Theoretical Computer Science 411 (2010) 542–565

Contents lists available at ScienceDirect

Theoretical Computer Science

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

On the computational power of BlenX

Alessandro Romanel *, Corrado Priami

CoSBi, I-38123 Povo, TN, Italy
DISI, Università di Trento, I-38123 Povo, TN, Italy

ARTICLE INFO

Article history:
Received 23 June 2008
Received in revised form 28 September 2009
Accepted 30 September 2009
Communicated by R. Gorrieri

Keywords:
Process calculi
Random Access Machines
Turing equivalence
Termination decidability
Systems biology

ABSTRACT

We present some decidability and undecidability results for subsets of the BlenX Language, a process-calculi-based programming language developed for modelling biological processes. We show that for a core subset of the language (which considers only communication primitives) termination is decidable. Moreover, we prove that by adding either global priorities or events to this core language, we obtain Turing equivalent languages. The proof is through encodings of Random Access Machines (RAMs), a well-known Turing equivalent formalism, into our subsets of BlenX. All the encodings are shown to be correct.

1. Introduction

Systems biology [21] aims at investigating the interactions and relationships among the components of biological systems in order to understand how they work globally. Several approaches based on computational models have been used to model and analyze complex behaviours and interaction mechanisms of biological systems (e.g. boolean networks [20], Petri nets [27], statecharts [17] and membrane systems [26]).

After the work of Regev et al. [35], an emergent and promising trend is to use concurrency theory and *process calculi* 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 [33,34,7,31,11,19].

On top of these process calculi several programming languages have been defined and frameworks for analysis and stochastic simulation have been implemented [33,28,16,13].

Some of these *new languages* [34,7,31,11] differ from classical process calculi because they are devised from the beginning for biology and aim to overcome some limitations by adding or deleting primitives and operators, and by developing new conceptual tools. An interesting question is whether and how those modifications affect the ability of these languages to act as computational devices. Some examples of these investigations can be found in [4,10,8].

In this paper we consider the Beta Workbench, a framework for modelling and simulating biological processes [13,14]. It incorporates a language, a compiler to a stochastic abstract machine, an execution environment and some graphical interface components. The BlenX Language is a stochastic language (i.e. quantitative information about speed and probability of actions is provided with systems specifications) based on Beta-binders [31,32,12], a process calculus developed to represent the interactions between biological entities. In BlenX biological entities are interpreted as the components that interact in a model to accomplish some biological function: for example, proteins, enzymes, organic or inorganic compounds as well as cells or tissues; biological entities are represented through *boxes*. Boxes have interaction sites, called *binders*, and an internal

* Corresponding author at: CoSBi, Piazza Manci 17, 38123, Povo (Trento), Italy. Tel.: +39 0461282804.
E-mail addresses: romanel@cosbi.eu (A. Romanel), priami@cosbi.eu (C. Priami).

0304-3975/$ – see front matter
doi:10.1016/j.tcs.2009.09.038

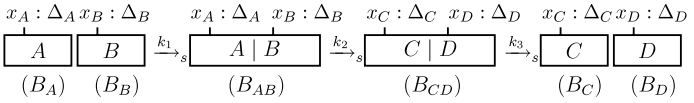


Fig. 1. BlenX model for a simple biochemical reaction.

structure as it is for biological entities. The binders represent, for example, protein domains or cell receptors and the internal program codifies the response to an external stimulus.

The main goal of this paper is to investigate the computational power of the nondeterministic version of BlenX, i.e., we do not consider here the stochastic aspect. Turing equivalence results for well-known process calculi like π-calculus [24,38] and Mobile Ambients [5,22] rely on encodings of Turing equivalent formalisms using some high-powered features like restriction operator and name passing in combination with operators like replication, recursion or recursive definitions. In BlenX the restriction operator is not present and the replication is guarded by an action; hence none of the classic results can be directly applied. For these reason, we decided to start by first developing on a core subset denoted by *BL*, which considers only primitives for communications. By using the theory of *well-structured transition systems* [15], we show that for *BL* the termination is decidable. Because of the nature of *BL* we think a relation with the CCS [23] exists and we plan to investigate this line in future work.

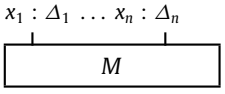
Then we add specific features of BlenX and show that the resulting languages are Turing equivalent. In particular, we prove that by adding either immediate actions to *BL* (we denote this subset with BL^{gp}) or *join* and *split* events (we denote this subset with BL^{e}) we obtain Turing equivalence. We show this by providing encodings of Random Access Machines (RAMs), a well-known Turing equivalent formalism, into BL^{gp} and BL^{e}. All the encodings are shown to be correct.

Notice that there is a conceptual similarity between the notion of global priorities in a nondeterministic semantics and the notion of infinite reaction rates in a stochastic semantics. For this reason we think that the results here provided will help us in our future studies on the expressive power of the full BlenX with stochastic semantics.

The paper is organized as follows. In Section 2 a brief introduction describing the syntax and the main features of BlenX is presented. In Section 3 the syntax and semantics of *BL* is presented. In Section 4 termination of the *BL* subset is proved to be decidable. In Section 5 encodings of RAMs into the BL^{gp} and BL^{e} subsets are given, along with their proofs of correctness and in Section 6 some conclusions are reported.

2. The BlenX language

In this section we briefly survey the BlenX language. For a more exhaustive description of the conceptual framework and the main concepts we refer the reader to [31,32,12,36]. A biological entity M is represented as a box B_M, depicted below:



The pairs $x_i : \Delta_i$ represent the sites through which B_M may interact with other boxes, i.e. the motifs of the molecule M. *Types* Δ_i express the interaction capabilities at x_i. The dynamic behaviour of B_M is specified through the internal process M. A process is a CCS-like process for representing biomolecular interactions, extended for manipulating the interaction sites of a box. The parallel composition of different boxes, called *bio-process*, abstracts a biological system composed by parallel interacting biological entities. For instance, consider the following biochemical reaction:

$$\mathsf{A} + \mathsf{B} \rightharpoonup^{k_1} \mathsf{AB} \rightharpoonup^{k_2} \mathsf{CD} \rightharpoonup^{k_3} \mathsf{C} + \mathsf{D}.$$

Two molecules A and B bind to form the complex AB with a stochastic rate k_1. A biochemical interaction within AB leads to complex CD (with rate k_2) and finally C and D are released at a rate k_3. With BlenX such a reaction can be modeled in different ways, one of which is sketched in Fig. 1. Boxes B_A and B_B for molecules A and B complex into box B_{AB}, if the types Δ_A and Δ_B are *compatible* up to a certain *user-defined* algorithm (see [12]). Then, the internal process $A \mid B$ evolves into $C \mid D$, and types Δ_A and Δ_B are modified into Δ_C and Δ_D, respectively. Finally the complex unbinds releasing B_C and B_D.

A BlenX program, called also *system*, is a tuple $Z = \langle B, E, \xi \rangle$ made up of a *bio-process* B, a list of *events* E and *ambient* ξ. The bio-process B intuitively represents the structure of the system, that is a set of entities (i.e. boxes) interacting in the same environment, E represents the list of possible events enabled in the system and the ambient ξ contains information about the environment. A notion of structural congruence is introduced to equate different implementations of the same biological systems [13, Def. 7]. Intuitively, two systems $Z = \langle B, E, \xi \rangle$ and $Z' = \langle B', E', \xi' \rangle$ are structurally congruent, denoted with $Z \equiv Z'$, if their bio-processes B and B' and their list of events E and E' are identical up to structure and their ambients are equal.

The dynamics of a system is formally specified through the *operational semantics* [30]. Given a system, its dynamic is described by three types of actions: **(monomolecular)** describe the evolution of single boxes. More precisely, an *intra-box communication* allows components to interact within the same box, the *expose* action adds a new site of interaction to the interface of the box containing the expose, the *change* action modifies the type of an interaction site, *hide* and *unhide*

Table 1
Definition of free names for bio-processes and processes.

$fn(\mathsf{Nil}) = fn(\mathsf{nil}) = fn(\beta(x, \Delta)) = fn(\beta(x, \Delta)I) = \emptyset$
$fn(I[P]) = fn(P) \setminus sub(I)$
$fn(B \parallel B') = fn(B) \cup fn(B')$
$fn(P\|P') = fn(P) \cup fn(P')$
$fn(!\pi.P) = fn(\pi.P)$
$fn(M + M') = fn(M) \cup fn(M')$
$fn(x(y).P) = \{x\} \cup (fn(P) \setminus \{y\})$
$fn(\bar{x}\langle y\rangle.P) = \{x, y\} \cup fn(P)$

actions make respectively invisible and visible an interaction site. Finally, the *die* action eliminates the box that performs the action and, recursively, all the boxes directly or indirectly complexed with them; **(bimolecular)** describe interactions that involves two boxes. More precisely, *inter-communication* enables interaction between boxes, *complex* and *decomplex* creates and destroys dedicated communication binding between boxes; **(events)** are the composition of a condition and an action and are triggered only when the condition associated with the event is satisfied. Events can be considered as global rules of the system which can substitute, create and delete boxes from the system. In particular, the list E can contain five types of events: *join*, which substitutes two boxes with single ones; *split*, which substitutes a box with two boxes; *new*, which introduces a specified number of instances of a box; *delete*, which eliminates boxes.

3. The *BL* subset

Let $\mathcal{N}$ be a countably infinite set of names (ranged over by lower-case letters) and let T be a countably infinite set of types (ranged over by $\Delta, \Gamma, \Delta', \Delta_0, \ldots$) such that $T \cap \mathcal{N} = \emptyset$. The syntax of *BL* is defined in the following way:

$$
\begin{aligned}
B &::= \mathsf{Nil} \mid I[P] \mid B \parallel B \\
I &::= \beta(x, \Delta) \mid \beta(x, \Delta)I \\
P &::= P|P \mid !\pi.P \mid M \\
M &::= \mathsf{nil} \mid \pi.P \mid M + M \\
\pi &::= x(y) \mid \bar{x}\langle y\rangle
\end{aligned}
$$

Bio-processes generated by the non-terminal symbol B can be either a *box* (the first two productions) or a parallel composition of boxes, i.e. boxes running concurrently. The special process Nil does nothing; i.e. it is the deadlocked box. The box $I[P]$ is a process (see below) prefixed by a specialized *interface* I that represents the interaction capabilities of the box. A program written in *BL*, called also *system*, is a *bio-process* B. We denote with $box_{\#}(B)$ the function returning the number of boxes composing the bio-process B.

An interface I is made up of a non-empty string of *binders* of the form $\beta(x, \Delta)$, where the name x is the *subject* of the binder and Δ represents the type of x. The subject x of a binder is a binding occurrence that binds all the free occurrences of x in the box to which the binder belongs. We let interfaces be ranged over by $I, I_1, I_2, \ldots, I', \ldots$. We write $I = I_1 I_2$ to mean that I is the interface given by the juxtaposition of I_1 and I_2. Also, the metavariables $I^*, I_1^*, I_2^*, \ldots$ stay for either an interface or the empty string. The above notation for the juxtaposition is extended to these metavariables in the natural way.

With $\mathcal{B}$ and $\mathcal{I}$ we denote the set of all the possible bio-processes and interfaces, respectively.

Definition 3.1. The functions $sub : \mathcal{I} \to 2^{\mathcal{N}}$ and $sub_t : \mathcal{B} \to 2^{\mathcal{N}}$ are defined as follows

$$
\begin{aligned}
sub(\beta(x, \Gamma)) &= \{x\} & \qquad sub_t(I[P]) &= sub(I) \\
sub(\beta(x, \Gamma)I) &= \{x\} \cup sub(I) & \qquad sub_t(B \parallel B') &= sub_t(B) \cup sub_t(B')
\end{aligned}
$$

Function sub returns the set of subjects present in an interface, while function sub_t returns the total set of subjects present in all the boxes interfaces composing a bio-process. A well-formed interface I is a non-empty string of binders where subjects and types are all distinct.

Definition 3.2. Let $B = I_1[P_1] \parallel \cdots \parallel I_n[P_n]$ be a bio-process. We say that B is well formed if $\forall i \in \{1, \ldots, n\}$ the interface I_i is well formed.

We denote with $\mathcal{Z}$ the set of all well-formed systems. In particular, for the *BL* subset we have that $\mathcal{Z} \subset \mathcal{B}$.

Processes generated by the non-terminal symbol P are referred as *processes* and the set of all possible processes is denoted by $\mathcal{P}$. The nil process does nothing; it is a deadlocked process. The binary operator | composes two processes that can run concurrently. The bang operator ! is used to replicate copies of the process passed as argument. Note that we use only guarded replication, i.e. the process argument of the ! must have a prefix π that forbids any other action of the process until it has been consumed. The last non-terminal symbol M of the productions of P is used to introduce guarded choices. In fact M generates summations of guarded prefixes of the form $\pi.P$.

Definition of free names for bio-processes and processes is given in Table 1 through the function fn.

The dynamics of a system is formally specified through the *operational semantics* in Table 2 which uses a notion of structural congruence $\equiv$.

Table 2
Reduction semantics of *BL*.

(intra)	$I[\overline{x}\langle z\rangle.P_1 + M_1 \mid x(w).P_2 + M_2 \mid P_3] \rightarrow I[P_1 \mid P_2\{^z/_w\} \mid P_3]$
(inter)	$\dfrac{P_1 \equiv_p \overline{x}\langle z\rangle.R_1 + M_1 \mid Q_1 \qquad P_2 \equiv_p y(w).R_2 + M_2 \mid Q_2}{I_1[P_1] \parallel I_2[P_2] \rightarrow I_1[R_1 \mid Q_1] \parallel I_2[R_2\{^z/_w\} \mid Q_2]}$ where $I_1 = \beta(x, \Delta)I_1^*$ and $I_2 = \beta(y, \Gamma)I_2^*$ and provided that $\alpha(\Gamma, \Delta) > 0$ and $z \notin sub(I_1) \cup sub(I_2)$
(struct)	$\dfrac{B_1 \equiv_b B_1' \qquad B_1' \rightarrow B_2' \qquad B_2' \equiv_b B_2}{B_1 \rightarrow B_2}$
(redex)	$\dfrac{B \rightarrow B'}{B \parallel B_1 \rightarrow B' \parallel B_1}$

Group a

a.1) $P_1 \equiv_p P_2$ if P_1 and P_2 α-equivalent
a.2) $P_1 \mid (P_2 \mid P_3) \equiv_p (P_1 \mid P_2) \mid P_3$
a.3) $P_1 \mid P_2 \equiv_p P_2 \mid P_1$
a.4) $P \mid \mathsf{nil} \equiv_p P$
a.5) $M_1 + (M_2 + M_3) \equiv_p (M_1 + M_2) + M_3$
a.6) $M_1 + M_2 \equiv_p M_2 + M_1$
a.7) $M + \mathsf{nil} \equiv_p M$
a.8) $!\pi.P \equiv_p \pi.(P \mid !\pi.P)$

Group b

b.1) $I[P_1] \equiv_b I[P_2]$ if $P_1 \equiv_p P_2$
b.2) $B_1 \parallel (B_2 \parallel B_3) \equiv_b (B_1 \parallel B_2) \parallel B_3$
b.3) $B_1 \parallel B_2 \equiv_b B_2 \parallel B_1$
b.4) $B \parallel \mathsf{Nil} \equiv_b B$
b.5) $I_1I_2[P_1] \equiv_b I_2I_1[P_2]$ provided $P_1 \equiv_p P_2$
b.6) $B \equiv_b B'$ if
$B = I^*\beta(x,\Gamma)[P]$ and $B' = I^*\beta(y,\Gamma)[P\{y/x\}]$ or
$B' = I^*\beta(x,\Gamma)[P]$ and $B = I^*\beta(y,\Gamma)[P\{y/x\}]$
with $y \notin fn(P) \cup sub(I^*)$

Fig. 2. Structural laws for *BL*.

