution algorithm for the problems in **GI** has been still found and it is not known if they are or not **NP-complete**. However, the congruence $\equiv^{\mathbf{min}}$ is decidable.

Theorem 9. *Let P and Q be guarded replication pi-processes. Then the evaluation of $P \equiv^{\mathbf{std}} Q$ is decidable.*

Proof. Using the Lemma 6, the Theorem 3.10 in [15] and the Lemma 8 we have that

$$\begin{array}{c} P \equiv^{\mathbf{std}} Q \\ \Longleftrightarrow \\ Impl(P) \equiv^{!\mathbf{fr}} Impl(Q) \\ \Longleftrightarrow \\ swf(Impl(P)) \equiv^{\mathbf{min}} swf(Impl(Q)) \\ \Longleftrightarrow \\ GS(swf(Impl(P))) \cong GS(swf(Impl(Q))) \end{array}$$

and therefore, for transitivity, we can conclude that

$$P \equiv^{\mathbf{std}} Q \Longleftrightarrow GS(swf(Impl(P))) \cong GS(swf(Impl(Q)))$$

where $GS(swf(Impl(P))) \cong GS(swf(Impl(Q)))$ is a decidable problem. □

Corollary 10. *Let $\boldsymbol{B}[P]$ and $\boldsymbol{B}'[P']$ be boxes where P and P' are guarded replication pi-processes. Then the evaluation of $\boldsymbol{B}[P] \equiv^{\mathbf{min}}_{bb} \boldsymbol{B}'[P]$ is decidable.*

Corollary 11. *Let B and B' be beta-processes composed by boxes with guarded replication pi-processes. Then the evaluation of $B \equiv^{\mathbf{std}}_{bb} B'$ is decidable.*

5. Generalization

Although we think that the restricted beta binder *well-formedness* definition, presented in Section 3, gives enough expressive power, in this section we briefly show that the congruence $\equiv^{\mathbf{min}}_{bb}$ for the stochastic semantics of Beta-binders is decidable also considering the classical *well-formedness* definition, given in Section 2.

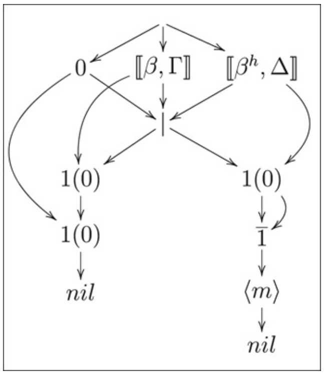

Fig. 13. lDAG $\overline{GS}$ for the box $\beta(x:\Gamma)\beta^h(y:\Delta)[(\nu z)(x(a).z(a).nil \mid y(b).\bar{b}\langle m\rangle.nil)]$.

Let $\boldsymbol{B}[P]$ and $\boldsymbol{B}'[P']$ be boxes where P and P' are guarded replication pi-processes. Moreover, let $\mathcal{L}$ be the set of the possible labels generated by the GS construction. We assume the existence of an injective, decidable and polynomial function $[\![\;]\!] : \widehat{\beta} \times \mathcal{T} \to \mathcal{S}$ where $\mathcal{S}$ is a set of strings such that $0 \notin \mathcal{S}$ and $\mathcal{S} \cap \mathcal{L} = \emptyset$. For deciding $\boldsymbol{B}[P] \equiv^{\mathbf{min}}_{bb} \boldsymbol{B}'[P']$ we construct the lDAGs $GS(Q)$ and $GS(Q')$, where $Q = swf(Impl(P))$ and $Q' = swf(Impl(P'))$, we interpret the beta binders lists $\boldsymbol{B}$ and $\boldsymbol{B}'$ as a set of top level restrictions and we put them on the top of the constructed lDAGs, modifying the bound nodes as described in Definition 7. The only difference is that a node that represents an elementary beta binder $\widehat{\beta}(x:\Gamma)$ is labelled with the result of the function $[\![\widehat{\beta},\Gamma]\!]$ instead of 0. We call the obtained graphs $\overline{GS}(\boldsymbol{B}[Q])$ and $\overline{GS}(\boldsymbol{B}'[Q'])$. In Fig. 13 an example is given.

