On the Computational Power of BlenX

Alessandro Romanel
CoSBi and Università di Trento
romanel@cosbi.eu

Corrado Priami
CoSBi and Università di Trento
priami@cosbi.eu
Abstract

We present some decidability and undecidability results for subsets of the \textsf{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 \textsf{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 \textsf{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 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 $\pi$-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 non deterministic 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 Sec. 2 a brief introduction describing the syntax and the main features of BlenX is presented. In Sec. 3 the syntax and semantics of $BL$ is presented. In Sec. 4 termination of the $BL$ subset is proved to be decidable. In Sec. 5 encodings of RAMs into the $BL^{gp}$ and $BL^e$ subsets are given, along with their proofs of correctness and in Sec. 6
Figure 1: BlenX model for a simple biochemical reaction

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:

\[
x_1 : \Delta_1 \ldots x_n : \Delta_n \\
M
\]

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 $\Delta_i$ 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:

\[
A + B \xrightarrow{k_1} AB \xrightarrow{k_2} \text{CD} \xrightarrow{k_3} C + D
\]

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 $\text{CD}$ (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 $\bar{B}_A$ and $\bar{B}_B$ for molecules $A$ and $B$ complex into box $\bar{B}_{AB}$, if the types $\Delta_A$ and $\Delta_B$ 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 $\Delta_A$ and $\Delta_B$ are modified into $\Delta_C$ and $\Delta_D$, respectively. Finally the complex unbinds releasing $\bar{B}_C$ and $\bar{B}_D$. 

3
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 $\xi$. 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 $\xi$ 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 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 $\Delta, \Gamma, \Delta', \Delta_0, ...$) such that $T \cap \mathcal{N} = \emptyset$. The syntax of $BL$ is defined in the following way:

$$
B \ ::= \ Nil \mid I[P] \mid B\|B
$$

$$
I \ ::= \ beta(x, \Delta) \mid beta(x, \Delta)I
$$

$$
P \ ::= \ P|P \mid !\pi.P \mid M
$$

$$
M \ ::= \ nil \mid \pi.P \mid M + M
$$

$$
\pi \ ::= \ x(y) \mid \pi(y)
$$
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 $\text{Nil}$ 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 $\text{box}_{\#}(B)$ 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 $\Delta$ 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 $I, I_1, I_2, \cdots, I', \cdots$. We write $I = I_1I_2$ to mean that $I$ is the interface given by the juxtaposition of $I_1$ and $I_2$. Also, the metavariables $I^*, I_1^*, I_2^*, \cdots$ 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 $B$ and $I$ we denote the set of all the possible bio-processes and interfaces, respectively.

**Definition 3.1.** The functions $\text{sub} : I \rightarrow 2^N$ and $\text{sub}_t : B \rightarrow 2^N$ are defined as follows

$$\text{sub}(\beta(x, \Gamma)) = \{x\}$$

$$\text{sub}(\beta(x, \Gamma)I) = \{x\} \cup \text{sub}(I)$$

$$\text{sub}_t(I[P]) = \text{sub}(I)$$

$$\text{sub}_t(B \parallel B') = \text{sub}_t(B) \cup \text{sub}_t(B')$$

Function $\text{sub}$ returns the set of subjects present in an interface, while function $\text{sub}_t$ 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 $Z$ the set of all well-formed systems. In particular, for the BL subset we have that $Z \subset B$.

Processes generated by the non-terminal symbol $P$ are referred as processes and the set of all possible processes is denoted by $P$. 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
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 Tab. 1 through the function $fn$.

\begin{align*}
fn(\text{Nil}) &= fn(\text{nil}) = fn(\beta(x, \Delta)) = fn(\beta(x, \Delta)I) = \emptyset \\
fn(I[P]) &= fn(P) \setminus \text{sub}(I) \\
fn(B \mid B') &= fn(B) \cup fn(B') \\
fn(P \mid 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(\overline{x<y}).P) &= \{x, y\} \cup fn(P)
\end{align*}