Definition 3.3. Structural congruence over processes, denoted $\equiv_p$, is the smallest relation which satisfies the laws in Fig. 2 (group a) and structural congruence over bio-processes, denoted $\equiv_b$, is the smallest relation which satisfies the laws in Fig. 2 (group b).

For the *BL* subset the relation $\equiv$ over systems coincides with the relation $\equiv_b$.

The actions that a process can perform are described by the syntactic category π. These actions are common to most process calculi. They represent respectively the input/reception of something that will instantiate the placeholder y over a channel named x ($x(y)$) and the output/send of a value y over a channel named x ($\overline{x}\langle y\rangle$). The placeholder y in the input is a binding occurrence that binds all the free occurrences of y in the scope of the prefix $x(y)$. Sometimes the channel name x is called subject and the placeholder/value y is called object of the prefix.

Parallel processes that perform complementary actions on the same channel inside the same box (a process performs an input $x(z)$ and the other one an output $\overline{x}\langle y\rangle$) can synchronize and exchange a message, performing an *intra-communication*. The value y flows from the process performing the output to the one performing the input. The flow of information affects the future behaviour of the system because all the free occurrences of z bound by the input placeholder are replaced in the receiving process by the actual value sent y.

Processes in different boxes can perform an *inter-communication* if one sends out of the box a value y over a link x that is bound to a binder $\beta(x, \Delta)$ of the box and a process in another box is willing to receive a value from a *compatible* binder $\beta(y, \Gamma)$ through the action $y(z)$. The two corresponding binders are compatible if a *compatibility* function α applied to the types returns a value greater than zero. Note that intra-communications occur on perfectly symmetric input/output pairs that share the same subject, while inter-communications can occur between primitives that have different subjects, provided that their types are compatible. In other words, we relax the perfect key–lock mechanism of classical process calculi on inter-communications.

In [37] we showed that the structural congruence relation over systems is decidable.

Definition 3.4. The *BL* Transition System (TS) is referred as $(\mathcal{Z}, \rightarrow)$, where $\mathcal{Z}$ is the set of well-formed systems and $\rightarrow \subseteq \mathcal{Z} \times \mathcal{Z}$ is the transition relation.

Throughout the paper, for simplicity, we denote with $\prod_{i=1}^{n} B_i$ a parallel composition of bio-processes $B_1 \parallel \cdots \parallel B_n$, with $\prod_{i=1}^{n} P_i$ a parallel composition of processes $P_1 \parallel \cdots \parallel P_n$ and with $\sum_{i=1}^{n} \pi_i.P_i$ a choice process of the form $\pi_1.P_1 + \cdots + \pi_n.P_n$.

4. A decidability result for the *BL* subset

In this section we show that termination is decidable for the *BL* subset. With respect to the methods used in this section, we take inspiration from [3–5] in which decidability results for *π-calculus*, *Pure Mobile Ambients* and *Brane Calculi* have been presented and we rely on the theory of well-structured transition systems [15]. In particular, the existence of an infinite computation starting from a given state is decidable for finitely branching transition systems, provided that the set of states can be equipped with a *well-quasi-ordering*. The main differences with the results contained in [4,5] are that in our language we have no static hierarchies of ambients and nested restrictions, but we have a two level hierarchy of boxes and processes and a form of name passing over finite sets of names.

The decidability of termination for *BL* is proved by first providing an alternative labelled transition semantics for a subset $\mathcal{Z}_s$ of *BL* bio-processes we call *safe* and then by showing that there is a correspondence of this semantics with the reduction semantics presented in Section 2. In particular, we show that we have always possible and easy to transform a generic *BL* bio-process into an equivalent *safe* one and that a bio-process admits an infinite computation according to the reduction semantics if and only if one of its corresponding safe bio-processes admits an infinite sequence of τ transitions according to the new labelled transition semantics.

Then, we define a quasi-ordering $\preceq_b$ on bio-processes which is strongly compatible with $\xrightarrow{\tau}$, we show that the relation $\preceq_b$ is a well-quasi-ordering and finally we prove that the termination of bio-processes in $\mathcal{Z}_s$ is decidable.

4.1. Well-structured transition systems

In this section we recall some basic definitions and results from [15,18]. A quasi-ordering (qo) is a reflexive and transitive relation.

Definition 4.1. A well-quasi-ordering (wqo) on a set X is a qo $\leq$ such that any infinite sequence of elements $x_0, x_1, x_2, \ldots$ from X contains an increasing pair $x_i \leq x_j$ with $i < j$. The set X is said to be well-quasi-ordered, or wqo for short.

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 \leq i_1 \leq i_2 \leq \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences.

Definition 4.2. A transition system is a tuple $TS = (S, \rightarrow)$ where S is a set of states and $\rightarrow \subseteq S \times S$ is a set of transitions. If $p, q \in S$, then $(p, q) \in \rightarrow$ is usually written as $p \rightarrow q$.

The set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of a state $s \in S$ is denoted with $Succ(s)$. TS is finite branching if $Succ(s)$ is finite for all $s \in S$.

Definition 4.3. A well-structured transition system with strong compatibility, denoted with $TS = (S, \rightarrow, \leq)$ is a transition system equipped with a qo $\leq$ on S such that the following conditions hold:

- $\leq$ is a well-quasi-ordering
- $\leq$ is (upward) compatible with $\rightarrow$, i.e., for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$ (strong compatibility).

Our decidability result is based on the following theorem [15]:

Theorem 4.4. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with decidable $\leq$ and computable Succ. The existence of an infinite computation starting from a state $s \in S$ is decidable.*

In order to prove that the qo we will define on bio-processes is a wqo, we need to introduce some important results proved by Higman in [18]. First of all we recall that given a set S, the set S^* denotes the set of finite sequences of elements in S.

Definition 4.5. Let S be a set and $\leq$ a wqo over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 \ldots t_m$ and $u = u_1 \ldots u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \ldots, m$.

Theorem 4.6 (*Higman*). *Let S be a set and $\leq$ be a wqo over S. Then, the relation $\leq_*$ is a wqo over S^*.*

Lemma 4.7. *Let S be a finite set. Then equality is a wqo over S.*

Table 3
Terminology and notation for action labels.

θ	Kind	$\mathsf{fn}(\theta)$	$\mathsf{bn}(\theta)$	$\mathsf{n}(\theta)$
xy	Process input	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$\bar{x}y$	Process output	$\{x, y\}$	$\emptyset$	$\{x, y\}$
Δxy	Box input over Δ	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$\Delta\bar{x}y$	Box output over Δ	$\{x, y\}$	$\emptyset$	$\{x, y\}$
τ	Internal	$\emptyset$	$\emptyset$	$\emptyset$

4.2. A labelled transition semantics for BL

In this section we define a labelled transition semantics for *BL* (Table 4) to get rid of structural congruence. Axioms and rules for processes are in the style of the transition semantics reported in [38] (page 38) for the π-calculus, and hence some results there reported can be reused.

We use the metavariable θ to range over xy, $\bar{x}y$, Δxy, $\Delta\bar{x}y$, and τ. The set of names, $\mathsf{n}(\theta)$, of θ is $\mathsf{fn}(\theta) \cup \mathsf{bn}(\theta)$. In Table 3 terminology and notation for labels are reported.

Notice that the semantics we define is not equivalent to the one presented in Section 2, because of the absence of rules for managing α-conversion. We do not explicitly consider α-convertible bio-processes to get rid of the infinite names over *intra-communication* that α-conversion introduces. This fact will be used to obtain the wqo over bio-processes.

However, we will show that there is a correspondence between the labelled transition semantics over the *safe* subset of *BL* bio-processes and the reduction semantics over *BL* bio-processes. Moreover, although the labelled transition semantics is not finite branching, we will show that the transition systems constructed over safe bio-processes by only considering $\xrightarrow{\tau}$ transitions are finite branching. This fact is essential to use the theory of well-structured transition systems.

Safe bio-processes are introduced to guarantee that no behaviours are lost when we get rid of structural congruence. Suppose $\alpha(\Delta, \Delta) > 0$ and consider the following bio-process:

$$B = \beta(x,\ \Delta)[\bar{x}\langle y\rangle.\mathsf{nil}] \parallel \beta(x,\ \Delta)\,\beta(y,\ \Gamma)[x(z).\bar{z}\langle k\rangle.\mathsf{nil} \mid y(z).\mathsf{nil}]$$

To avoid captures in inter-communications, rule (inter) in Table 2 requires $y \notin sub(\beta(x,\ \Delta)) \cup sub(\beta(x,\ \Delta)\,\beta(y,\ \Gamma))$, which in this case does not hold. Hence, in order to consume the inter-communication we have to consider (one among infinitely many others) the bio-process:

$$B' = \beta(x,\ \Delta)[\bar{x}\langle y\rangle.\mathsf{nil}] \parallel \beta(x,\ \Delta)\,\beta(n,\ \Gamma)[x(z).\bar{z}\langle k\rangle.\mathsf{nil} \mid n(z).\mathsf{nil}]$$

and derive the transition through the rule struct, i.e., structural law (b.6) implies $B \equiv_b B'$. Safe bio-processes guarantee that we never need to apply the structural law (b.6) in order to derive an inter-communication, simplifying the definition of our labelled transition semantics.

Definition 4.8. The bio-process B is safe iff $fn(B) \cap sub_t(B) = \emptyset$.

We denote with $\mathcal{Z}_s \subset \mathcal{Z}$ the set of safe well-formed bio-processes.

Lemma 4.9. *Let $B \in \mathcal{Z}$. There exists $B' \in \mathcal{Z}_s$ such that $B \equiv_b B'$.*

Proof. Immediate from the structural congruence rules reported in Fig. 2 (Group b). □

We denote with $safe(B) \subset \mathcal{Z}_s$ the set of safe bio-processes structurally congruent to a generic bio-process $B \in \mathcal{Z}$. Given a bio-process $B \in \mathcal{Z}$, it is easy to see that the problem of finding an equivalent safe bio-process B' is decidable an efficiently solvable. Indeed, considering the finite set $fn(B)$ of free names in B and the number m of binders of B (both the set and the value can be computed linearly in the size of B), a safe bio-process B' structurally congruent to B can be obtained simply by substituting all the binder subjects of B with names contained in a set $M \subset \mathcal{N}$ such that $M \cap fn(B) = \emptyset$ and $|M| = m$. The problem of finding this set M is effectively computable.

Now, we show that the transition system $(\mathcal{Z}_s, \xrightarrow{\tau})$ is finite branching. In order to do this we first have to show that the safe property of bio-processes is preserved over τ transitions. However, some preliminary results are needed. The first result describes how, given a transition $P \xrightarrow{\theta} P'$ over processes or a transition $B \xrightarrow{\theta} B'$ over bio-processes, the free names of P' and B' are the finite set made up of the free names of P and B and the names in θ.

Lemma 4.10. *Let $P, P' \in \mathcal{P}$. Suppose $P \xrightarrow{\theta} P'$, then*

(a) *if $\theta = \bar{x}y$ then $x, y \in fn(P)$ and $fn(P') \subseteq fn(P)$*
(b) *if $\theta = xy$ then $x \in fn(P)$ and $fn(P') \subseteq fn(P) \cup \{y\}$*
(c) *if $\theta = \tau$ then $fn(P') \subseteq fn(P)$.*

Proof. See [38], page 44. □

Table 4
Labelled transition semantics of *BL* (we omit the symmetric rules r_pi_sum, r_pi_par, r_intra, r_bio_par and r_inter).

(pi_in)	$x(w).P \xrightarrow{xy} P\{y/w\}$	(bio_out)	$\dfrac{P \xrightarrow{\bar{x}y} P'}{\beta(x,\Delta)I^*[P] \xrightarrow{\Delta\bar{x}y} \beta(x,\Delta)I^*[P']}$ with $y \notin sub(I^*) \cup \{x\}$
(pi_out)	$\bar{x}\langle y\rangle.P \xrightarrow{\bar{x}y} P$		
(rep_in)	$!x(w).P \xrightarrow{xy} P\{y/w\} \mid !x(w).P$	(bio_in)	$\dfrac{P \xrightarrow{xy} P'}{\beta(x,\Delta)I^*[P] \xrightarrow{\Delta xy} \beta(x,\Delta)I^*[P']}$ with $y \notin sub(I^*) \cup \{x\}$
(rep_out)	$!\bar{x}\langle y\rangle.P \xrightarrow{\bar{x}y} P \mid !\bar{x}\langle y\rangle.P$		
(l_pi_sum)	$\dfrac{M_0 \xrightarrow{\theta} M_0'}{M_0 + M_1 \xrightarrow{\theta} M_0'}$	(pi_bio)	$\dfrac{P \xrightarrow{\tau} P'}{I[P] \xrightarrow{\tau} I[P']}$
(l_pi_par)	$\dfrac{P_0 \xrightarrow{\theta} P_0'}{P_0 \mid P_1 \xrightarrow{\theta} P_0' \mid P_1}$	(l_bio_par)	$\dfrac{B_0 \xrightarrow{\theta} B_0'}{B_0 \parallel B_1 \xrightarrow{\theta} B_0' \parallel B_1}$
(l_intra)	$\dfrac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	(l_inter)	$\dfrac{B_0 \xrightarrow{\Delta\bar{x}z} B_0' \quad B_1 \xrightarrow{\Gamma yz} B_1'}{B_0 \parallel B_1 \xrightarrow{\tau} B_0' \parallel B_1'}$ with $\alpha(\Delta,\Gamma) > 0$

Lemma 4.11. *Let $B, B' \in \mathcal{Z}_s$. Suppose $B \xrightarrow{\theta} B'$. Then*

(a) *if $\theta = \Delta\bar{x}y$ then $y \in fn(B)$ and $fn(B') \subseteq fn(B)$*
(b) *if $\theta = \Delta xy$ then $fn(B') \subseteq fn(B) \cup \{y\}$*
(c) *if $\theta = \tau$ then $fn(B') \subseteq fn(B)$.*

Proof. By induction on the inference of $B \xrightarrow{\theta} B'$. We present only the most relevant cases. The other cases can be proved similarly.
(a)
(Case bio_out) We have $B = (x,\Delta)I^*[P]$ and $B' = (x,\Delta)I^*[P']$. By definition we have $fn(B) = fn(P) \setminus (sub(I^*) \cup \{x\})$ and $fn(B') = fn(P') \setminus (sub(I^*) \cup \{x\})$. By hypothesis we have $P \xrightarrow{\bar{x}y} P'$ and hence, by Lemma 4.10, $x, y \in fn(P)$ and $fn(P') \subseteq fn(P)$. Therefore $fn(P') \setminus (sub(I^*) \cup \{x\}) \subseteq fn(P) \setminus (sub(I^*) \cup \{x\})$, which means $fn(B') \subseteq fn(B)$. Moreover, since $y \in fn(P)$ and $y \notin (sub(I^*) \cup \{x\})$ we have $y \in fn(B)$.
(Case l_bio_par) By definition $fn(B_0 \parallel B_1) = fn(B_0) \cup fn(B_1)$. By inductive hypothesis we have $y \in fn(B_0)$ and $fn(B_0') \subseteq fn(B_0)$. Therefore $fn(B_0') \cup fn(B_1) \subseteq fn(B_0) \cup fn(B_1)$ which means $fn(B') \subseteq fn(B)$. Moreover, since $y \in fn(B_0)$, we have $y \in fn(B)$.
(b)
(Case bio_in) We have $B = (x,\Delta)I^*[P]$ and $B' = (x,\Delta)I^*[P']$. By definition we have $fn(B) = fn(P) \setminus (sub(I^*) \cup \{x\})$ and $fn(B') = fn(P') \setminus (sub(I^*) \cup \{x\})$. By hypothesis we have $P \xrightarrow{xy} P'$ and hence, by Lemma 4.10, $x \in fn(P)$ and $fn(P') \subseteq (fn(P) \cup \{y\})$. Therefore $(fn(P') \setminus (sub(I^*) \cup \{x\})) \cup \{y\} \subseteq (fn(P) \setminus (sub(I^*) \cup \{x\})) \cup \{y\}$, which means $fn(B') \subseteq fn(B) \cup \{y\}$.
(c)
(Case l_iter) By definition $fn(B_0 \parallel B_1) = fn(B_0) \cup fn(B_1)$. By inductive hypothesis we have $z \in fn(B_0)$, $fn(B_0') \subseteq fn(B_0)$, and $fn(B_1') \subseteq fn(B_1) \cup \{z\}$. Since $z \in fn(B_0)$ we have $fn(B_0) \cup fn(B_1) \cup \{z\} = fn(B_0) \cup fn(B_1)$ and therefore $fn(B_0') \cup fn(B_1') \subseteq fn(B_0) \cup fn(B_1)$, which means $fn(B') \subseteq fn(B)$. □

Lemma 4.12. *Let $B \in \mathcal{Z}_s$. Then $B \xrightarrow{\tau} B'$ implies $B' \in \mathcal{Z}_s$.*

Proof. By hypothesis we have $fn(B) \cap sub_t(B) = \emptyset$. By the semantics definition it is $sub_t(B) = sub_t(B')$, because no rule can change the subjects of binders. Moreover, by Lemma 4.11, we have $fn(B') \subseteq fn(B)$. As a consequence $fn(B') \cap sub_t(B') = \emptyset$ and hence the lemma holds. □

Now, we recall some results on image finiteness of π-calculus processes, reported in [38] (page 45) that are still valid for our processes.