The $\overline{GS}$ graphs can be built in polynomial time and since the graphs $\overline{GS}(\boldsymbol{B}[Q])$ and $\overline{GS}(\boldsymbol{B}'[Q'])$ differ from $GS(Q)$ and $GS(Q')$ only in the number and labels of nodes that represent restrictions, the Lemma 8 continues to hold and thus we have that:

Corollary 12. *Let $\boldsymbol{B}[P]$ and $\boldsymbol{B}'[P']$ be boxes where P and P' are guarded replication pi-processes. Then $\boldsymbol{B}[P] \equiv^{\mathbf{min}}_{bb} \boldsymbol{B}'[P']$ iff $\overline{GS}(\mathbf{B}[Q]) \cong \overline{GS}(\mathbf{B}'[Q'])$, where $Q = swf(Impl(P))$ and $Q' = swf(Impl(P'))$.*

The function BB_{std} and the Corollaries 10 and 11 can be simply redefined considering the graph $\overline{GS}$ construction.

6. An efficient subset of the calculus

In this section we introduce a subset of Beta-binders, that we call $\mathcal{BB}^e$, for which the structural congruence is not only decidable, but also efficiently solvable. In general, this subset is obtained by removing the restriction operator and considering the well-formedness definition for beta binder lists introduced in Section 3.

The syntax of $\mathcal{BB}^e$ is given by the following context-free grammar:

$$P ::= nil \mid \pi.P \mid P|P \mid\ !\pi.P$$
$$\pi ::= \bar{x}\langle y\rangle \mid x(y) \mid \tau \mid expose(x,\Gamma) \mid hide(x) \mid unhide(x)$$
$$\boldsymbol{B} ::= \beta(x:\Delta) \mid \beta^h(x:\Delta) \mid \beta(x:\Delta)\boldsymbol{B} \mid \beta^h(x:\Delta)\boldsymbol{B}$$
$$B ::= Nil \mid B||B \mid \boldsymbol{B}[P].$$

We denote with $\mathcal{P}^e$ the subset of pi-processes generated by this grammar. Obviously $\mathcal{P}^e \subset \mathcal{P}$.

Definition 13. The structural congruence over pi-processes in $\mathcal{P}^e$, denoted by $\equiv$, is the smallest relation which satisfies the laws in Fig. 1 (group a) and the structural congruence over beta-processes in $\mathcal{BB}^e$, denoted by $\equiv$, is the smallest relation which satisfies the laws in Fig. 1 (group b).

We denote with $\equiv^{\mathbf{std}}_{bbe}$ the congruence relation generated by all the structural laws reported in Fig. 14 and with $\equiv^{\mathbf{min}}_{bbe}$ the one generated by the laws of the *group a* and the laws $(b.1)$, $(b.5)$ and $(b.6)$. Moreover, we denote with $\equiv^{\mathbf{std}}_{e}$ the congruence relation generated by the structural laws of the *group a*, with $\equiv^{!\mathbf{fr}}_{e}$ the one generated by the structural laws $(a.1)$, $(a.2)$, $(a.3)$ and $(a.4)$, and with $\equiv^{\mathbf{min}}_{e}$ the one generated by the laws $(a.2)$ and $(a.3)$.

Since $\mathcal{BB}^e \subset \mathcal{BB}$, we can use all the functions and results presented in Section 3 and Section 4. To avoid confusion, when we consider a function previously defined, we modify it by substituting all the used congruence relations with the current corresponding one (i.e. $\equiv^{!\mathbf{fr}}$ is substituted with $\equiv^{!\mathbf{fr}}_{e}$). Thus, by using the results obtained in Section 3 and Section 4 we have that:

Lemma 14. *The decidability of the structural congruence over pi-processes in $\mathcal{P}^e$ is a necessary and sufficient condition for the decidability of the structural congruence over beta-processes in $\mathcal{BB}^e$.*

Lemma 15. *Let P and Q be pi-processes in $\mathcal{P}^e$. Then $P \equiv^{\mathbf{std}}_{e} Q$ if and only if $Impl(P) \equiv^{!\mathbf{fr}}_{e} Impl(Q)$.*