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

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

**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 $\pi$. 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 $x$ ($\overline{x<y}$). 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 $\overline{x<y}$) 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 behavior 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 $\alpha$ 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

\[
\begin{align*}
\text{(intra)} & \quad I[\overline{p}(z), P_1 + M_1 | x(w), P_2 + M_2 | P_3] \rightarrow I[P_1 | P_2{\overline{y}} | P_3] \\
\text{(inter)} & \quad P_1 \equiv_p \overline{p}(z), R_1 + M_1 | Q_1 \quad P_2 \equiv_p y(w), R_2 + M_2 | Q_2 \\
& \quad \frac{I_1[P_1] \parallel I_2[P_2] \rightarrow I_1[R_1 | Q_1] \parallel I_2[R_2{\overline{y}} | Q_2]}{	ext{where } I_1 = \beta(x, \Delta) I_1' \text{ and } I_2 = \beta(y, \Gamma) I_2'} \\
& \quad \text{and provided that } \alpha(\Gamma, \Delta) > 0 \text{ and } z \not\in \text{sub}(I_1) \cup \text{sub}(I_2) \\
\text{(struct)} & \quad B_1 \equiv_b B_1' \quad B_1' \rightarrow B_2' \quad B_2' \equiv_b B_2 \\
& \quad \frac{B_1 \rightarrow B_2'}{B_1 \parallel B_1' \rightarrow B_2' \parallel B_1} \\
\text{(redex)} & \quad B \rightarrow B'
\end{align*}
\]

Table 2: Reduction semantics of BL.
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, 4, 5] in which decidability results for $\pi$-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 $Z_s$ 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 Sec. 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 $\tau$ transitions according to the new labelled transition semantics.

Then, we define a quasi-ordering $\preceq_b$ on bio-processes which is strongly compatible with $\tau \rightarrow$, we show that the relation $\preceq_b$ is a well-quasi-ordering and finally we prove that the termination of bio-processes in $Z_s$ 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 $\text{Succ}(s)$. $TS$ is finite branching if $\text{Succ}(s)$ 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 $\text{Succ}$. 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 $t = t_1...t_m$ and $u = u_1...u_n$. We have that $t \leq^* u$ iff there exists an injection $f$ from $\{1,...,m\}$ to $\{1,...,n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1,...,m$.

**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 $S$.

### 4.2 A labelled transition semantics for $BL$

In this section we define a labelled transition semantics for $BL$ (Tab. 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 $\pi$-calculus, and hence some results there reported can be reused.

We use the meta-variable $\theta$ to range over $xy, \pi y, \Delta xy, \Delta \pi y$, and $\tau$. The set of names, $n(\theta)$, of $\theta$ is $\text{fn}(\theta) \cup \text{bn}(\theta)$. In Table 3 terminology and notation for labels are reported.

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>kind</th>
<th>$\text{fn}(\theta)$</th>
<th>$\text{bn}(\theta)$</th>
<th>$n(\theta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$xy$</td>
<td>process input</td>
<td>${x, y}$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$\pi y$</td>
<td>process output</td>
<td>${x, y}$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$\Delta xy$</td>
<td>box input over $\Delta$</td>
<td>${x, y}$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$\Delta \pi y$</td>
<td>box output over $\Delta$</td>
<td>${x, y}$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$\tau$</td>
<td>internal</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Table 3: Terminology and notation for action labels.

Notice that the semantics we define is not equivalent to the one presented in Section 2, because of the absence of rules for managing $\alpha$-conversion. We do not explicitly consider $\alpha$-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 $\tau \rightarrow$ 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 behaviors are lost when we get rid of structural congruence. Suppose $\alpha(\Delta, \Delta) > 0$ and consider the following bio-process:

$$B = \beta(x, \Delta)[x(y).\text{nil} \parallel \beta(x, \Delta) \beta(y, \Gamma)[x(z).z(k).\text{nil} \parallel y(z).\text{nil}]]$$

To avoid captures in inter-communications, rule (inter) in Tab. 2 requires $y \notin \text{sub}(\beta(x, \Delta)) \cup \text{sub}(\beta(x, \Delta) \beta(y, \Gamma))$, 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:

$$B' = \beta(x, \Delta)[x(y).\text{nil} \parallel \beta(x, \Delta) \beta(n, \Gamma)[x(z).z(k).\text{nil} \parallel n(z).\text{nil}]]$$

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 $\text{fn}(B) \cap \text{sub}_b(B) = \emptyset$.

We denote with $Z_s \subset Z$ the set of safe well-formed bio-processes.