Lemma 4.13. *Let $P \in \mathcal{P}$. Then*
(1) *There are only finitely many x such that $P \xrightarrow{xz} P'$ for some z and P'.*
(2) *There are only finitely many triples x, y, P' such that $P \xrightarrow{\bar{x}y} P'$.*

These results can be used to show that for any process P there are only finitely many processes Q such that $P \xrightarrow{\tau} Q$ and that for any safe bio-process B there are only finitely many safe bio-processes B' such that $B \xrightarrow{\tau} B'$.

Lemma 4.14. *Let $P \in \mathcal{P}$. Then the set $Succ(P) = \{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\}$ is finite.*

Proof. By induction on the structure P.
(Induction base) If P has the form nil, $x(y).P'$, $\overline{x}\langle y\rangle.P'$, $!x(y).P'$ or $!\overline{x}\langle y\rangle.P'$ it is simple to see that no τ actions can be derived using the semantics rules. Hence, in this cases the set is $\{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\} = \emptyset$.
(Case $P = M_0 + M_1$) By inductive hypothesis the sets $Succ(M_0)$ and $Succ(M_1)$ are finite. Since no intra-communications between processes M_0 and M_1 can be performed, then the set $\{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\} = Succ(M_0) \cup Succ(M_1)$ is finite.
(Case $P = P_0 \mid P_1$) By inductive hypothesis the sets $Succ(P_0)$ and $Succ(P_1)$ are finite. However, P_0 and P_1 are parallel processes and hence they can synchronize on inputs and outputs actions and perform intra-communications, generating τ transitions. By Lemma 4.13, we obtain that the number of possible input and output in P_0 and P_1 is finite and hence only a finite number of τ actions (using l_intra and r_intra rules) can be derived. Therefore, by inductive hypothesis and Lemma 4.13 the set $Succ(P)$ is finite. □

We extend results of Lemma 4.13 and Lemma 4.14 to bio-processes.

Lemma 4.15. *Let $B \in \mathcal{B}$. Then*
(1) *There are only finitely many pairs x, Δ such that $B \xrightarrow{\Delta xz} B'$ for some z and B'.*
(2) *There are only finitely many tuples Δ, x, y, B' such that $B \xrightarrow{\Delta \bar{x}y} B'$.*

Proof. By induction on the structure of B.
(1)
(Case Nil) No couple x, Δ such that $B \xrightarrow{\Delta xz} B'$ for some z and B' exists.
(Case $I[P]$) For each subject $x \in sub(I)$, we have (by Lemma 4.13) that there are only finitely many x such that $P \xrightarrow{xz} P'$ for some z and P' and hence only finitely many x such that $P \xrightarrow{xz} P'$ for some $z \notin sub(I)$ and P'. Since by definition I is well-formed and the set of binders subjects $sub(I)$ is finite, we obtain (by the application of rule bio_in) that there are only finitely many pairs Δ, x such that $I[P] \xrightarrow{\Delta xz} I[P']$ for some z and $I[P']$.
(Case $B \parallel B'$) By inductive hypothesis on B and B' the lemma follows immediately.
(2)
(Case Nil) No tuple Δ, x, y, B' such that $B \xrightarrow{\Delta \bar{x}y} B'$ exists.
(Case $I[P]$) For each subject $x \in sub(I)$, we have (by Lemma 4.13) that there are only finitely many tuples x, y, P' such that $P \xrightarrow{\bar{x}y} P'$ and hence there are only finitely many tuples x, y, P' such that $P \xrightarrow{\bar{x}y} P'$ and $y \notin sub(I)$. Since by definition I is well-formed and the set of binder subjects $sub(I)$ is finite, we obtain (by the application of rule bio_out) that there are only finitely many tuples Δ, x, y, $I[P']$ such that $I[P] \xrightarrow{\Delta \bar{x}y} I[P']$.
(Case $B \parallel B'$) By inductive hypothesis on B and B' the lemma follows immediately. □

Lemma 4.16. *Let $B \in \mathcal{Z}_s$. Then the set $Succ(B) = \{B' \in \mathcal{Z}_s \mid B \xrightarrow{\tau} B'\}$ is finite.*

Proof. By induction on the structure of B.
(Case Nil) In this case $Succ(B)$ is obviously $\emptyset$.
(Case $B[P]$) By Lemma 4.14 the set $Succ(P)$ is finite. It immediately follows (by application of rule pi_bio) that the set $Succ(B[P])$ is finite.
(Case $B_0 \parallel B_1$) By inductive hypothesis the sets $Succ(B_0)$ and $Succ(B_1)$ are finite. However, B_0 and B_1 are parallel bio-processes and hence they can synchronize on inputs and outputs actions over compatible binders and perform inter-communications, generating τ transitions. By Lemma 4.15, the number of possible input and output over compatible binders between B_0 and B_1 is finite and hence only a finite number of τ actions (using l_inter and r_inter rules) can be derived. Therefore, by inductive hypothesis and Lemma 4.13, the set $Succ(B)$ is finite. □

Corollary 4.17. *The transition system $(\mathcal{Z}_s, \xrightarrow{\tau})$ is finite branching.*

Proof. Immediate by Lemma 4.16. □

To reason on the new labelled semantics we need to show its correspondence with the reduction semantics. In particular, we show that the τ transition relation over safe bio-processes and the reduction relation over bio-processes agree.

Lemma 4.18. *Let $B, B' \in \mathcal{Z}_s$. If $B \equiv_b B'$ and $B \xrightarrow{\tau} B''$, then for some $B''' \in \mathcal{Z}_s$ we have that $B' \xrightarrow{\tau} B'''$ and $B'' \equiv_b B'''$.*

Proof. By induction on the number of structural congruence rules applied to B for obtaining B'. □

Theorem 4.19. *Let $B \in \mathcal{Z}$ and $B'' \in safe(B)$. Then $B \to B'$ iff $B'' \xrightarrow{\tau} B'''$ and $B''' \in safe(B')$.*

Proof. ($\Rightarrow$)
If $B \to B'$ then there exist bio-processes $B_0 \equiv_b B$ and $B_1 \equiv_b B'$ such that

$$B_0 = I[\overline{x}\langle z\rangle.P_1 + M_1 \mid x(w).P_2 + M_2 \mid P_3] \parallel B_2 \quad \text{and } B_1 = I[P_1 \mid P_2\{z/w\} \mid P_3] \parallel B_2$$

or

$$B_0 = \beta(x, \Delta)\, I_1^*[\overline{x}\langle z\rangle.R_1 + M_1 \mid Q_1] \parallel \beta(y, \Gamma)\, I_2^*[y(w).R_2 + M_2 \mid Q_2] \parallel B_2 \quad \text{and } B_1 = I_1[R_1 \mid Q_1] \parallel I_2[R_2\{z/w\} \mid Q_2] \parallel B_2$$

with $\alpha(\Gamma, \Delta) > 0$ and $z \notin sub(I_1^*) \cup \{x\}$. By Lemma 4.9, in both cases there exist safe bio-processes $B_0^s \equiv_b B_0$ and $B_1^s \equiv_b B_1$. In particular, in the second case we have that B_0^s is such that $z \notin sub(I_1^*) \cup sub(I_2^*) \cup sub_t(B_2) \cup \{x, y\}$. It is easy to see that in both cases we can derive $B_0^s \xrightarrow{\tau} B_1^s$. Moreover, $B_0^s \equiv_b B_0 \equiv_b B$ means that $B_0^s \in safe(B)$ and hence $B'' \equiv_b B_0^s$. By Lemma 4.18, there exists $B''' \in \mathcal{Z}_s$ such that $B''' \equiv_b B_1^s$ and from $B''' \equiv_b B_1^s \equiv_b B_1 \equiv_b B'$ we obtain $B''' \in safe(B')$.
($\Leftarrow$)
It is enough to show that $B \xrightarrow{\tau} B'$ implies $B \to B'$. The proof is by induction on the inference of $B \xrightarrow{\tau} B'$.
(Case I_inter) By hypothesis we know that $B_0 \xrightarrow{\Delta\overline{x}z} B_0'$, $B_1 \xrightarrow{\Gamma xz} B_1'$. Notice that:

(1) if $B_0 \xrightarrow{\Delta\overline{x}z} B_0'$ then $B_0 \equiv_b \beta(x, \Delta)\, I_0^*[\overline{x}\langle z\rangle.Q + M \mid R] \parallel B_2, B_0' \equiv_b \beta(x, \Delta)\, I_0^*[Q \mid R] \parallel B_2$ and $z \notin sub(I_0^*) \cup \{x\}$.
(2) if $B_1 \xrightarrow{\Gamma yz} B_1'$ then $B_1 \equiv_b \beta(y, \Gamma)\, I_1^*[y(w).S + N \mid T] \parallel B_3, B_1' \equiv_b \beta(y, \Gamma)\, I_1^*[S\{z/w\} \mid T] \parallel B_3$ and $z \notin sub(I_1^*) \cup \{y\}$.

Hence, the bio-process $B_0 \parallel B_1$ is structurally congruent ($\equiv_b$) to

$$\beta(x, \Delta)\, I_0^*[\overline{x}\langle z\rangle.Q + M \mid R] \parallel \beta(y, \Gamma)\, I_1^*[y(w).S + N \mid T] \parallel B_2 \parallel B_3$$

and by applying inter and redex rules of the reduction semantics we can derive the transition

$$B_0 \parallel B_1 \to \beta(x, \Delta)\, I_0^*[Q \mid R] \parallel \beta(y, \Gamma)\, I_1^*[S\{z/w\} \mid T] \parallel B_2 \parallel B_3,$$

where the resulting bio-process is structurally congruent to $B_0' \parallel B_1'$. By applying the rule struct of the reduction semantics we are done.
(Case pi_bio) By hypothesis we have $P \xrightarrow{\tau} P'$. Notice that if $P \xrightarrow{\tau} P'$ than $P \equiv_p \overline{x}\langle z\rangle.Q + N \mid x(w).S + M \mid R$ and $P' \equiv_p Q \mid S\{z/w\} \mid R$. By applying the intra rule of the reduction semantics we can derive the transition $I[P] \to I[Q \mid S\{z/w\} \mid R]$, where the resulting bio-process is structurally congruent to $I[P']$. By applying the rule struct of the reduction semantics we end the proof.
(Case I_bio_par) By inductive hypothesis we have $B_0 \xrightarrow{\tau} B_0'$ implies $B_0 \to B_0'$ and hence, by applying the rule redex of the reduction semantics we derive $B_0 \parallel B_1 \to B_0' \parallel B_1$. □

As consequence of Theorem 4.19, a bio-process $B \in \mathcal{Z}$ admits an infinite computation according to the reduction semantics if and only if a corresponding bio-process $B' \in safe(B)$ admits an infinite sequence of τ transitions according to the labelled transition semantics. In particular, B' is terminating according to the labelled transition semantics if an infinite sequence of τ transitions starting from B' does not exists.

Corollary 4.20. *Let $B \in \mathcal{Z}$ and $B' \in safe(B)$. The bio-process B terminates according to the reduction semantics iff B' terminates according to the labelled transition semantics.*

In the reminder of the paper we consider only safe bio-processes in $\mathcal{Z}_s$.

4.3. Decidability of termination for $(\mathcal{Z}_s, \xrightarrow{\tau})$

Here we show that the existence of an infinite computation over safe bio-processes is computable in *BL*. We equip the labelled transition system $(BL, \xrightarrow{\tau})$ with a qo $\preceq_b$ on bio-processes which turns out to be wqo compatible with $\xrightarrow{\tau}$. Then we show that termination is decidable.

Definition 4.21. Let $B \in \mathcal{Z}_s$. The set of bio-processes reachable from B with a sequence of τ transitions is:

$$Deriv(B) = \{B' \mid B \xrightarrow{\tau}{}^{*} B'\}.$$

In order to define the qo $\preceq_b$, we first introduce a simplified structural congruence relation which is compatible with $\xrightarrow{\theta}$. This relation captures only reordering of sums, the monoidal laws for the parallel composition of processes and bio-processes and reordering of binders.

Definition 4.22. The $\equiv_p^{dec}$ congruence relation over processes, is the smallest relation which satisfies the laws a.2, a.3, a.4, a.5, a.6, a.7 in Fig. 2 and the $\equiv_b^{dec}$ congruence relation over bio-processes, is the smallest relation which satisfies the laws b.1, b.2, b.3, b.4, b.5 in Fig. 2, where in b.1 and b.5 the entries $P_1 \equiv_p P_2$ are substituted with $P_1 \equiv_p^{dec} P_2$.

Since $\equiv_p^{dec} \subset \equiv_p$ and $\equiv_b^{dec} \subset \equiv_b$ all the previous results on safe bio-processes hold also for the simplified structural congruence.

Lemma 4.23. *Let $B_0, B_1 \in \mathcal{Z}_s$. If $B_0 \equiv_b^{dec} B_1$ and $B_1 \xrightarrow{\theta} B_1'$ then there exists B_0' such that $B_0 \xrightarrow{\theta} B_0'$ and $B_0' \equiv_b^{dec} B_1'$.*

Proof. By induction on the number of structural congruence rules applied to B for obtaining B'. □

We can now introduce a quasi-order $\preceq_b$ which will be proven to be a well-quasi-order.

Definition 4.24. Let $B, B' \in \mathcal{Z}_s$. We define $B \preceq_b B'$ if and only if there exist $I_1, \ldots, I_n, P_1, \ldots, P_n, Q_1, \ldots, Q_n, P_1', \ldots, P_n'$ such that $B \equiv_b^{dec} \prod_{i=1}^n I_i[P_i]$, $B' \equiv_b^{dec} \prod_{i=1}^n I_i[Q_i]$ and $Q_i = P_i \mid P_i'$ for $i = 1, \ldots, n$.

The $\preceq_b$ relation is reflexive, transitive and strongly compatible with $\xrightarrow{\theta}$.