group a - **pi-processes**		group b - **beta-processes**	
$a.1)$	$P_1 \equiv P_2$ if P_1 and P_2 are α-equivalent	$b.1)$	$\mathbf{B}[P_1] \equiv \mathbf{B}[P_2]$ if $P_1 \equiv P_2$
$a.2)$	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$	$b.2)$	$B_1 \parallel (B_2 \parallel B_3) \equiv (B_1 \parallel B_2) \parallel B_3$
$a.3)$	$P_1 \mid P_2 \equiv P_2 \mid P_1$	$b.3)$	$B_1 \parallel B_2 \equiv B_2 \parallel B_1$
$a.4)$	$P \mid nil \equiv P$	$b.4)$	$B \parallel Nil \equiv B$
$a.5)$	$!\pi.P \equiv \pi.(P \mid !\pi.P)$	$b.5)$	$\mathbf{B}_1\mathbf{B}_2[P] \equiv \mathbf{B}_2\mathbf{B}_1[P]$
		$b.6)$	$\mathbf{B}^*\widehat{\beta}(x:\Gamma)[P] \equiv \mathbf{B}^*\widehat{\beta}(y:\Gamma)[P\{y/x\}]$
			with y fresh in P and $y \notin sub(\mathbf{B}^*)$

Fig. 14. Structural laws for $\mathcal{BB}^e$.

Table 7
Definition of function *nl*

$nl(nil, n) = nil$	$nl(x(z).P, n) = x(n).nl(P\{n/z\}, n+1)$
$nl(!\overline{x}\langle z\rangle.P, n) = !\overline{x}\langle z\rangle.nl(P, n)$	$nl(P_0 \mid P_1, n) = nl(P_0, n) \mid nl(P_1, n)$
$nl(\overline{x}\langle z\rangle.P, n) = \overline{x}\langle z\rangle.nl(P, n)$	$nl(!x(z).P, n) = !x(n).nl(P\{n/z\}, n+1)$
$nl(nil \mid P, n) = nl(P, n)$	$nl(hide(x).P, n) = hide(x).nl(P, n)$
$nl(\tau.P, n) = \tau.nl(P, n)$	$nl(expose(x).P, n) = expose(n).nl(P\{n/x\}, n+1)$
$nl(P \mid nil, n) = nl(P, n)$	$nl(unhide(x).P, n) = unhide(x).nl(P, n)$

For showing that $\equiv_e^{\mathbf{std}}$ for beta-processes in $\mathcal{BB}^e$ is efficiently solvable, we have only to show that $\equiv_e^{\mathbf{std}}$ for pi-processes in $\mathcal{P}^e$ is efficiently solvable. For doing this, we prove that the problem $P \equiv_e^{\mathbf{std}} Q$ could be reduced to an isomorphism problem over labeled trees, that we know to be a decidable and efficiently solvable problem [21].

Let P be a pi-process in $\mathcal{P}^e$. We first define a function $nl : \mathcal{P} \times \mathbb{N} \to \mathcal{P}$, based on the De Bruijs indices approach [23], that receives as parameters a pi-process P and a natural number n and returns a pi-process where all the bound names are substituted with an in-deep indexing starting from n, and where all the parallelizations with *nil* pi-processes are eliminated. The function is defined by induction on the structure of pi-processes in Table 7.

In Fig. 15 an example is given. The function *nl* has linear complexity in the length of the passed pi-process and it is easy to see that $nl(P, n) \equiv_e^{!\mathbf{fr}} P$ and that $P \equiv_\alpha Q$ implies $nl(P, n) = nl(Q, n)$ and $nl(P, n) \equiv_\alpha nl(Q, m)$, where $n, m \in \mathbb{N}$ and $n \neq m$.

Lemma 16. *Let P and Q be pi-processes in $\mathcal{P}^e$. Then $P \equiv_e^{!\mathbf{fr}} Q$ if and only if $nl(P, n) \equiv_e^{\mathbf{min}} nl(Q, n)$.*

Proof. ($\Rightarrow$) Since $nl(P, n) \equiv_e^{!\mathbf{fr}} P \equiv_e^{!\mathbf{fr}} Q \equiv_e^{!\mathbf{fr}} nl(Q, n)$ we obtain that $nl(P, n) \equiv^{!\mathbf{fr}} nl(Q, n)$. To show this implication we prove that in $nl(P, n) \equiv^{!\mathbf{fr}} nl(Q, n)$ the laws $(a.1)$ and $(a.2)$ are never used. Assume that $nl(P, n)$ is obtainable from $nl(Q, n)$ by applying, for some subterm of $nl(Q, n)$, one of the following laws: $(a.2)$ This means that one of the two processes has a subterm in the form $R \mid nil$. But this is a contradiction, because by definition of the function *nl* this subterm in $nl(P, n)$ or $nl(Q, n)$ does not exists; $(a.1)$ This means that $nl(P, n) \neq nl(Q, n)$ and, precisely, that they differ in a subset of their bound names. Since $P \equiv_\alpha nl(P, n)$ and $Q \equiv_\alpha nl(Q, n)$, we have by transitivity that $P \equiv_\alpha Q$. But this implies $nl(P, n) = nl(Q, n)$, and thus we have a contradiction.