**Lemma 4.9.** Let $B \in Z$. There exists $B' \in Z_s$ such that $B \equiv_b B'$.

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

We denote with $\text{safe}(B) \subset Z_s$ the set of safe bio-processes structurally congruent to a generic bio-process $B \in Z$. Given a bio-process $B \in Z$, 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 $\text{fn}(B)$ 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
<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pi.in)</td>
<td>( x(w).P \xrightarrow{\tau y} P[y/w] )</td>
</tr>
<tr>
<td>(pi.out)</td>
<td>( \pi(y).P \xrightarrow{\tau y} P )</td>
</tr>
<tr>
<td>(rep.in)</td>
<td>( !x(w).P \xrightarrow{\tau y} P[y/w] \mid !x(w).P )</td>
</tr>
<tr>
<td>(rep.out)</td>
<td>( \Pi(y).P \xrightarrow{\tau y} P \mid \Pi(y).P )</td>
</tr>
<tr>
<td>(l.pi.sum)</td>
<td>( M_0 \xrightarrow{\theta} M_0' )</td>
</tr>
<tr>
<td>(l.pi.par)</td>
<td>( P_0 \mid P_1 \xrightarrow{\theta} P_0' \mid P_1 )</td>
</tr>
<tr>
<td>(l.intra)</td>
<td>( P \xrightarrow{\tau} P' \quad Q \xrightarrow{\tau} Q' )</td>
</tr>
<tr>
<td>(bio.out)</td>
<td>( \beta(x, \Delta)I^<em>[P] \xrightarrow{\Delta \tau y} \beta(x, \Delta)I^</em>[P'] ) ( y \notin \text{sub}(I^*) \cup {x} )</td>
</tr>
<tr>
<td>(bio.in)</td>
<td>( \beta(x, \Delta)I^<em>[P] \xrightarrow{\Delta \tau y} \beta(x, \Delta)I^</em>[P'] ) ( y \notin \text{sub}(I^*) \cup {x} )</td>
</tr>
<tr>
<td>(pi.bio)</td>
<td>( I[P] \xrightarrow{\tau} I[P] )</td>
</tr>
<tr>
<td>(l.bio.par)</td>
<td>( B_0 \xrightarrow{\theta} B_0' )</td>
</tr>
<tr>
<td>(l.inter)</td>
<td>( B_0 \parallel B_1 \xrightarrow{\tau} B_0' \parallel B_1' ) ( B_0 \xrightarrow{\Delta \tau z} B_0' B_1 \xrightarrow{\Gamma \tau z} B_1' ) ( \alpha(\Delta, \Gamma) &gt; 0 )</td>
</tr>
</tbody>
</table>

Table 4: Labelled transition semantics of BL (we omit the symmetric rules \( r_{\pi \text{sum}}, r_{\pi \text{par}}, r_{\text{intra}}, r_{\text{bio \ par}} \) and \( r_{\text{inter}} \)).
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 $(Z_s, \xrightarrow{\theta})$ is finite branching. In order to do this we first have to show that the safe property of bio-processes is preserved over $\tau$ 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 $\theta$.

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

(a) if $\theta = \overline{x}y$ 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. \qed

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

(a) if $\theta = \Delta \overline{x}y$ 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 \texttt{bio\_out}) We have $B = (x, \Delta)I^*[P]$ and $B' = (x, \Delta)I^*[P']$. By definition we have $fn(B) = fn(P) \setminus (\text{sub}(I^*) \cup \{x\})$ and $fn(B') = fn(P') \setminus (\text{sub}(I^*) \cup \{x\})$. By hypothesis we have $P \xrightarrow{\theta} P'$ and hence, by Lemma 4.10, $x, y \in fn(P)$ and $fn(P') \subseteq fn(P) \setminus (\text{sub}(I^*) \cup \{x\}) = fn(P) \setminus (\text{sub}(I^*) \cup \{x\})$, which means $fn(B') \subseteq fn(B)$. Moreover, since $y \in fn(P)$ and $y \notin (\text{sub}(I^*) \cup \{x\})$ we have $y \notin fn(B)$.

(Case \texttt{l\_bio\_par}) By definition $fn(B_0 || B_1) = fn(B_0) \cup fn(B_1)$. By inductive hypothesis we have $y \notin fn(B_0)$ 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 $y \notin fn(B_0)$, we have $y \notin fn(B)$.