Theorem 4.25. *Let $B_0, B_0', B_1 \in \mathcal{Z}_s$. If $B_0 \xrightarrow{\theta} B_0'$ and $B_0 \preceq_b B_1$ then there exists B_1' such that $B_1 \xrightarrow{\theta} B_1'$ and $B_0' \preceq_b B_1'$.*

Proof. By cases on the inference of $\xrightarrow{\theta}$. We present only one case because all the others are similar.
(Case bio_out) We have that $B_0 = \beta(x, \Delta)I^*[P] \xrightarrow{\Delta\bar{x}y} \beta(x, \Delta)I^*[P'] = B_0'$. By definition of $\preceq_b$ we have that there exists a process Q such that $B_1 \equiv_b^{dec} \beta(x, \Delta)I^*[P|Q]$. By hypothesis we know that $P \xrightarrow{\bar{x}y} P'$ and hence, by applying the rules in Table 4, we can derive the transition $P|Q \xrightarrow{\bar{x}y} P'|Q$ from which we have $B_1 \xrightarrow{\Delta\bar{x}y} \beta(x, \Delta)I^*[P'|Q] = B_1'$. By $\preceq_b$ definition it results $B_0' \preceq_b B_1'$. □

Corollary 4.26. *$\preceq_b$ is strongly compatible with $\xrightarrow{\tau}$.*

We now introduce some auxiliary functions that will be used to prove that $\preceq_b$ is a wqo. The *Sub* function generates the set of all *possible* sequential and replicated subprocesses of a given process.

Definition 4.27. Let $P \in \mathcal{P}$ and $S \subseteq 2^{\mathcal{N}}$ be a finite set of names. The set of *possible* sequential and replicated subprocesses of P over the set of names S is defined as:

$$
\begin{aligned}
Sub(\mathsf{nil}, S) &= \emptyset \\
Sub(\bar{x}\langle m\rangle.P, S) &= \{\bar{x}\langle m\rangle.P\} \cup Sub(P, S) \\
Sub(x(m).P, S) &= \{x(m).P\} \cup \left(\bigcup_{n\in S} Sub(P\{n/m\}, S)\right) \\
Sub(M + N, S) &= \{M + N\} \cup Sub(M, S) \cup Sub(N, S) \\
Sub(P \mid Q, S) &= Sub(P, S) \cup Sub(Q, S) \\
Sub(!\bar{x}\langle m\rangle.P, S) &= \{!\bar{x}\langle m\rangle.P\} \cup Sub(P, S) \\
Sub(!x(m).P, S) &= \{!x(m).P\} \cup \left(\bigcup_{n\in S} Sub(P\{n/m\}, S)\right)
\end{aligned}
$$

The set of processes generated by the application of the function *Sub* on a process P and a finite set of names S is finite.

Lemma 4.28. *Let $P \in \mathcal{P}$ and $S \subseteq 2^{\mathcal{N}}$ be a finite set of names. Then $Sub(P, S)$ is finite.*

Proof. By induction on the structure of P.
(case nil) The empty set is finite.
(case $\bar{x}\langle m\rangle.P$) By inductive hypothesis $Sub(P, S)$ is finite and hence by only adding the element $\bar{x}\langle m\rangle.P$ the set is finite.
(case $x(m).P$) By inductive hypothesis for all $n \in S$ the set $Sub(P\{n/m\}, S)$ is finite. Since S is finite, then the union of a finite number of finite sets is a finite set. Moreover, by adding the process $x(m).P$ this set remains finite.
(case $M + N$) By inductive hypothesis $Sub(M, S)$ and $Sub(N, S)$ are finite set and hence their union with the set $\{M + N\}$ results in a finite set.
The other cases are similar. □

Corollary 4.29. *Let $P \in \mathcal{P}$. Then $Sub(P, fn(P))$ is finite.*

Proof. Immediate from the fact that $fn(P)$ is a finite set of names and by Lemma 4.28. □

We now prove some useful properties of the function *Sub*.

Lemma 4.30. *Let $P \in \mathcal{P}$ and $S, S' \subseteq 2^{\mathcal{N}}$. If $S' \subseteq S$ then $Sub(P, S') \subseteq Sub(P, S)$.*

Proof. By induction on the structure of P.
(Case $x(m).P$) By inductive hypothesis for all names $n \in S'$ we have that $Sub(P\{n/m\}, S') \subseteq Sub(P\{n/m\}, S)$. As a consequence, we obtain the condition $\bigcup_{n\in S'} Sub(P\{n/m\}, S') \subseteq \bigcup_{n\in S} Sub(P\{n/m\}, S)$ and hence the lemma holds.
The other cases are trivial. □

Lemma 4.31. *Let $P, Q \in \mathcal{P}$ and $S, S', S'' \subseteq 2^{\mathcal{N}}$. If $S' \subseteq S''$ and $Sub(P, S') \subseteq Sub(Q, S'')$ then $Sub(P, S' \cup S) \subseteq Sub(Q, S'' \cup S)$.*

Proof. By induction on the structure of P.
(Case $\overline{x}\langle m\rangle.P$) By definition $Sub(\overline{x}\langle m\rangle.P', S') = \{\overline{x}\langle m\rangle.P'\} \cup Sub(P', S')$. By hypothesis and Lemma 4.30 we have $Sub(\overline{x}\langle m\rangle.P', S') \subseteq Sub(Q, S'') \subseteq Sub(Q, S'' \cup S)$ and therefore $\overline{x}\langle m\rangle.P' \in Sub(Q, S'' \cup S)$. By definition of Sub function and by inductive hypothesis, this means that $Sub(P', S'' \cup S) \subseteq Sub(Q, S'' \cup S)$ and hence the lemma holds.
(Case $x(m).P$) By definition we have that $Sub(x(m).P', S') = \{x(m).P'\} \cup (\bigcup_{n\in S'} Sub(P'\{n/m\}, S'))$. By hypothesis and Lemma 4.30 $Sub(\overline{x}\langle m\rangle.P', S') \subseteq Sub(Q, S'') \subseteq Sub(Q, S'' \cup S)$ and therefore $\overline{x}\langle m\rangle.P' \in Sub(Q, S'' \cup S)$. By definition of Sub function and by inductive hypothesis, this means that for all $n \in S'' \cup S$ we have $Sub(P'\{n/m\}, S'' \cup S) \subseteq Sub(Q, S'' \cup S)$ and hence the lemma follows.
(Case $P_0|P_1$) By hypothesis and Sub function definition we have $Sub(P_0, S') \subseteq Sub(Q, S'')$ and $Sub(P_1, S') \subseteq Sub(Q, S'')$. By inductive hypothesis we have $Sub(P_0, S' \cup S) \subseteq Sub(Q, S'' \cup S)$ and $Sub(P_1, S' \cup S) \subseteq Sub(Q, S'' \cup S)$ and therefore $Sub(P_0, S' \cup S) \cup Sub(P_1, S' \cup S) \subseteq Sub(Q, S'' \cup S)$. The lemma follows.
The other cases are similar. □

The Sub function definition and its properties can be extended to safe bio-processes.

Definition 4.32. Let $B \in \mathcal{Z}_s$ and $S \subseteq 2^{\mathcal{N}}$ be a finite set of names. The set of *possible* subprocesses of B over the set of names S is defined as:

$$Sub(\mathsf{Nil}, S) = \emptyset$$
$$Sub(I[P], S) = \{I[P'] \mid P' \in Sub(P, S \cup sub(I))\}$$
$$Sub(B \parallel B', S) = Sub(B, S) \cup Sub(B', S)$$

Lemma 4.33. *Let $B \in \mathcal{Z}_s$ and $S \subseteq 2^{\mathcal{N}}$ be a finite set of names. Then $Sub(B, S)$ is finite.*

Proof. By induction on the structure of B.
(case Nil) The empty set is finite.
(case $I[P]$) From the fact that $sub(I)$ is finite and by Lemma 4.28, we have that $Sub(P, S \cup sub(I))$ is finite and hence the set $\{I[P'] \mid P' \in Sub(P, S \cup sub(I))\}$ is finite.
(case $B \parallel B'$) By inductive hypothesis $Sub(B, S)$ and $Sub(B', S)$ are finite sets and therefore their union is finite. □

Corollary 4.34. *Let $B \in \mathcal{Z}_s$. Then $Sub(B, fn(B))$ is finite.*

Proof. Immediate from the fact that $fn(B)$ is a finite set of names and by Lemma 4.33. □

Lemma 4.35. *Let $B \in \mathcal{Z}_s$ and $S, S' \subseteq 2^{\mathcal{N}}$. If $S' \subseteq S$ then $Sub(B, S') \subseteq Sub(B, S)$.*

Proof. By induction on the structure of B.
(Case $I[P]$) By Lemma 4.30 we have $Sub(P, S' \cup sub(I)) \subseteq Sub(P, S \cup sub(I))$. This means $\{I[P']|P' \in Sub(P, S' \cup sub(I))\} \subseteq \{I[P']|P' \in Sub(P, S \cup sub(I))\}$ and the lemma follows.
The other cases are trivial. □

Lemma 4.36. *Let $B, B' \in \mathcal{Z}_s$ and $S, S', S'' \subseteq 2^{\mathcal{N}}$. If $S' \subseteq S''$ and $Sub(B, S') \subseteq Sub(B', S'')$ then $Sub(B, S' \cup S) \subseteq Sub(B', S'' \cup S)$.*

Proof. By induction on the structure of B.
(Case $I[P]$) By hypothesis and Sub function definition we have $Sub(P, S' \cup sub(I)) \subseteq Sub(P, S'' \cup sub(I))$. By Lemma 4.30 we have $Sub(P, S' \cup sub(I) \cup S) \subseteq Sub(P, S'' \cup sub(I) \cup S)$. This means $\{I[P']|P' \in Sub(P, S' \cup sub(I) \cup S)\} \subseteq \{I[P']|P' \in Sub(P, S'' \cup sub(I) \cup S)\}$ and the lemma follows.
The other cases are trivial. □

To prove that $\preceq_b$ is a wqo we need some preliminary results. The first result states that, given a transition $P \xrightarrow{\theta} P'$ over processes and a transition $B \xrightarrow{\theta} B'$ over bio-processes, the set of possible sequential and replicated processes of P' over $fn(P')$ and of B' over $fn(B')$ are delimited by the set of possible sequential and replicated processes of P over $fn(P)$ and of B over $fn(B)$ and the names in θ.

Lemma 4.37. *Let $P, P' \in \mathcal{P}$ and $S \in 2^{\mathcal{N}}$. Suppose $P \xrightarrow{\theta} P'$, then*

(a) *if $\theta = \overline{x}y$ then $Sub(P', fn(P')) \subseteq Sub(P, fn(P))$*
(b) *if $\theta = xy$ then $Sub(P', fn(P')) \subseteq Sub(P, fn(P) \cup \{y\})$*
(c) *if $\theta = \tau$ then $Sub(P', fn(P')) \subseteq Sub(P, fn(P))$.*

Proof. For all the three statements the proof is by induction on the inference of $P \xrightarrow{\theta} P'$. Cases rep_in and rep_out are similar to pi_in and pi_out. The pi_par cases are similar for all the three statements and hence we provide the proof only for (a). The pi_sum cases are similar to the pi_par ones.
(a)

(Case pi_out) We have $P = \bar{x}\langle y\rangle.P' \xrightarrow{\bar{x}y} P'$. By definition we have that $Sub(P, fn(P)) = \{\bar{x}\langle y\rangle.P'\} \cup Sub(P', fn(P))$. By Lemma 4.10 we know that $fn(P') \subseteq fn(P)$ and hence (by Lemma 4.30) $Sub(P', fn(P')) \subseteq Sub(P', fn(P))$. As a consequence $Sub(P', fn(P')) \subseteq Sub(P, fn(P))$.

(Case pi_par) By hypothesis we have $P_0 \xrightarrow{\bar{x}y} P_0'$ and hence, by inductive hypothesis, we obtain $Sub(P_0', fn(P_0')) \subseteq Sub(P_0, fn(P_0))$. By Lemma 4.10 we know $fn(P_0') \subseteq fn(P_0)$ and hence by Lemma 4.31 $Sub(P_0', fn(P_0') \cup fn(P_1)) \subseteq Sub(P_0, fn(P_0) \cup fn(P_1))$. Moreover, by Lemma 4.30 we obtain $Sub(P_1, fn(P_0') \cup fn(P_1)) \subseteq Sub(P_1, fn(P_0) \cup fn(P_1))$. By definition of the *Sub* function the lemma follows.
(b)

(Case pi_in) We have $P = x(w).P' \xrightarrow{xy} P'\{y/w\}$. By definition we have that $Sub(P, fn(P) \cup \{y\}) = \{\bar{x}\langle w\rangle.P'\} \cup (\bigcup_{n \in fn(P) \cup \{y\}} Sub(P'\{n/w\}, fn(P) \cup \{y\})$ and therefore contains the subset $Sub(P'\{y/w\}, fn(P) \cup \{y\})$. By Lemma 4.10 we know that $fn(P'\{y/w\}) \subseteq fn(P) \cup \{y\}$ and hence (by Lemma 4.30) $Sub(P'\{y/w\}, fn(P'\{y/w\})) \subseteq Sub(P'\{y/w\}, fn(P) \cup \{y\})$. The lemma follows.
(c)

(Case l_intra) By hypothesis we have $P_0 \xrightarrow{\bar{x}z} P_0'$ and $P_1 \xrightarrow{xz} P_1'$. By (a) and (b) we have that $Sub(P_0', fn(P_0')) \subseteq Sub(P_0, fn(P_0))$ and $Sub(P_1', fn(P_1')) \subseteq Sub(P_1, fn(P_1) \cup \{z\})$. By applying Lemmas 4.10, 4.30 and 4.31 we obtain $Sub(P_0', fn(P_0') \cup fn(P_1')) \subseteq Sub(P_0, fn(P_0) \cup fn(P_1))$ and $Sub(P_1', fn(P_0') \cup fn(P_1')) \subseteq Sub(P_1, fn(P_0) \cup fn(P_1))$ and, by *Sub* function definition, the lemma follows. □

Lemma 4.38. *Let $B, B' \in \mathcal{Z}_s$. Suppose $B \xrightarrow{\theta} B'$. Then*

(a) *if $\theta = \Delta\bar{x}y$ then $Sub(B', fn(B')) \subseteq Sub(B, fn(B))$*
(b) *if $\theta = \Delta xy$ then $Sub(B', fn(B')) \subseteq Sub(B, fn(B) \cup \{y\})$*
(c) *if $\theta = \tau$ then $Sub(B', fn(B')) \subseteq Sub(B, fn(B))$.*

Proof. For all the three statements the proof is by induction on the inference of $B \xrightarrow{\theta} B'$. We present only the most important cases. The other cases can be proved using the results and the ideas of Lemma 4.37.
(a)

(Case bio_out) By hypothesis we know that $P \xrightarrow{\bar{x}y} P'$ and therefore by Lemma 4.37 we have $Sub(P', fn(P')) \subseteq Sub(P, fn(P))$. By applying Lemma 4.31 we obtain $Sub(P', fn(P') \cup sub(I^*) \cup \{x\}) \subseteq Sub(P, fn(P) \cup sub(I^*) \cup \{x\})$. But this means that $\{(x, \Delta)I^*[P''] | P'' \in Sub(P', fn(P') \cup sub(I^*) \cup \{x\})\} \subseteq \{(x, \Delta)I^*[P''] | P'' \in Sub(P, fn(P) \cup sub(I^*) \cup \{x\})\}$ and the lemma follows.
(b)

(Case bio_in) By hypothesis we know that $P \xrightarrow{xy} P'$ and therefore by Lemma 4.37 we have $Sub(P', fn(P')) \subseteq Sub(P, fn(P) \cup \{y\})$. By applying Lemma 4.31 we obtain $Sub(P', fn(P') \cup sub(I^*) \cup \{x\}) \subseteq Sub(P, fn(P) \cup sub(I^*) \cup \{x, y\})$. But this means that $\{(x, \Delta)I^*[P''] | P'' \in Sub(P', fn(P') \cup sub(I^*) \cup \{x\})\} \subseteq \{(x, \Delta)I^*[P''] | P'' \in Sub(P, fn(P) \cup sub(I^*) \cup \{x, y\})\}$ and the lemma follows. □

Now, we define a superset of the set of derivatives of a bio-process B, denoted with $\mathcal{P}_B$. This set includes all the bio-processes whose possible sequential and replicated subprocesses are contained in the corresponding elements of B.