For these reasons, the laws $(a.1)$ and $(a.2)$ are never used and the implication is true.

($\Leftarrow$) Since the structural laws of congruence $\equiv_e^{\mathbf{min}}$ are a subset of the structural laws of congruence $\equiv_e^{!\mathbf{fr}}$, by transitivity $P \equiv_e^{!\mathbf{fr}} nl(P, n) \equiv_e^{\mathbf{min}} nl(Q, n) \equiv_e^{!\mathbf{fr}} Q$ implies $P \equiv_e^{!\mathbf{fr}} Q$. $\square$

We now define a polynomial procedure that to construct a class of trees that we will use in the next proof.

Definition 17. Let P be a pi-process. The tree $TS(P)$ is built from the syntax tree of P by replacing the multiple composition of binary parallels with a unique n-ary parallel (Fig. 7).

With $\backsimeq$ we denote the classical isomorphism relation between labeled trees, where the isomorphism is a bijection of nodes that maintains label and adjacency properties.

Lemma 18. *Let P and Q be pi-processes in $\mathcal{P}^e$. Then $P \equiv_e^{\mathbf{min}} Q$ if and only if $TS(P) \backsimeq TS(Q)$.*

Proof. Let R be a pi-process in $\mathcal{P}^e$. The nodes of the tree $TS(R)$ are enumerated with a pre-order.

($\Rightarrow$) We assume by hypothesis that $P \equiv_e^{\mathbf{min}} Q$. This means that P is obtainable from Q (and vice versa) by applying, in Q, a sequence $r_1, \ldots, r_n$ of structural laws. Like in Section 4, we denote with Q_i the pi-processes obtained from $Q = Q_0$ by applying the rules $r_1, \ldots, r_i$ and we assume that r_i supplies the information about where to apply the law in Q_{i-1}. The construction of an isomorphism ϕ_i between $GS(Q_{i-1})$ and $GS(Q_i)$ depends on the structural law r_i applied.

Here we have two cases: **(1)** Suppose that Q_i is obtained from Q_{i-1} by applying the law $(a.2)$ on a subterm $Q'|Q''$ of Q_{i-1}. The proof of this case is equal to the proof of the case 2 in Lemma 8. **(2)** If Q_i is obtained from Q_{i-1} by applying α-conversion or the law $(a.3)$ then the isomorphism ϕ_i is the identity *id* because, for the *TS* construction, the trees $TS(Q_{i-1})$ and $TS(Q_i)$ are equal.

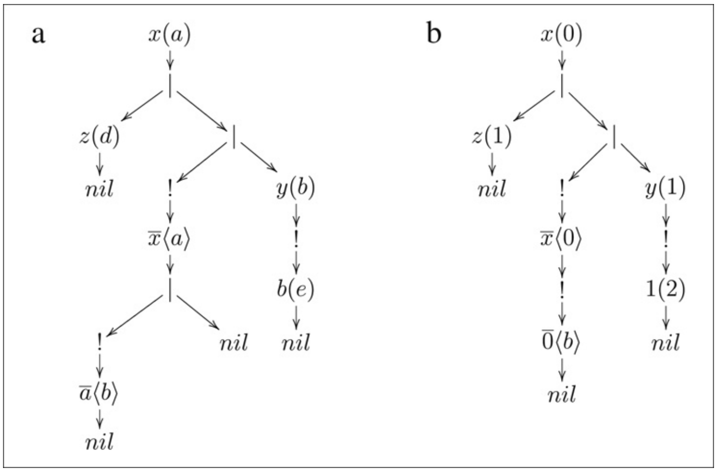

Fig. 15. Example of application of the function *nl*: (a) Syntax tree of a pi-process $P = x(a).(z(d).nil|!x(a).(!y(b).nil|z(c).nil)|y(b).!y(e).nil)$; (b) Syntax tree of the pi-process $nl(P, 0)$.

Also in this case the composition $\phi_1 \circ \cdots \circ \phi_n$ is the isomorphism between $TS(Q)$ and $TS(P)$ we wanted.