(b)

(Case \texttt{bio\_in}) We have $B = (x, \Delta)I^*[P]$ and $B' = (x, \Delta)I^*[P']$. By definition we have $fn(B) = fn(P) \setminus (\text{sub}(I^*) \cup \{x\})$ and $fn(B') = fn(P') \setminus (\text{sub}(I^*) \cup \{x\})$...
\{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 literal) 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 \(B \in Z_s\). Then \(B \xrightarrow{\tau} B'\) implies \(B' \in Z_s\).

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

Now, we recall some results on image-finiteness of \(\pi\)-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 \(P \xrightarrow{xy} P'\) for some \(z\) and \(P'\).
2. There are only finitely many triples \(x, y, P'\) such that \(P \xrightarrow{\tau y} P'\).

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 of \(P\).
(Induction base) If \(P\) has the form nil, \(x(y).P'.P'.P'\), \(\overline{x}(y).P'.P'\) or \(\overline{!x}(y).P'.P'\) it is simple to see that no \(\tau\) 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 \(P = P_0 \parallel P_1\)) 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 \(\tau\) 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 $\tau$ actions (using $l_{\text{intra}}$ and $r_{\text{intra}}$ rules) can be derived. Therefore, by inductive hypothesis and Lemma 4.13 the set $\text{Succ}(P)$ is finite.

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

**Lemma 4.15.** Let $B \in B$. Then

1. There are only finitely many pairs $x, \Delta$ such that $B \xrightarrow{\Delta xz} B'$ for some $z$ and $B'$.
2. There are only finitely many tuples $\Delta, x, y, B'$ such that $B \xrightarrow{\Delta xy} B'$.

**Proof.** By induction on the structure of $B$.

1. (Case Nil) No couple $x, \Delta$ such that $B \xrightarrow{\Delta xz} B'$ for some $z$ and $B'$ exists.
   (Case $I[P]$) For each subject $x \in \text{sub}(I)$, 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 $P \xrightarrow{xy} P'$ for some $z \notin \text{sub}(I)$ and $P'$. Since by definition $I$ is well-formed and the set of binders subjects $\text{sub}(I)$ is finite, we obtain (by the application of rule $\text{bio}_{\text{in}}$) that there are only finitely many pairs $\Delta, x$ such that $I[P] \xrightarrow{\Delta xz} I[P']$ for some $z$ and $I[P']$.
   (Case $B || B'$) By inductive hypothesis on $B$ and $B'$ the Lemma follows immediately.

2. (Case Nil) No tuple $\Delta, x, y, B'$ such that $B \xrightarrow{\Delta xy} B'$ exists.
   (Case $I[P]$) For each subject $x \in \text{sub}(I)$, we have (by Lemma 4.13) that there are only finitely many tuples $x, y, P'$ such that $P \xrightarrow{xy} P'$ and hence there are only finitely many tuples $x, y, P'$ such that $P \xrightarrow{xy} P'$ and $y \notin \text{sub}(I)$. Since by definition $I$ is well-formed and the set of binder subjects $\text{sub}(I)$ is finite, we obtain (by the application of rule $\text{bio}_{\text{out}}$) that there are only finitely many tuples $\Delta, x, y, I[P']$ such that $I[P] \xrightarrow{\Delta xy} I[P']$.
   (Case $B || B'$) By inductive hypothesis on $B$ and $B'$ the Lemma follows immediately.

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

**Proof.** By induction on the structure of $B$.

(Case Nil) In this case $\text{Succ}(B)$ is obviously $\emptyset$.

(Case $B[P]$) By Lemma 4.14 the set $\text{Succ}(P)$ is finite. It immediately follows (by application of rule $\text{pi}_{\text{bio}}$) that the set $\text{Succ}(B[P])$ is finite.

(Case $B_0 || B_1$) By inductive hypothesis the sets $\text{Succ}(B_0)$ and $\text{Succ}(B_1)$ 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 \( \tau \) 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 \( \tau \) actions (using \text{l\_inter} and \text{r\_inter} rules) can be derived. Therefore, by inductive hypothesis and Lemma 4.13, the set \( \text{Succ}(B) \) is finite.