Definition 4.39. Let $B \in \mathcal{Z}_s$. Then

$$\mathcal{P}_B = \{B' \in \mathcal{Z}_s \mid Sub(B', fn(B')) \subseteq Sub(B, fn(B)) \wedge box_\#(B') = box_\#(B)\}$$

The following result describes how given a bio-process B and a bio-process $B' \in \mathcal{P}_B$, all the derivatives of B' are contained in $\mathcal{P}_B$.

Lemma 4.40. *Let $B \in \mathcal{Z}_s$ and $B' \in \mathcal{P}_B$. If $B' \xrightarrow{\tau} B''$ then $B'' \in \mathcal{P}_B$.*

Proof. $B' \in \mathcal{P}_B$ implies $Sub(B', fn(B')) \subseteq Sub(B, fn(B))$. Since by hypothesis $B' \xrightarrow{\tau} B''$, by Lemma 4.38 we have $Sub(B'', fn(B'')) \subseteq Sub(B', fn(B'))$. Transitivity of $\subseteq$ proves $Sub(B'', fn(B'')) \subseteq Sub(B, fn(B))$, which means $B'' \in \mathcal{P}_B$. □

A consequence of this lemma is that all the derivatives of a bio-process B are contained in $\mathcal{P}_B$.

Corollary 4.41. *Let $B \in \mathcal{Z}_s$. Then $Deriv(B) \subseteq \mathcal{P}_B$.*

Proof. Immediate from Lemma 4.40. □

The following lemma shows how a bio-process in $\mathcal{P}_B$ can be rewritten (up to $\equiv_b^{dec}$) as a parallel composition of boxes that are in relation with boxes in $Sub(B, fn(B))$.

Lemma 4.42. *Let $B \in \mathcal{Z}_s$ and $B' \in \mathcal{P}_B$. Then we have that*

$$B' \equiv_b^{dec} \prod_{i=1}^{n} I_i \left[\prod_{j=1}^{m} Q_{i,j} \right]$$

with $I_i[Q_{i,j}] \in Sub(B, fn(B))$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Proof. Immediate from the bio-processes syntax and from the definition of *Sub* function. □

In the following lemma we define the relation $=_*$ over bio-processes according to Definition 4.5.

Lemma 4.43. *Let $B \in \mathcal{Z}_s$ and $I[P_1], \ldots, I[P_n], I'[Q_1], \ldots, I'[Q_m]$ belonging to $Sub(B, fn(B))$. If $I[P_1], \ldots, I[P_n] =_* I'[Q_1], \ldots, I'[Q_m]$ then $I[\prod_{i=1}^{n} P_i] \preceq_b I'[\prod_{i=1}^{m} Q_i]$.*

Proof. If $I[P_1], \ldots, I[P_n] =_* I'[Q_1], \ldots, I'[Q_m]$ then there exists an injection $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $I[P_i] = I'[Q_{f(i)}]$ and $i = f(i)$, with $i = 1, \ldots, n$. The injection f corresponds to the identity function of the set $\{1, \ldots, n\}$ and hence for each i in $\{1, \ldots, n\}$ we have $I[P_i] = I'[Q_i]$. By $\equiv_b^{dec}$ definition we can write $I'[\prod_{i=1}^{m} Q_i] \equiv_b^{dec} I'[\prod_{i=1}^{n} Q_i \mid \prod_{i=n+1}^{m} Q_i]$ and by the equality result we have $I'[\prod_{i=1}^{m} Q_i] \equiv_b^{dec} I'[\prod_{i=1}^{n} P_i \mid \prod_{i=n+1}^{m} Q_i]$ (notice that $I = I'$), which means $I[\prod_{i=1}^{n} P_i] \preceq_b I'[\prod_{i=1}^{m} Q_i]$. □

The following theorem shows that the qo $\preceq_b$ is a wqo.

Theorem 4.44. *Let $B \in \mathcal{Z}_s$. The relation $\preceq_b$ is a wqo over $\mathcal{P}_B$.*

Proof. We take an infinite sequence $B_1, \ldots, B_i, \ldots$ such that $B_i \in \mathcal{P}_B$ for $i > 0$. By Lemma 4.42, for any i we have that:

$$B_i \equiv_b^{dec} \prod_{j=1}^{n} I_{i,j} \left[\prod_{k=1}^{m_{i,j}} P_{i,j,k} \right].$$

Hence, each B_i can be seen as composed of n finite sequences:

$$\begin{array}{l} I_{i,1}[P_{i,1,1}], \ldots, I_{i,1}[P_{i,1,m_{i,1}}] \\ I_{i,2}[P_{i,2,1}], \ldots, I_{i,2}[P_{i,2,m_{i,2}}] \\ \cdots \\ I_{i,n}[P_{i,n,1}], \ldots, I_{i,n}[P_{i,n,m_{i,n}}] \end{array}$$

Note that all the sequences are composed of elements from the finite set $Sub(B, fn(B))$. Each sequence is hence an element of $Sub(B, fn(B))^*$ and hence we have n infinite sequences of elements in $Sub(B, fn(B))^*$. By Corollary 4.34 $Sub(B, fn(B))$ is finite, and by applying Lemma 4.7 and Higman's Theorem 4.6 we have that $=_*$ is a wqo over $Sub(B, fn(B))^*$.

Now, we can extract an infinite subsequence from $B_1, \ldots, B_i, \ldots$ making the finite sequences $I_{i,1}[P_{i,1,1}], \ldots, I_{i,1}[P_{i,1,m_{i,1}}]$ increasing w.r.t. $=_*$; then, we continue by extracting an infinite subsequence from the subsequence obtained previously, making the finite sequences $I_{i,2}[P_{i,2,1}], \ldots, I_{i,2}[P_{i,2,m_{i,2}}]$ increasing also in this case w.r.t. $=_*$. We continue for all the n subsequences.

We end up with an infinite subsequence $B_{n_0}, \ldots, B_{n_i}, \ldots$ (with $n_0 < \cdots < n_i < \cdots$) of $B_1, \ldots, B_i, \ldots$ such that all the n finite sequences are ordered w.r.t. $=_*$. By Lemma 4.43 we obtain:

$$I_{n_0,j} \left[\prod_{l=1}^{m_{n_0,j}} P_{n_0,j,l} \right] \preceq_b \cdots \preceq_b I_{n_i,j} \left[\prod_{l=1}^{m_{n_i,j}} P_{n_i,j,l} \right] \preceq_b \cdots \quad \text{for } j = 1, \ldots, n$$

from which we finally obtain $B_{n_0} \preceq_b \cdots \preceq_b B_{n_i} \preceq_b \cdots$. □

The hypothesis of Theorem 4.4 are satisfied by the following theorem.

Theorem 4.45. *Let $B \in \mathcal{Z}_s$. The transition system $(Deriv(B), \xrightarrow{\tau}, \preceq_b)$ is a well-structured transition system with decidable $\preceq_b$ and computable Succ.*

Proof. The relation $\preceq_b$ has been proved strong compatible in Theorem 4.25. Moreover, the fact that $\preceq_b$ is a wqo on $Deriv(B)$ is a consequence of Corollary 4.41 and Theorem 4.44.

Given $B, B' \in Deriv(B)$, deciding whether $B \preceq_b B'$ means to find a subterm B'' of B' such that $B' \equiv_b^{dec} B \parallel B''$, which is a decidable problem. □

Corollary 4.46. *Let $B \in \mathcal{Z}_s$. The termination of B is decidable.*

5. Undecidability results

In this section we prove that termination is undecidable for BL^{gp} and BL^{e}. We show this by providing encodings of Random Access Machines (RAMs) [39], a well-known Turing-complete formalism, into BL^{gp} and BL^{e}. First of all we recall the definition of RAMs.

5.1. Random Access Machine

A Random Access Machine (RAM) is an abstract machine in the general class of register machines. RAMs are a computational model based on finite programs acting on a finite set of registers.

A RAM R is composed of a finite set of registers $r_1, \ldots, r_n$ and a sequence of indexed instructions $(1, I_1), \ldots, (m, I_m)$. Registers store natural numbers, one for each register, and can be updated (incremented or decremented) and tested for zero. In [25] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Incr(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the instruction indexed with the number 1 and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

The state of a RAM R is a tuple $(j, k_1, \ldots, k_n)$ where j is the index of next instruction to be executed and $k_1, \ldots, k_n$ are the current contents of the registers. The execution is defined by a transition relation among states

$$(j, k_1, \ldots, k_n) \rightarrow_R (j', k'_1, \ldots, k'_n)$$

meaning that the state of the RAM changes from $(j, k_1, \ldots, k_n)$ to $(j', k'_1, \ldots, k'_n)$, as a consequence of the execution of the jth instruction.

A state $(j, k_1, \ldots, k_n)$ is terminated if the program counter j is greater than the number of instructions m. We say that a RAM R terminates if its computation reaches a terminated state.

5.2. Encoding with BL^{gp}

BlenX is a stochastic language, i.e. quantitative information about speed and probability of actions (in the form of reaction rates) is provided with systems specifications. In particular, by using infinite reaction rates we can define *immediate* actions, i.e. actions that have precedence over the actions with a reaction rate in $\mathbb{R}$. In other words, immediate actions allow us to create a two level priority between actions. Here we do not want to consider the stochastic domain, but we want to show that immediate actions can increase the expressive power of a language. In order to do this, we enrich the *BL* language by introducing a priority mechanism and we show that this extension turns out to be Turing equivalent by providing an encoding of RAMs. Priority is a frequently used feature of many computational systems and many process algebras have been enriched with some priority mechanisms [6,9,29]. How priorities affect the expressive power of a language has been previously studied in [29,40]. In this work we use a mechanism based on *global* priorities [2,9], where high-priority actions are able to preempt any other low-priority action in the system. In *BL* we handle priorities with an approach similar to the one proposed by Cleaveland and Hennessy [9], where prioritized actions are represented with underlined names.

The key ingredient in this encoding is the combined use of choice and priorities; boxes and types are used only to maintain a certain homogeneity and uniformity w.r.t. this encoding and the one presented in the next section. Very recently, in [1] the authors show an encoding of RAMs into a subset of CCS with replication and enriched with priorities which is similar in the spirit to the one here presented.

The *BL* language is enriched with global priorities by adding a new *immediate* output action:

$$\pi ::= \cdots \mid \underline{\overline{x}}\langle y \rangle$$

which has precedence over the usual $\overline{x}\langle y \rangle$. In particular, we call *immediate* communications those that involve an immediate output. Well-formedness conditions and structural congruence relations $\equiv_p$ and $\equiv_b$ remain unchanged w.r.t. Section 3 and the new reduction semantics is reported in Table 5. Moreover, we denote with BL^{gp} this extension and with $\mathcal{Z}^{gp}$ the set of well-formed systems of BL^{gp}. Table 5 shows that intra-communications and inter-communications with outputs $\overline{x}\langle y \rangle$ can be derived only if, respectively, no intra-communications and inter-communications through immediate outputs $\underline{\overline{x}}\langle y \rangle$ are enabled. Notice that in Table 5 with $B \not\twoheadrightarrow$ we mean that no bio-process B' such that $B \twoheadrightarrow B'$ exists.

It is important to note that the introduction of priorities causes the generation of transition systems which are not well-structured. Indeed, since priorities remove certain possibilities that would have existed without priorities, we are no longer able to define a quasi-ordering over bio-processes (of the kind presented in Section 4) that satisfies the *strong compatibility* property.

Definition 5.1. The BL^{gp} Transition System (TS^{gp}) is referred as $(\mathcal{Z}^{gp}, \rightarrow)$, where $\mathcal{Z}^{gp}$ is the set of well-formed systems and $\rightarrow \subseteq \mathcal{Z}^{gp} \times \mathcal{Z}^{gp}$ is the transition relation.

As usual, $\rightarrow^+$ indicates the transitive closure and $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$. Moreover, $Z \rightarrow^k Z'$ with $k \geq 1$ indicates that Z' can be reached from Z with k transitions $Z \rightarrow^1 Z_1 \rightarrow^2 \cdots \rightarrow^{k-1} Z_{k-1} \rightarrow^k Z'$.

Table 5
Reduction semantics of BL^{gp}.

(intra$_i$)	$I[\overline{x}\langle z\rangle.P_1 + M_1 \mid x(w).P_2 + M_2 \mid P_3] \parallel B \twoheadrightarrow I[P_1 \mid P_2\{^z/_w\} \mid P_3] \parallel B$
(intra)	$I[\overline{x}\langle z\rangle.P_1 + M_1 \mid x(w).P_2 + M_2 \mid P_3] \parallel B \mapsto I[P_1 \mid P_2\{^z/_w\} \mid P_3] \parallel B$
(inter$_i$)	$\dfrac{P_1 \equiv_p \overline{x}\langle z\rangle.R_1 + M_1 \mid Q_1 \qquad P_2 \equiv_p y(w).R_2 + M_2 \mid Q_2}{I_1[P_1] \parallel I_2[P_2] \parallel B \twoheadrightarrow I_1[R_1 \mid Q_1] \parallel I_2[R_2\{^z/_w\} \mid Q_2] \parallel B}$ provided $\alpha(\Gamma, \Delta) > 0$ and $z \notin sub(I_1)$ and where $I_1 = \beta(x,\ \Delta)\, I_1^*$ and $I_2 = \beta(y,\ \Gamma)\, I_2^*$
(inter)	$\dfrac{P_1 \equiv_p \overline{x}\langle z\rangle.R_1 + M_1 \mid Q_1 \qquad P_2 \equiv_p y(w).R_2 + M_2 \mid Q_2}{I_1[P_1] \parallel I_2[P_2] \parallel B \mapsto I_1[R_1 \mid Q_1] \parallel I_2[R_2\{^z/_w\} \mid Q_2] \parallel B}$ provided $\alpha(\Gamma, \Delta) > 0$ and $z \notin sub(I_1)$ and where $I_1 = \beta(x,\ \Delta)\, I_1^*$ and $I_2 = \beta(y,\ \Gamma)\, I_2^*$
(struct$_i$)	$\dfrac{B_1 \equiv_b B_1' \qquad B_1' \twoheadrightarrow B_2' \qquad B_2' \equiv_b B_2}{B_1 \rightarrow B_2}$
(struct)	$\dfrac{B_1 \equiv_b B_1' \qquad B_1' \mapsto B_2' \qquad B_2' \equiv_b B_2 \qquad B_1' \not\rightarrow}{B_1 \rightarrow B_2}$

Consider a RAM R with program $(1, I_1), \ldots, (m, I_M)$ and state $(j, k_1, \ldots, k_n)$. The encoding of the RAM is:

$$[\![(j, k_1, \ldots, k_n)]\!]_R^{gp} = B$$

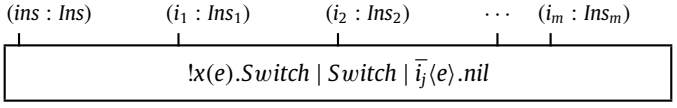
where,

$$B \triangleq_B Switch_j \parallel [\![(1, I_1)]\!]_R^{gp} \parallel \cdots \parallel [\![(m, I_m)]\!]_R^{gp} \parallel [\![r_1 = k_1]\!]_R^{gp} \parallel \cdots \parallel [\![r_n = k_n]\!]_R^{gp}$$