(⇐) Let P and Q pi-processes such that $TS(P) \simeq TS(Q)$. We prove the implication by contradiction assuming that $P \not\equiv_e^{\mathbf{min}} Q$. The proof is by induction on the structure of the processes P and Q and is equal to the one reported in Lemma 8. □

Since the tree isomorphism problem is efficiently solvable [21], by combining Lemmas 16 and 18 we obtain that the congruence $\equiv_e^{!\mathbf{fr}}$ is efficiently solvable, and hence that for the considered subset of Beta-binders the function *Impl* works in polynomial time (see Section 4).

Now, combining all the obtained results, we can show that the standard congruence for $\mathcal{BB}^e$ is efficiently solvable.

Theorem 19. *Let P and Q be pi-processes in $\mathcal{P}^e$. Then the evaluation of $P \equiv_e^{\mathbf{std}} Q$ is efficiently solvable.*

Proof. Using the Lemmas 15, 16 and 18 we have that

$$\begin{array}{c} P \equiv_e^{\mathbf{std}} Q \\ \Longleftrightarrow \\ Impl(P) \equiv_e^{!\mathbf{fr}} Impl(Q) \\ \Longleftrightarrow \\ nl(Impl(P), n) \equiv_e^{\mathbf{min}} nl(Impl(Q), n) \\ \Longleftrightarrow \\ TS(nl(Impl(P), n)) \simeq TS(nl(Impl(Q), n)) \end{array}$$

with $n \in \mathbb{N}$. Therefore, by transitivity, we can conclude that

$$P \equiv_e^{\mathbf{std}} Q \Longleftrightarrow TS(nl(Impl(P), n)) \cong TS(nl(Impl(Q), n))$$

where $TS(nl(Impl(P), n)) \simeq TS(nl(Impl(Q), n))$ is an efficiently solvable problem. □

Corollary 20. *Let $\boldsymbol{B}[P]$ and $\boldsymbol{B}'[P']$ be boxes in $\mathcal{BB}^e$. Then the evaluation of $\boldsymbol{B}[P] \equiv_{bbe}^{\mathbf{min}} \boldsymbol{B}'[P]$ is decidable and efficiently solvable.*

Proof. Immediate from the definition of the function $BB_{\min}$ and the Theorem 19. □

Corollary 21. *Let B and B' be beta-processes in $\mathcal{BB}^e$. Then the evaluation of $B \equiv_{bbe}^{\mathbf{std}} B'$ is decidable and efficiently solvable.*

Proof. Immediate from the definition of the function BB_{std} and the Corollary 20. □

7. Conclusions

We proved the decidability of the structural congruence used in [14] to define the stochastic semantics of Beta-binders. Moreover, we introduced a subset of Beta-binders, called $\mathcal{BB}^e$, for which the structural congruence is not only decidable, but also efficiently solvable. The proofs are constructive so that we have suggestions for possible implementations.

The subset $\mathcal{BB}^e$ has been used as a basis for the definition and implementation of the Beta Workbench,[3] a framework for modelling and simulating biological processes [24]. In particular, the results here obtained for the structural congruence of $\mathcal{BB}^e$ allowed us to consider species instead of single instances of biological entities and consequently to implement a modified version of the Gillespie's stochastic selection algorithm [10] similar to the Next Reaction Method [25].

[3] Available at the url http://www.cosbi.eu/Rpty_Soft_BetaWB.php.

Although $\mathcal{BB}^e$ imposes restrictions on the definition of beta-processes interaction sites, the effectiveness of the subset is demonstrated by its ability of being used for modelling complex biological processes like the NF-κB pathway [26,27], the MAPK cascade [28,29] and the cell cycle control mechanism [30].

Moreover, the implementation shows a consistent improvement in the stochastic simulation time efficiency. We can save up to an order of magnitude compared to standard implementations.