**Corollary 4.17.** The transition system \( (Z_s, \tau) \) 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 \( \tau \) transition relation over safe bio-processes and the reduction relation over bio-processes agree.

**Lemma 4.18.** Let \( B, B' \in Z_s \). If \( B \equiv_b B' \) and \( B \xrightarrow{\tau} B'' \), then for some \( B''' \in Z_s \) 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 \( B \in Z \) and \( B'' \in \text{safe}(B) \). Then \( B \rightarrow B' \) iff \( B'' \xrightarrow{\tau} B''' \) and \( B''' \in \text{safe}(B') \).

**Proof.** (\( \Rightarrow \))

If \( B \rightarrow B' \) then there exist bio-processes \( B_0 \equiv_b B \) and \( B_1 \equiv_b B' \) such that

\[
B_0 = I [\pi(z) \cdot P_1 + M_1 | x(w) \cdot P_2 + M_2 | P_3] \parallel B_2 \\
\text{and } B_1 = I [P_1 | P_2 \{z/w\} | P_3] \parallel B_2
\]

or

\[
B_0 = \beta(x, \Delta) I_1^1 [\pi(z) \cdot R_1 + M_1 | Q_1] \parallel \beta(y, \Gamma) I_2^2 [y(w) \cdot R_2 + M_2 | Q_2] \parallel B_2 \\
\text{and } B_1 = I_1^1 [R_1 | Q_1] \parallel I_2^2 [R_2 \{z/w\} | Q_2] \parallel B_2
\]

with \( \alpha(\Gamma, \Delta) > 0 \) and \( z \notin \text{sub}(I_1^1) \cup \{x\} \). 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 \( z \notin \text{sub}(I_1^1) \cup \text{sub}(I_2^2) \cup \text{sub}(B_2) \cup \{x, y\} \). 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 \( B_0^s \in \text{safe}(B) \) and hence \( B'' \equiv_b B_0^s \). By Lemma 4.18, there exists \( B''' \in Z_s \) 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 \( B''' \in \text{safe}(B') \).

(\( \Leftarrow \))

It is enough to show that \( B \xrightarrow{\tau} B' \) implies \( B \rightarrow B' \). The proof is by induction on the inference of \( B \rightarrow B' \).
Corollary 4.20. Let $B \in \mathcal{Z}$ and $B' \in \text{safe}(B)$. 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 $(\mathcal{Z}_s, \tau)$

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

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

In order to define the $qo \preceq_b$, we first introduce a simplified structural congruence relation which is compatible with $\theta$. 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 $\equiv_{\text{dec}}$ 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 $\equiv_{\text{dec}}$ 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 $P_1 \equiv \rho P_2$ are substituted with $P_1 \equiv_{\text{dec}} P_2$.

Since $\equiv_{\text{dec}} \subset \equiv_\rho$ and $\equiv_{\text{dec}} \subset \equiv_b$ 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 $B_0 \equiv_{\text{dec}} B_1$ and $B_1 \xrightarrow{\theta} B_1'$ then there exists $B_0'$ such that $B_0 \xrightarrow{\theta} B_0'$ and $B_0' \equiv_{\text{dec}} B_1'$.

**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 $B \equiv_{\text{dec}} \prod_{i=1}^n I_i[P_i], B' \equiv_{\text{dec}} \prod_{i=1}^n I_i[Q_i]$ and $Q_i = P_i | P'_i$ 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 $\theta$. We present only one case because all the others are similar.

(Case bio.out) We have that $B_0 = \beta(x, \Delta)I^*[P] \xrightarrow{\Delta_{xy}} \beta(x, \Delta)I^*[P'] = B'_0$. By definition of $\preceq_b$ we have that there exists a process $Q$ such that $B_1 \equiv_{dec} \beta(x, \Delta)I^*[P|Q]$. By hypothesis we know that $P \xrightarrow{\tau_y} P'$ and hence, by applying the rules in Tab. 4, we can derive the transition $P|Q \xrightarrow{\tau_y} P'|Q$ from which we have $B_1 \xrightarrow{\Delta_{xy}} \beta(x, \Delta)I^*[P'|Q] = B'_1$. By $\preceq_b$ definition it results $B'_0 \preceq_b B'_1$. \qed