The $Switch_j$ bio-process is defined in the following way

$$Switch_j \triangleq_B \beta(ins : Ins)\beta(i_1 : Ins_1)\beta(i_2 : Ins_2)\cdots\beta(i_m : Ins_m) [\ !x(e).Switch \mid Switch \mid \overline{i_j}\langle e\rangle.nil\ ]$$
$$Switch \triangleq_P ins(type).(\overline{type}\langle e\rangle.nil \mid (\textstyle\sum_{l=1}^{m} ins_l(e).\overline{x}\langle e\rangle.\overline{i_l}\langle e\rangle.nil))$$

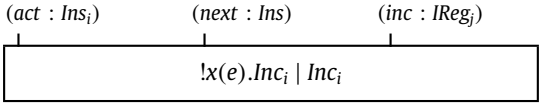
and its graphical representation is



The encoding of the instruction $(i, Incr(r_j))$ is

$$[\![(i, Incr(r_j))]\!]_R^{gp} = \beta(act : Ins_i)\beta(next : Ins)\beta(inc : IReg_j) [\ !x(e).Inc_i \mid Inc_i\ ]$$
$$Inc_i \triangleq_P act(e).\overline{inc}\langle e\rangle.\overline{x}\langle e\rangle.\overline{next}\langle ins_{i+1}\rangle.nil$$

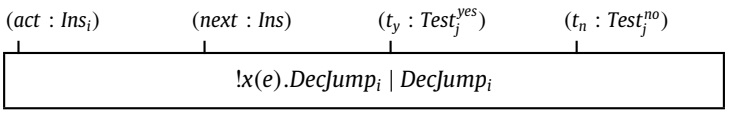
and its graphical representation is



The encoding of the instruction $(i, DecJump(r_j, s))$ is

$$[\![(i, DecJump(r_j, s))]\!]_R^{gp} = \beta(act : Ins_i)\beta(next : Ins)\beta(t_y : Test_j^{yes}) \beta(t_n : Test_j^{no}) [\ !x(e).DecJump_i \mid DecJump_i\ ]$$
$$DecJump_i \triangleq_P act(e).(t_y(e).Dec_i + t_n(e).Jump_i)$$
$$Dec_i \triangleq_P \overline{x}\langle e\rangle.\overline{next}\langle ins_{i+1}\rangle.nil$$
$$Jump_i \triangleq_P \overline{x}\langle e\rangle.\overline{next}\langle ins_s\rangle.nil$$

and its graphical representation is



The encoding of a register r_j with content l is defined as follows

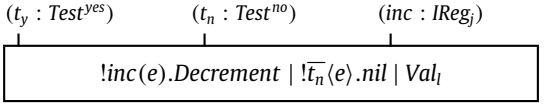
$$[\![r_j = l]\!]_R^{gp} \triangleq_B \beta(t_y : Test_j^{yes})\beta(t_n : Test_j^{no})\beta(inc : IReg_j) \\ [\ !inc(e).Decrement \mid !\overline{t_n}\langle e\rangle.nil \mid Val_l\ ]$$

where,

$$Val_l \triangleq_P \underbrace{Decrement \mid \cdots \mid Decrement}_{l}$$

$$Decrement \triangleq_P \overline{t_y}\langle e\rangle.nil$$

The graphical representation of $[\![r_j = l]\!]_R^{gp}$ is



Finally, the function α is defined in the following way

$$\alpha(Type_1, Type_2) = \begin{cases} 1 & \text{if } Type_1 = Type_2 \\ 0 & otherwise. \end{cases}$$

The encoding produces a system in $\mathcal{Z}^{gp}$ due to the unique immediate output in the definition of the process *Decrement*. This encoding is a parallel composition of a switching box, which controls the activation of the instructions sequence, m boxes encoding instructions and n boxes encoding registers. The two types of instructions are encoded in different ways, but in both cases the encoding box is activated by performing an inter-communication on the channel *act* with the switching box $Switch_j$.

Each register r_j is modeled with a box whose internal process structure depends on the content of the register. A register can be incremented and tested for not zero value. The number of parallel unguarded *Decrement* processes present in the internal structure of the box represents the content of the register.

The box encoding the instruction $(i, Incr(r_j))$, after its activation, consumes an inter-communication with the box encoding the register r_j (through the interfaces of type $IReg_j$), representing a request for its increment. In the register box, the inter-communications produces the replication of the process *Decrement*, representing the increment of one, while in the instruction box, the inter-communication produces the replication of the internal machinery and the consumption of an inter-communication with the switching box (through the interfaces of type *Ins*) for the activation of instruction $i + 1$.

The box encoding the instruction $(i, DecJump(r_j, s))$, after its activation, presents an alternative behaviour (encoded with the choice operator), which implements the mechanism used for testing the content of the register r_j. In particular, the content of the register is tested with two alternative inter-communications on channels t_y and t_n through interfaces of types $Test^{yes}$ and $Test^{no}$, respectively. In the register box, outputs on channel t_y, if present, are of the form $\overline{t_y}\langle e\rangle$ and hence generate *immediate* inter-communications.

If the encoded register contains a value $n > 0$, then n parallel compositions of process $Decrement \triangleq_P \overline{t_y}\langle e\rangle.nil$ are present and hence inter-communications on output $\overline{t_y}\langle e\rangle$ have always precedence w.r.t. the inter-communications that the process $!\overline{t_n}\langle e\rangle.nil$ offers. In the register box, the consumption of an immediate inter-communication deletes an instance of *Decrement* process in its internal structure, representing the decrement of one, while in the instruction box, the consumption of an immediate inter-communication enables the process Dec_i, which replicates its internal box machinery and performs an inter-communication with the switching box (through the interfaces of type *Ins*) for the activation of instruction $i + 1$.

If the encoded register contains the value 0, then no unguarded *Decrement* processes are present in the internal structure of the register box and hence the inter-communications that the process $!\overline{t_n}\langle e\rangle.nil$ offers can be consumed. This causes, in the instruction box, the activation of the process $Jump_i$, which replicates its internal box machinery and performs an inter-communication with the switching box for the activation of instruction s.

A formal proof of the encoding correctness follows.

Lemma 5.2. *Let R be a RAM with program* $(1, I_1), \ldots, (m, I_m)$ *and state* $(j, k_1, \ldots, k_n)$.

If $(j, k_1, \ldots, k_n) \to_R (j', k'_1, \ldots, k'_n)$, *then there exists a well-formed system* $Z \in \mathcal{Z}^{gp}$ *such that* $[\![(j, k_1, \ldots, k_n)]\!]_R^{gp} \to^+ Z$ *and* $Z \equiv_b [\![(j', k'_1, \ldots, k'_n)]\!]_R^{gp}$.

Proof. The proof is by case analysis. There are three cases: (i) Instruction $I_j = DecJump(r_l, s)$ with r_l content greater than zero; (ii) Instruction $I_j = DecJump(r_l, s)$ with r_l content equal to zero; (iii) Instruction $I_j = Incr(r_l)$. We only prove case (i), because the other cases can be proved similarly.

(i) We consider the computation of the bio-process $[\![(j, k_1, \ldots, k_n)]\!]_R^{gp}$. An inter-communication between the component $Switch_j$ and the component $[\![(j, DecJump(r_l, s))]\!]_R^{gp}$ is consumed. In particular, the two boxes synchronize on output $\overline{i_j}\langle e\rangle$ and input $act(e)$ through their binders of type Ins_j. This cause the activation of the $[\![(j, DecJump(r_l, s))]\!]_R^{gp}$ component. Notice that, after the inter-communication, the components codifying for the other instructions, the registers and the switching box are blocked.

The activation of the $[\![(j, DecJump(r_l, s))]\!]_R^{gp}$ box causes the enabling of a choice process. This process is used for testing the content of the register r_j that is a choice composition of two processes blocked on inputs $t_y(e)$ and $t_n(e)$, bound to the interfaces $Test^{yes}$ and $Test^{no}$, respectively. By hypothesis the content of the register r_l is greater than 0 and hence the internal structure of the box $[\![r_l = k_l]\!]_R^{gp}$ is a parallel composition of processes that contains at least one *Decrement* processes. Two types of inter-communications between the boxes encoding the instructions and the registers are enabled. One inter-communication through interfaces with type $Test^{no}$ and k_j inter-communications through interfaces with type $Test^{yes}$. However, since the inter-communications through interfaces with type $Test^{yes}$ are immediate, they have precedence w.r.t. the one through interfaces with type $Test^{no}$. The consumption of one of the immediate inter-communications deletes one of the k_l *Decrement* processes in $[\![r_l = k_l]\!]_R^{gp}$, resulting (for all the possible inter-communications of this type) in a bio-process structurally congruent to $[\![r_l = k_l - 1]\!]_R^{gp}$, and activates process Dec_j in the instruction box.

At this point, an intra-communication in the instruction box on channel x replicates the internal machinery of the box and enables the process $\overline{next}\langle ins_{j+1}\rangle.nil$. This produces a synchronization between the instruction box and the switching box, which generates an inter-communication on output $\overline{next}\langle ins_{j+1}\rangle$ and input $ins(type)$ through interfaces of type Ins.

The instruction box is now returned in its form $[\![(j, DecJump(r_l, s))]\!]_R^{gp}$, while in the switching box the process

$$\left(\overline{ins_{j+1}}\langle e\rangle.nil \mid \left(\sum_{o=1}^{m} ins_o(e).\bar{x}\langle e\rangle.\overline{i_o}\langle e\rangle.nil\right)\right)$$

is enabled. An intra-communication on channel ins_{j+1} is consumed, the internal machinery is replicated with an intra-communication on channel x and the switching box is now structurally congruent to the box $Switch_{j+1}$. □

Lemma 5.3. *Let R be a RAM with program $(1, I_1), \ldots, (m, I_m)$ and state $(j, k_1, \ldots, k_n)$. If the system $Z = [\![(j, k_1, \ldots, k_n)]\!]_R^{gp}$ can produce a transition $Z \to Z_1$, then there exists a computation $Z \to Z_1 \to Z_2 \to \cdots \to Z_l$ such that $Z_l = [\![(j', k'_1, \ldots, k'_n)]\!]_R^{gp}$ and $(j, k_1, \ldots, k_n) \to_R (j', k'_1, \ldots, k'_n)$.*

Proof. Consider the structure of the bio-process $Z = [\![(j, k_1, \ldots, k_n)]\!]_R^{gp}$. If the bio-process Z can perform a first step $Z \to Z_1$, this corresponds to an inter-communication between the box $Switch_j$ and the box encoding for the instruction (j, I_j), representing the activation of the instruction box. The encoding definition ensures that the instruction (j, I_j) exists; hence the instruction can be executed in the state $(j, k_1, \ldots, k_n)$ of the RAM R, generating a new state $(j', k'_1, \ldots, k'_n)$.

There are three cases: (i) Instruction $I_j = DecJump(r_l, s)$ with r_l content greater than zero; (ii) Instruction $I_j = DecJump(r_l, s)$ with r_l content equal to zero; (iii) Instruction $I_j = Incr(r_l)$. In all the cases, it is possible to show that from the moment in which the switch activates an instruction till the moment in which the switch is able to activate a new instruction, the computation proceeds deterministically (up to structural congruence $\equiv_b$). The encoding is hence deterministic up to structural congruence. We prove only case (iii), because all the other cases can be proved similarly.

(iii) By encoding definition we have that the structure of the instruction box is

$$[\![(j, Incr(r_l))]\!]_R^{gp} = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l) \\ [!x(e).Inc_j \mid act(e).\overline{inc}\langle e\rangle.\bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

This box is the only one able to synchronize with the box $Switch_j$ for an inter-communication through interfaces of type Ins_j. After the inter-communication we have that in Z_1 the box encoding the instruction j becomes structurally congruent to

$$B' = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l) \\ [!x(e).Inc_j \mid \overline{inc}\langle e\rangle.\bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

At this point, the only possible action $Z_1 \to Z_2$ is the inter-communication between the box B' and the box $[\![r_l = k]\!]_R^{gp}$ through their interfaces of type $IReg_l$ on output $\overline{inc}\langle e\rangle$ and input $inc(e)$, respectively. After the inter-communication the box encoding for the register r_l becomes structurally congruent to

$$\beta(t_y : Test_j^{yes})\beta(t_n : Test_j^{no})\beta(inc : IReg_l) \\ [\ !inc(e).Decrement \mid !\overline{t_n}\langle e\rangle.nil \mid Val_k \mid Decrement\ ]$$

which corresponds to the box $[\![r_l = k + 1]\!]_R^{gp}$. Instead the box encoding the instruction j becomes structurally congruent to

$$B'' = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l) \\ [!x(e).Inc_j \mid \bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

Now, the action $Z_2 \to Z_3$ is the intra-communication of B'' on channel x which becomes a box B''' with internal structure $!x(e).Inc_j \mid \overline{next}\langle ins_{j+1}\rangle.nil$, and the action $Z_3 \to Z_4$ is the inter-communication between the box B''' and the switching box. After the inter-communication, the instruction box returns in its initial form $[\![(j, Incr(r_l))]\!]^{gp}_R$ and the switching box starts a sequence of intra-communications which produces a box structurally congruent to $Switch_{j+1}$ and representing the sequence of actions $Z_4 \to Z_5 \to Z_6$. It is easy to see that Z_6 is congruent to $[\![(j+1, k_1, \ldots, k_{l-1}, k+1, k_{l+1}, \ldots, k_n)]\!]^{gp}_R$. □

Lemmas 5.2 and 5.3 give us the instruments for proving the undecidability of termination for BL^{gp} bio-processes.

Theorem 5.4. *Let R be a RAM with program* $(1, I_1), \ldots, (m, I_m)$ *and initial state* $(j, k_1, \ldots, k_n)$. *Then the computation of the RAM R terminates if and only if the computation of the system* $Z = [\![(j, k_1, \ldots, k_n)]\!]^{gp}_R$ *terminates.*

Proof. ($\Rightarrow$) By hypothesis we have that the RAM R terminates. This means that the computation of R reaches, in a number l of steps, a terminated state $(j', k'_1, \ldots, k'_n)$, i.e. a state with a program counter greater than the number of instructions. The proof is by contradiction assuming that the system Z does not terminate, which means we have an infinite computation $Z \to Z_1 \to \cdots \to Z_i \to \cdots$. By Lemma 5.2 we have that there exists a well-formed system $Z' \in \mathcal{Z}^{gp}$ such that $Z \to^* Z'$ and $Z' \equiv_b [\![(j', k'_1, \ldots, k'_n)]\!]^{gp}_R$. By assumption, Z does not terminate and hence there exists Z'' such that $Z' \to Z''$. By Lemma 5.3 we have that there exists a computation $Z' \to Z'' \to Z_2 \to \cdots \to Z_l$ such that $Z_l = [\![(j'', k''_1, \ldots, k''_n)]\!]^{gp}_R$ and $(j', k'_1, \ldots, k'_n) \to_R (j'', k''_1, \ldots, k''_n)$. But this contradicts our hypothesis, which states that $(j', k'_1, \ldots, k'_n)$ is a terminated state and therefore the implication holds.
($\Leftarrow$) By hypothesis we have that the system Z terminates. This means that there exists a computation $Z \to^l Z'$ such that $Z' \not\to$. The proof is by contradiction assuming that the RAM R does not terminate. By applying Lemma 5.3 we have that $Z' \equiv_b [\![(j', k'_1, \ldots, k'_n)]\!]^{gp}_R$ and that $(j, k_1, \ldots, k_n) \to_R \cdots \to_R (j', k'_1, \ldots, k'_n)$. By assumption we have that $(j', k'_1, \ldots, k'_n) \to_R (j'', k''_1, \ldots, k''_n)$ and hence by Lemma 5.2 we have that there exists a well-formed system $Z'' \in \mathcal{Z}^{gp}$ such that $Z' \to^+ Z''$ and $Z'' \equiv_b [\![(j'', k''_1, \ldots, k''_m)]\!]^{gp}_R$. But this contradicts our hypothesis, which states that $Z' \not\to$ and therefore the implication holds. □

5.3. Encoding with BL^e

In BlenX we introduced a new concept of *event* [13]. Events can be considered as a reformulation of the f_{split} and f_{join} axioms present of the original version of Beta-binders [31]. Events are global rules of the system which can substitute single boxes or pairs of boxes.