References

[1] C. Priami, A. Romanel, The decidability of the structural congruence for beta-binders, Electron. Notes Theor. Comput. Sci. 171 (2) (2007) 155–170.
[2] A. Regev, E. Shapiro, Cells as computation, Nature 419 (6905). http://dx.doi.org/10.1038/419343a.
[3] R. Milner, Communication and Concurrency, Prentice-Hall, Inc., 1989.
[4] C. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.
[5] C. Priami, A. Regev, E. Shapiro, W. Silvermann, Application of a stochastic name-passing calculus to representation and simulation of molecular processes, Inform. Process. Lett. 80 (1) (2001) 25–31.
[6] A. Regev, E. Panina, W. Silverman, L. Cardelli, E. Shapiro, Bioambients: An abstraction for biological compartments, Theoret. Comput. Sci. 325 (1) (2004) 141–167.
[7] L. Cardelli, Brane calculi, in: Computational Methods in Systems Biology, in: LNCS, vol. 3082, 2005, pp. 257–278.
[8] C. Priami, P. Quaglia, Beta binders for biological interactions, in: Proc. of Computational Methods in Systems Biology, in: LNCS, vol. 3082, 2005, pp. 20–33.
[9] D. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem. 81 (25) (1977) 2340–2361.
[10] D. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, J. Phys. Chem. 22 (1976) 403–434.
[11] A. Phillips, L. Cardelli, A correct abstract machine for the stochastic pi-calculus, in: Concurrent Models in Molecular Biology, 2004.
[12] A. Romanel, A stochastic abstract machine for beta-binders, Master's thesis, 2006.
[13] C. Priami, P. Quaglia, Operational patterns in Beta-binders, T. Comp. Sys. Biology 1 (2005) 50–65.
[14] P. Degano, D. Prandi, C. Priami, P. Quaglia, Beta-binders for biological quantitative experiments, Electron. Notes Theor. Comput. Sci. 164 (3) (2006) 101–117.
[15] J. Engelfriet, T. Gelsema, The decidability of structural congruence for replication restricted pi-calculus processes, Tech. Rep. 04-07, Leiden Institute of Advanced Computer Science, Leiden University, The Nederlands, 2004.
[16] J. Engelfriet, A multiset semantics for the pi-calculus with replication, Theoret. Comput. Sci. 153 (1–2) (1996) 65–94.
[17] J. Engelfriet, T. Gelsema, Multisets and structural congruence of the pi-calculus with replication, Theoret. Comput. Sci. 211 (1–2) (1999) 311–337.
[18] J. Engelfriet, T. Gelsema, Structural congruence in the pi-calculus with potential replication, Tech. Rep. 00-02, Leiden Institute of Advanced Computer Science, Leiden University, The Nederlands, 2000.
[19] J. Engelfriet, T. Gelsema, A new natural structural congruence in the pi-calculus with replication, Acta Inform. 40 (6-7) (2004) 385–430.
[20] R. Milner, The polyadic pi-calculus: A tutorial, Tech. Rep. ECSLFCS-91-180, Computer Science Department, University of Edinburgh, UK, 1991.
[21] J. Köbler, U. Schöning, J. Torán, The Graph Isomorphism Problem: Its Structural Complexity, Birkhäuser, 1993.
[22] K. Booth, C. Colbourn, Problems polynomially equivalent to graph isomorphism, Tech. Rep. CS-77-04, Department of Computer Science, University of Waterloo, Ontario, Canada, 1977.
[23] N. de Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, Indag. Math. 34 (1972) 381–392.
[24] L. Dematté, C. Priami, A. Romanel, BetaWB: Modelling and simulating biological processes, in: 2007 Summer Simulation Multiconference, 2007, pp. 777–784.
[25] M. Gibson, J. Bruck, Efficient exact stochastic simulation of chemical systems with many species and many channels, J. Phys. Chem. 104 (2000) 1876–1889.
[26] T. Gilmore, The Rel/NF-κB signal transduction pathway: Introduction, Oncogene 19 (49) (1999) 6842–6844.
[27] A. Ihekwaba, R. Larcher, R. Mardare, C. Priami, Poster abstract: BetaWB - a language for modular representation of biological SYSTEMS, in: ICSB 2007: The Eighth International Conference on Systems Biology, 2007, pp. 61–62. http://www.icsb-2007.org/proceedings/index.html.
[28] C. Huang, J. Ferrell, Ultrasensitivity in the mitogen-activated protein kinase cascade, Proc. Natl. Acad. Sci. USA 93 (19) (1996) 10078–10083.
[29] L. Dematté, C. Priami, A. Romanel, O. Soyer, A formal and integrated framework to simulate evolution of biological pathways, in: CMSB, 2007, pp. 106–120.
[30] B. Novak, A. Csikasz-Nagy, B. Gyorffy, K. Nasmyth, J. Tyson, Model scenarios for evolution of the eukaryotic cell cycle, Philos. Trans. Biol. Sci. 353 (1378) (1998) 2063–2076.