We extend the *BL* language by introducing events; the syntax of *BL* is enriched in the following way:

$$\begin{aligned} cond &::= I[P] \mid I[P], I[P] \\ verb &::= \mathsf{split}(I[P], I[P]) \mid \mathsf{join}(I[P]) \\ event &::= \bullet \mid (cond)\ verb \\ E &::= event \mid E :: E \end{aligned}$$

The non-terminal symbol E generates a list of events (for more details see [36]). A list of events is always related to a bio-process B and each single event occurs only if its condition is satisfied on a set of one or more boxes composing B. A single *event* is the composition of a condition *cond* and an action *verb*. A system becomes a pair $Z = \langle B, E\rangle$, where B is a bio-process and E is the list of possible events enabled on the system.

Well-formedness conditions remain the same as the ones presented in Section 3, while a new definition of structural congruence over systems (Definition 5.5) and a new reduction semantics (Table 6) are introduced. In Table 6, a *join* event substitutes two boxes with a single one, while a *split* event substitutes a box with two boxes.

We denote with BL^e this extension and with $\mathcal{Z}^e$ the set of well-formed systems of BL^e.

Definition 5.5. Structural congruence over processes, denoted $\equiv_p$, is the smallest relation which satisfies the laws in Fig. 2 (group a), structural congruence over beta-processes, denoted $\equiv_b$, is the smallest relation which satisfies the laws in Fig. 2 (group b) and structural congruence over events, denoted $\equiv_e$, is the smallest relation which satisfies the laws in Fig. 3. Hence, two systems $Z = \langle B, E\rangle$ and $Z' = \langle B', E'\rangle$ are structurally congruent, indicated with $Z \equiv Z'$, only if $B \equiv_b B'$ and $E \equiv_e E'$.

Definition 5.6. The BL^e Transition System (TS^e) is referred as $(\mathcal{Z}^e, \to)$, where $\mathcal{Z}^e$ is the set of well-formed systems and $\to \subseteq \mathcal{Z}^e \times \mathcal{Z}^e$ is the transition relation.

As usual, $\to^+$ indicates the transitive closure and $\to^*$ the reflexive and transitive closure of $\to$. Moreover, $Z \to^k Z'$ with $k \geq 1$ indicates that Z' can be reached from Z with k transitions $Z \to^1 Z_1 \to^2 \cdots \to^{k-1} Z_{k-1} \to^k Z'$.

Consider a RAM R with program $(1, I_1), \ldots, (m, I_m)$ and state $(j, k_1, \ldots, k_n)$. The encoding of the RAM is:

$$[\![(j, k_1, \ldots, k_n)]\!]^e_R = \langle B, E\rangle$$

- $(B_0)\ split(B_1, B_2) \equiv_e (B'_0)\ split(B'_1, B'_2)$ if $B_0 \equiv_b B'_0$, $B_1 \equiv_b B'_1$ and $B_2 \equiv_b B'_2$
- $(B_0, B_1)\ join(B_2) \equiv_e (B'_0, B'_1)\ join(B'_2)$ if $B_0 \equiv_b B'_0$, $B_1 \equiv_b B'_1$ and $B_2 \equiv_b B'_2$
- $E::\bullet \equiv_e E$
- $E_0::E_1 \equiv_e E_1::E_0$
- $E_0::(E_1::E_2) \equiv_e (E_0::E_1)::E_2$

Fig. 3. Structural laws for events.

Table 6
Operational semantics of BL^e.

(intra)	$\langle I[\overline{x}\langle z\rangle.P_1 + M_1 \mid x(w).P_2 + M_2 \mid P_3], E\rangle \rightarrow \langle I[P_1 \mid P_2\{z/w\} \mid P_3], E\rangle$
(inter)	$\dfrac{P_1 \equiv_p \overline{x}\langle z\rangle.R_1 + M_1 \mid Q_1 \qquad P_2 \equiv_p y(w).R_2 + M_2 \mid Q_2}{\langle I_1[P_1] \parallel I_2[P_2], E\rangle \rightarrow \langle I_1[R_1 \mid Q_1] \parallel I_2[R_2\{z/w\} \mid Q_2], E\rangle}$ provided $\alpha(\Gamma, \Delta) > 0$ and $z \notin sub(I_1)$ and where $I_1 = \beta(x,\ \Delta)\, I_1^*$ and $I_2 = \beta(y,\ \Gamma)\, I_2^*$
(split)	$\langle I[P], E\rangle \rightarrow \langle I_0[P_0] \parallel I_1[P_1], E\rangle$ where $E = (I[P])\ \mathsf{split}(I_0[P_0], I_1[P_1]) :: E'$
(join)	$\langle I_0[P_0] \parallel I_1[P_1], E\rangle \rightarrow \langle I[P], E\rangle$ where $E = (I_0[P_0], I_1[P_1])\ \mathsf{join}(I[P]) :: E'$
(struct)	$\dfrac{Z_1 \equiv Z'_1 \qquad Z'_1 \rightarrow Z'_2 \qquad Z'_2 \equiv Z_2}{Z_1 \rightarrow Z_2}$
(redex)	$\dfrac{\langle B, E\rangle \rightarrow \langle B', E\rangle}{\langle B \parallel B_1, E\rangle \rightarrow \langle B' \parallel B_1, E\rangle}$

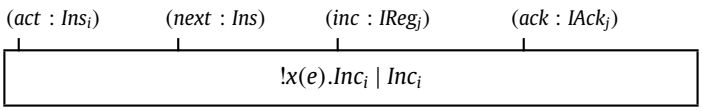
where,

$$B \triangleq_B Switch_j \parallel [\![(1, I_1)]\!]^e_R \parallel \cdots \parallel [\![(m, I_m)]\!]^e_R \parallel [\![r_1 = k_1]\!]^e_R \parallel \cdots \parallel [\![r_n = k_n]\!]^e_R$$
$$E \triangleq_E ZeroToOne_1 :: OneToZero_1 :: \cdots :: ZeroToOne_n :: OneToZero_n$$

The differences w.r.t. the encoding presented in Section 5.2 concern the encoding of instructions and registers. The encoding of the instruction $(i, Incr(r_j))$ is

$$\begin{aligned}[\![(i, Incr(r_j))]\!] = {} & \beta(act : Ins_i)\beta(next : Ins)\beta(inc : IReg_j) \\ & \beta(ack : IAck_j)[\ !x(e).Inc_i \mid Inc_i\ ] \\ Inc_i \triangleq_P {} & act(e).\overline{inc}\langle e\rangle.ack(e).\overline{x}\langle e\rangle.\overline{next}\langle ins_{i+1}\rangle.nil\end{aligned}$$

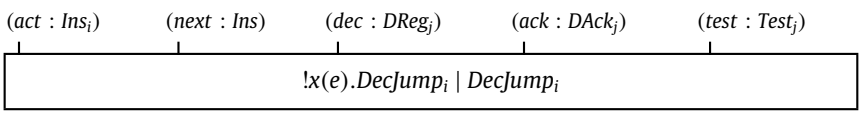
and its graphical representation is



The encoding of the instruction $(i, DecJump(r_j, s))$ is

$$\begin{aligned}[\![(i, DecJump(r_j, s))]\!] = {} & \beta(act : Ins_i)\beta(next : Ins)\beta(dec : DReg_j) \\ & \beta(ack : DAck_j)\beta(test : Test_j) \\ & [\ !x(e).DecJump_i \mid DecJump_i\ ] \\ DecJump_i \triangleq_P {} & act(e).test(t).(\overline{t}\langle e\rangle.nil \mid (yes(e).Dec_i + \\ & no(e).Jump_i)) \\ Dec_i \triangleq_P {} & \overline{dec}\langle e\rangle.ack(e).\overline{x}\langle e\rangle.\overline{next}\langle ins_{i+1}\rangle.nil \\ Jump_i \triangleq_P {} & \overline{x}\langle e\rangle.\overline{next}\langle ins_s\rangle.nil\end{aligned}$$

and its graphical representation is



The encoding of a register r_j with content l is defined as follows

$$[\![r_j = l]\!] = \begin{cases} B_j^0 & \text{if } l = 0 \\ B_j^{test} \parallel B_j^l & otherwise \end{cases}$$

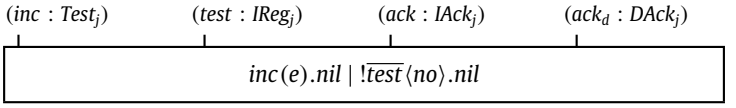
where the boxes B_j^0, B_j^{test}, B_j^l are defined as follows

$$\begin{aligned} B_j^0 &\triangleq_B \beta(test : Test_j)\beta(inc : IReg_j)\beta(ack_i : IAck_j) \\ &\quad \beta(ack_d : DAck_j)[\ \underline{inc}(e).nil \mid !\overline{test}\langle no\rangle.nil\ ] \\ B_j^{test} &\triangleq_B \beta(test : Test_j)[\ !\overline{test}\langle yes\rangle.nil\ ] \\ B_j^l &\triangleq_B \beta(inc : IReg_j)\beta(dec : DReg_j)\beta(ack_i : IAck_j) \\ &\quad \beta(ack_d : DAck_j)[\ !inc(e).Increment \mid AckL_l \mid DecL_l\ ] \end{aligned}$$

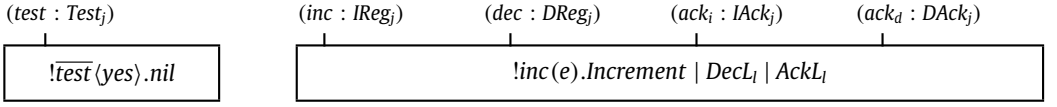
where,

$$\begin{aligned} DecL_l &\triangleq_P \underbrace{Decrement \mid \cdots \mid Decrement}_{l} \\ AckL_l &\triangleq_P \underbrace{\overline{ack_d}\langle e\rangle.nil \mid \cdots \mid \overline{ack_d}\langle e\rangle.nil}_{l-1} \\ Increment &\triangleq_P \overline{ack_d}\langle e\rangle.nil \mid \overline{ack_i}\langle e\rangle.Decrement \\ Decrement &\triangleq_P dec(e).nil \end{aligned}$$

If $l = 1$ the process $AckList_l \equiv_p nil$. Moreover, the processes *Increment* and *Decrement* are equal for the encoding of all the registers. The graphical representation of the bio-process B_j^0 is



and the graphical representation of the bio-process $B_l^{test} \parallel B_l^{value}$ is



The events $ZeroToOne_j$ and $OneToZero_j$, which encode the ability to change the state of a register l from the representation of 0 and the representation of 1 and vice versa, are defined as follows

$$\begin{aligned} ZeroToOne_j &\triangleq_E (B_j^{split0})\ \mathsf{split}(B_j^{test}, B_j^{split1}) \\ OneToZero_j &\triangleq_E (B_j^{test}, B_j^{join1})\ \mathsf{join}(B_j^{join0}) \end{aligned}$$

where,

$$\begin{aligned} B_j^{split0} &\triangleq_B \beta(test : Test_j)\beta(inc : IReg_j)\beta(ack_i : IAck_j)\beta(ack_d : DAck_j) \\ &\quad [\ !\overline{test}\langle no\rangle.nil\ ] \\ B_j^{join0} &\triangleq_B \beta(test : Test_j)\beta(inc : IReg_j)\beta(ack_i : IAck_j)\beta(ack_d : DAck_j) \\ &\quad [\ \overline{ack_d}\langle e\rangle.inc(e).nil \mid !\overline{test}\langle no\rangle.nil\ ] \\ B_j^{split1} &\triangleq_B \beta(inc : IReg_j)\beta(dec : DReg_j)\beta(ack_i : IAck_j)\beta(ack_d : DAck_j) \\ &\quad [\ !inc(e).Increment \mid \overline{ack_i}\langle e\rangle.Decrement\ ] \\ B_j^{join1} &\triangleq_B \beta(inc : IReg_j)\beta(dec : DReg_j)\beta(ack_i : IAck_j)\beta(ack_d : DAck_j) \\ &\quad [\ !inc(e).Increment\ ] \end{aligned}$$

The function α is defined in the following way

$$\alpha(Type_1, Type_2) = \begin{cases} 1 & \text{if } Type_1 = Type_2 \\ 0 & otherwise. \end{cases}$$

The encoding produces a system $Z = \langle B, E\rangle$ in $\mathcal{Z}^e$. The bio-process B is a parallel composition of a switching box, which controls the activation of the instructions sequence, m boxes encoding instructions and n boxes encoding registers; the two

types of instructions are encoded in different ways, but in both cases the encoding box is activated by performing an inter-communication with the box $Switch_j$. The list of events E contains a couple of events for each register which controls the transformation of a register with content 0 to a register with content 1, and vice versa.

The modelling of the register r_j depends on its content. If the content of the register is 0, then the box B_j^0 is used; if the content of the register is greater than zero, then bio-process $B_j^{test} \parallel B_j^l$ is used.

The box encoding the instruction $(i, Incr(r_j))$, after its activation, consumes an inter-communication with the box encoding the register r_j (through the interfaces of type $IReg_j$), representing a request for its increment; then the instruction box waits for another inter-communication (through the interfaces of type $IAck_j$) with the register box; a kind of acknowledgment indicating that the increment has been executed. Finally, after the acknowledgment, the box replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of type Ins) for the activation of instruction $i + 1$. The behaviour of the register box depends on its content.

If the content is 0, after consuming the increment inter-communication, the box becomes structurally congruent to the box B_j^{split0}, causing the activation of event $ZeroToOne_j$. This event substitutes B_j^{split0} with the bio-process $B_j^{test} \parallel B_j^{split1}$. After consuming the acknowledgment inter-communication on the channel ack_i, the box becomes structurally congruent to the box B_j^1, indicating that the register has been correctly incremented. Notice that when the event $ZeroToOne_j$ is enabled no other actions in the system are enabled. This guarantees that the register transformation is achieved between the request of the instruction and the acknowledgment of the register.

If the content is greater than zero, the increment inter-communication enables the internal replication of the process $Increment$, representing the addition of 1 on the content of the register. The corresponding acknowledgment is performed after the replication, consuming the acknowledgment inter-communication on the channel ack_i.

The box encoding the instruction $(i, DecJump(r_j, s))$, after its activation, consumes first an inter-communication with the box encoding for the register r_j, in order to test its content (through the interfaces of type $Test_j$). In particular, the instruction box receives a name yes if the content of the register r_j is greater than zero, receives the name no otherwise. With the choice operator two alternative behaviours are encoded, depending on the result of the testing communication. In case of yes name reception, the instruction box consumes an inter-communication with the r_j register box, representing a request for its decrement (through the interface of type $DReg_j$), then waits for an acknowledgment indicating that the decrement has been executed (through the interfaces of types $DAck_j$) and finally replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of type Ins) for the activation of instruction $i + 1$. Instead, in case of no name reception, the box simply replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of type Ins) for the activation of instruction s. The behaviour of the register box in the case of decrement depends on its content.

If a decrement inter-communication is consumed by a box representing a register with content 1, then the box becomes structurally congruent to the box B_j^{join1}. This box activates the event $OneToZero_j$, which substitutes the bio-process $B_j^{test} \parallel B_j^{join1}$ with the box B_j^{join0}. After consuming the decrement inter-communication on the channel ack_d, the box becomes structurally congruent to the box B_j^0, indicating that the register has been correctly decremented. Notice that, also in this case, when the event $OneToZero_j$ is enabled no other actions in the system are enabled.

If a decrement inter-communication is consumed by a box representing a register with content greater than zero, then the acknowledgment inter-communication is then consumed, deleting an instance of the parallel processes composing the $AckList_l$ process and hence representing the decrement of 1.

A formal proof of the encoding correctness follows.

Lemma 5.7. *Let R be a RAM with program $(1, I_1), \ldots, (m, I_m)$ and state $(j, k_1, \ldots, k_n)$. If $(j, k_1, \ldots, k_n) \rightarrow_R (j', k'_1, \ldots, k'_n)$, then there exists a system Z such that $[\![(j, k_1, \ldots, k_n)]\!]_R^e \rightarrow^+ Z$ and $Z \equiv [\![(j', k'_1, \ldots, k'_n)]\!]_R^e$.*

Proof. The proof is by case analysis. There are five cases: (i) Instruction $I_j = DecJump(r_l, s)$ and r_l value greater than one; (ii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to one; (iii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to zero; (iv) Instruction $I_j = Incr(r_l)$ and r_l value greater than zero; (v) Instruction $I_j = Incr(r_l)$ and r_l value equal to zero. We prove only case (ii), because the other cases are similar.

(ii) We consider the computation of the bio-process $[\![(j, k_1, \ldots, k_n)]\!]_R^e$. As in Lemma 5.2 an inter-communication between the component $Switch_j$ and the component $[\![(j, DecJump(r_l, s))]\!]_R^e$ is the first consumed action. The two boxes synchronize on output $\overline{i_j}\langle e\rangle$ and input $act(e)$ through their binders of type Ins_j. This cause the activation of the $[\![(j, DecJump(r_l, s))]\!]_R^e$ component. Also in this case, after the inter-communication the components codifying for the other instructions, the registers and the switching box are blocked.

We have that the content of the register is 1 and hence it is encoded by the bio-process $B_l^{test} \parallel B_l^1$. The activation of the $[\![(j, DecJump(r_l, s))]\!]_R^e$ box causes the consumption of an inter-communication between the box B_l^{test} and the instruction box; since the content of the register is 1, then the instruction box receives the name yes, indicating that the content of the register is greater than zero. After the inter-communication the instruction box performs another intra-communication with the register box, which causes the activation of the process Dec_j. This process synchronizes with the box B_k^1, consumes an inter-communication on output $\overline{dec}\langle e\rangle$ through interface of type $DReg_l$ and remains blocked on input $ack(e)$; after that inter-communication, the box B_l^1 becomes structurally congruent to the box B_l^{join1} and the event $OneToZero_l$ becomes active. The

execution of the event substitutes the bio-process $B_l^{test} \parallel B_l^{join1}$ with the box B_l^{join0}, which consumes an intra-communication with the instruction box (on interfaces of type $DAck_l$) and becomes structurally congruent to $[\![r_l = 0]\!]_R^e$. Moreover, the last inter-communication unblocks the instruction box, which consumes an intra-communication on channel x, replicating its internal machinery, and enables the process $\overline{next}\langle ins_{j+1}\rangle.nil$. This produces a synchronization between the instruction box and the switching box. Indeed, the boxes consume an inter-communication on output $\overline{next}\langle ins_{j+1}\rangle$ and input $ins(type)$ through interfaces of type Ins.

The instruction box is now returned in its form $[\![(j, DecJump(r_l, s))]\!]_R^e$, while in the switching box the process

$$\left(\overline{ins_{j+1}}\langle e\rangle.nil \mid \left(\sum_{o=1}^{m} ins_o(e).\bar{x}\langle e\rangle.\bar{i_o}\langle e\rangle.nil\right)\right)$$

is enabled. An intra-communication on channel ins_{j+1} is consumed, the internal machinery is replicated with an intra-communication on channel x and the switching box is now structurally congruent to the box $Switch_{j+1}$. □

Lemma 5.8. *Let R be a RAM with program $(1, I_1), \ldots, (m, I_m)$ and state $(j, k_1, \ldots, k_n)$. If the system $Z = [\![(j, k_1, \ldots, k_n)]\!]_R^e$ can produce a transition $Z \to Z_1$, then there exists a computation $Z \to Z_1 \to Z_2 \to \cdots \to Z_l$ such that $Z_l = [\![(j', k'_1, \ldots, k'_n)]\!]_R^e$ and $(j, k_1, \ldots, k_n) \to_R (j', k'_1, \ldots, k'_n)$.*

Proof. Consider the structure of the bio-process $Z = [\![(j, k_1, \ldots, k_n)]\!]_R^{gp}$. As in Lemma 5.3, if the bio-process Z can perform a first step $Z \to Z_1$, this corresponds to an inter-communication between the box $Switch_j$ and the box encoding for the instruction (j, I_j), representing the activation of the instruction box. By encoding definition this means that the instruction (j, I_j) exists; hence the instruction can be executed in the state $(j, k_1, \ldots, k_n)$ of the RAM R, generating a new state $(j', k'_1, \ldots, k'_n)$.

The proof is by case analysis. There are five cases: (i) Instruction $I_j = DecJump(r_l, s)$ and r_l value greater than one; (ii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to one; (iii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to zero; (iv) Instruction $I_j = Incr(r_l)$ and r_l value greater than zero; (v) Instruction $I_j = Incr(r_l)$ and r_l value equal to zero. In all the cases, it is possible to show that from the moment in which the switch activates an instruction till the moment in which the switch is able to activate a new instruction, the computation proceeds deterministically (up to structural congruence $\equiv$). The encoding is hence deterministic up to structural congruence. We prove only case (v), because the other cases can be proved similarly.

(v) By encoding definition we have that the structure of the instruction box is

$$[\![(j, Incr(r_l))]\!]_R^e = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l)\beta(ack : IAck_j)\\ [!x(e).Inc_j \mid act(e).\overline{inc}\langle e\rangle.ack(e).\bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

This box is the only one able to synchronize with the box $Switch_j$ for an inter-communication through interfaces of type Ins_j. After the communication we have that in Z_1 the box encoding the instruction j becomes structurally congruent to

$$B' = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l)\beta(ack : IAck_j)\\ [!x(e).Inc_j \mid \overline{inc}\langle e\rangle.ack(e).\bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

Now, the only possible action $Z_1 \to Z_2$ is the inter-communication between the box B' and the box $[\![r_l = 0]\!]_R^e$ through their interfaces of type $IReg_l$ on output $\overline{inc}\langle e\rangle$ and input $inc(e)$, respectively. After the inter-communication the instruction box remains blocked on input $ack(e)$ over interface of type $IAck_l$ and the box encoding for the register r_l becomes structurally congruent to the box B^{split0}; the event $ZeroToOne_j$ is now active. The execution of the event, which correspond to the action $Z_2 \to Z_3$, substitutes in Z_2 the box B_l^{split0} with the bio-process $B_l^{test} \parallel B_l^{split1}$. At this point, the action $Z_3 \to Z_4$ is an inter-communication between the box B_l^{split1} and the instruction box; the register box becomes structurally congruent to $[\![r_l = 1]\!]_R^e$, while the instruction box is unblocked and structurally congruent to

$$B'' = \beta(act : Ins_j)\beta(next : Ins)\beta(inc : IReg_l)\beta(ack : IAck_j)\\ [!x(e).Inc_j \mid \bar{x}\langle e\rangle.\overline{next}\langle ins_{j+1}\rangle.nil]$$

Now, the action $Z_4 \to Z_5$ is the intra-communication of B'' on channel x which becomes a box B''' with internal structure $!x(e).Inc_j \mid \overline{next}\langle ins_{j+1}\rangle.nil$, and the action $Z_5 \to Z_6$ is the inter-communication between the box B''' and the switching box. After the inter-communication, the instruction box returns in its initial form $[\![(j, Incr(r_l))]\!]_R^e$ and the switching box starts a sequence of intra-communications which produces a box structurally congruent to $Switch_{j+1}$ and representing the sequence of actions $Z_6 \to Z_7 \to Z_8$. It is easy to see that Z_8 is congruent to $[\![(j + 1, k_1, \ldots, k_{l-1}, 1, k_{l+1}, \ldots, k_n)]\!]_R^e$. □

Lemmas 5.7 and 5.8 can now be used for proving the undecidability of termination for BL^e bio-processes.

Theorem 5.9. *Let R be a RAM with program $(1, I_1), \ldots, (m, I_m)$ and initial state $(j, k_1, \ldots, k_n)$. The computation of the RAM R terminates if and only if the computation of the system $Z = [\![(j, k_1, \ldots, k_n)]\!]_R^e$ terminates.*

Proof. The theorem can be proved similarly to Theorem 5.4 and by using Lemmas 5.7 and 5.8. □

6. Conclusion

In this paper we investigated the computational power of the nondeterministic version of BlenX, a language based on Beta-binders.

We first considered a core subset of BlenX denoted with *BL*, showing that termination for *BL* is decidable. The *BL* subset is constructed using only primitives for communication. The *BL* subset is then enriched with immediate action (i.e. global priorities are added) and the obtained subset, denoted with BL^{gp}, is shown to be Turing equivalent by providing an encoding of Random Access Machines into BL^{gp}. Another undecidability result is then given; the *BL* subset is enriched with events (i.e. join and split events are added) and the obtained subset, denoted with BL^{e}, is shown to be Turing equivalent by providing an encoding of Random Access Machines into BL^{e}.

The recent paper by Cardelli and Zavattaro [8] suggests us that we can obtain Turing equivalence also by enriching the *BL* subset with the BlenX primitives for complexes management and the split event; we plan to investigate this aspect in the near future.

Although this work allows us to conclude that BlenX is a Turing equivalent language, we think that the obtained results represent also an interesting investigation into how the addition of global priorities affects the expressive power of a language and on the role that some high-powered features like restriction operator play in Turing equivalence encodings. Moreover, we think that these results are a basis for further investigations and for a better understanding of how different primitives and operators can be added, deleted or combined to obtain classes of languages with different computational power.

Finally, future work on the expressivity of the fully stochastic version of BlenX is also planned.

Acknowledgments

We would like to thank Gianluigi Zavattaro for the precious comments and for pointing out an error in a preliminary version of the paper. Moreover, we would like to thank Matteo Cavaliere and Paola Quaglia for the fruitful discussions about the topic and Alida Palmisano for reviewing the preliminary versions of the document. Finally, we thank the anonymous referees for their useful comments and suggestions.

References

[1] J. Aranda, F.D. Valencia, C. Versari, On the expressive power of restriction and priorities in ccs with replication, in: FOSSACS, in: Lecture Notes in Comput. Sci., vol. 5504, Springer, 2009, pp. 242–256.
[2] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, Syntax and defining equations for an interruptmechanism in process algebra, FUNINF IX (2) (1986) 127–168.
[3] N. Busi, M. Gabbrielli, G. Zavattaro, On the expressive power of recursion, replication, and iteration in process calculi, Math. Struct. Comput. Sci. (in press).
[4] N. Busi, R. Gorrieri, On the computational power of Brane calculi, Theory Comput. Syst. Biol. (2006) 16–43.
[5] N. Busi, G. Zavattaro, On the expressive power of movement and restriction in pure mobile ambients, Theoret. Comput. Sci. 322 (3) (2004) 477–515.
[6] J. Camilleri, G. Winskel, CCS with priority choice, Inform. Comput. 116 (1) (1995) 26–37.
[7] L. Cardelli, Brane calculi, in: CMSB, 2004, pp. 257–278.
[8] L. Cardelli, G. Zavattaro, On the computational power of biochemistry, in: AB '08: Proceedings of the 3rd International Conference on Algebraic Biology, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 65–80.
[9] R. Cleaveland, M. Hennessy, Priorities in process algebras, Inform. Comput. 87 (1–2) (1990) 58–77.
[10] P.l. Curien, V. Danos, J. Krivine, M. Zhang, Computational self-assembly, Theoret. Comput. Sci. 404 (1–2) (2008) 61–75.
[11] V. Danos, C. Laneve, Formal molecular biology, Theoret. Comput. Sci. 325 (1) (2004) 69–110.
[12] P. Degano, D. Prandi, C. Priami, P. Quaglia, Beta-binders for biological quantitative experiments, ENTCS 164 (3) (2006) 101–117.
[13] L. Dematté, C. Priami, A. Romanel, BetaWB: Modelling and simulating biological processes, in: SCSC: Proceedings of the 2007 Summer Computer Simulation Conference, Society for Computer Simulation International, San Diego, CA, USA, 2007, pp. 777–784.
[14] L. Dematté, C. Priami, A. Romanel, The Beta Workbench: A computational tool to study the dynamics of biological systems, Brief. Bioinform. 9 (5) (2008) 437–449.
[15] A. Finkel, P. Schnoebelen, Well-structured transition systems everywhere! Theoret. Comput. Sci. 256 (1–2) (2001) 63–92.
[16] S. Gilmore, J. Hillston, The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling, in: LNCS., vol. 794, Springer-Verlag, 1994, pp. 353–368.
[17] D. Harel, A visual formalism for complex systems, Sci. Comput. Program. 8 (2002) 231–274.
[18] G. Higman, Ordering by divisibility in abstract algebras, Proc. London Math. Soc. (3) 2 (1952) 326–336.
[19] Jane Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, New York, NY, USA, 1996.
[20] S.A. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, J. Theoret. Biol. 22 (1969) 437–467.
[21] H. Kitano (Ed.), Foundations of Systems Biology, MIT Press, Cambridge, MA, USA, November 2001.
[22] S. Maffeis, I. Phillips, On the computational strength of pure ambient calculi, Theoret. Comput. Sci. 330 (3) (2005) 501–551.
[23] R. Milner, Communication and Concurrency, Prentice-Hall, Inc., 1989.
[24] R. Milner, Function as processes, MSCS 2 (1992) 119–141.
[25] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
[26] G. Paun, Membrane Computing: An Introduction, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
[27] C.A. Petri, Kommunikation mit Automaten, Ph.D. Thesis, University of Bonn, 1962.
[28] A. Phillips, L. Cardelli, A correct abstract machine for the stochastic pi-calculus, in: Bioconcur'04, August 2004.
[29] I. Phillips, CCS with Priority Guards, in: LNCS, vol. 2154, 2001.
[30] G.D. Plotkin, A structural approach to operational semantics, J. Logic Algebraic Program. (2004) 17–139.
[31] C. Priami, P. Quaglia, Beta binders for biological interactions, in: V. Danos, V. Schächter (Eds.), CMSB, in: LNCS, vol. 3082, Springer, 2005.
[32] C. Priami, P. Quaglia, Operational patterns in beta-binders, Theory Comput. Syst. Biol. 1 (2005) 50–65.
[33] 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.

[34] A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro, Bioambients: An abstraction for biological compartments, Theoret. Comput. Sci. 325 (1) (2004) 141–167.
[35] A. Regev, E. Shapiro, Cells as computation, Nature 419 (September) (2002).
[36] A. Romanel, L. Dematté, C. Priami, The beta workbench, Technical Report TR-03-2007, CoSBi, 2007.
[37] A. Romanel, C. Priami, On the decidability and complexity of the structural congruence for beta-binders, Theoret. Comput. Sci. 404 (1–2) (2008) 156–169.
[38] D. Sangiorgi, D. Walker, The π-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[39] J.C. Shepherdson, H.E. Sturgis, Computability of recursive functions, J. ACM 10 (2) (1963) 217–255.
[40] C. Versari, N. Busi, R. Gorrieri, On the expressive power of global and local priority in process calculi, in: CONCUR, in: Lecture Notes in Comput. Sci., vol. 4703, Springer, 2007, pp. 241–255.