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 $S \subseteq 2^N$ 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:

\[
\begin{align*}
Sub(nil, S) &= \emptyset \\
Sub(\overline{x}(m).P, S) &= \{\overline{x}(m).P\} \cup Sub(P, S) \\
Sub(x(m).P, S) &= \{x(m).P\} \cup \left(\bigcup_{n \in S} Sub(P\{n/m\}, S)\right) \\
Sub(M + N, S) &= \{M + N\} \cup Sub(M, S) \cup Sub(N, S) \\
Sub(P \mid Q, S) &= Sub(P, S) \cup Sub(Q, S) \\
Sub(!\overline{x}(m).P, S) &= \{!\overline{x}(m).P\} \cup Sub(P, S) \\
Sub(!x(m).P, S) &= \{!x(m).P\} \cup \left(\bigcup_{n \in S} Sub(P\{n/m\}, S)\right)
\end{align*}
\]

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 $S \subseteq 2^N$ be a finite set of names. Then $Sub(P, S)$ is finite.


(case nil) The empty set is finite.

(case $\overline{x}(m).P$) By inductive hypothesis $Sub(P, S)$ is finite and hence by only
adding the element $\overline{\pi}(m).P$ the set is finite.

(case $x(m).P$) By inductive hypothesis for all $n \in S$ the set $\text{Sub}(P\{n/m\}, S)$
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 $\text{Sub}(M, S)$ and $\text{Sub}(N, S)$ are finite
set and hence their union with the set $\{M + N\}$ results in a finite set.

The other cases are similar.

\begin{corollary}
Let $P \in \mathcal{P}$. Then $\text{Sub}(P, fn(P))$ is finite.
\end{corollary}

\begin{proof}
Immediate from the fact that $fn(P)$ is a finite set of names and by
Lemma 4.28.
\end{proof}

We now prove some useful properties of the function $\text{Sub}$.

\begin{lemma}
Let $P \in \mathcal{P}$ and $S, S' \subseteq 2^N$. If $S' \subseteq S$ then $\text{Sub}(P, S') \subseteq
\text{Sub}(P, S)$.
\end{lemma}

\begin{proof}
By induction on the structure of $P$.

(Case $x(m).P$) By inductive hypothesis for all names $n \in S'$ we have that
$\text{Sub}(P\{n/m\}, S') \subseteq \text{Sub}(P\{n/m\}, S)$. As a consequence, we obtain the
condition $\bigcup_{n \in S} \text{Sub}(P\{n/m\}, S') \subseteq \bigcup_{n \in S} \text{Sub}(P\{n/m\}, S)$ and hence the
Lemma holds.

The other cases are trivial.
\end{proof}

\begin{lemma}
Let $P, Q \in \mathcal{P}$ and $S, S', S'' \subseteq 2^N$. If $S' \subseteq S''$ and $\text{Sub}(P, S') \subseteq
\text{Sub}(Q, S'')$ then $\text{Sub}(P, S' \cup S) \subseteq \text{Sub}(Q, S'' \cup S)$.
\end{lemma}

\begin{proof}
By induction on the structure of $P$.

(Case $\overline{\pi}(m).P$) By definition $\text{Sub}(\overline{\pi}(m).P', S') = \{\overline{\pi}(m).P'\} \cup \text{Sub}(P', S')$.
By hypothesis and Lemma 4.30 we have $\text{Sub}(\overline{\pi}(m).P', S') \subseteq \text{Sub}(Q, S'') \subseteq
\text{Sub}(Q, S'' \cup S)$ and therefore $\overline{\pi}(m).P' \in \text{Sub}(Q, S'' \cup S)$. By definition of
$\text{Sub}$ function and by inductive hypothesis, this means that $\text{Sub}(P', S'' \cup S) \subseteq
\text{Sub}(Q, S'' \cup S)$ and hence the Lemma holds.

(Case $x(m).P$) By definition we have that $\text{Sub}(x(m).P', S') = \{x(m).P'\} \cup
(\bigcup_{n \in S} \text{Sub}(P\{n/m\}, S'))$. By hypothesis and Lemma 4.30 $\text{Sub}(\overline{\pi}(m).P', S') \subseteq
\text{Sub}(Q, S'') \subseteq \text{Sub}(Q, S'' \cup S)$ and therefore $\overline{\pi}(m).P' \in \text{Sub}(Q, S'' \cup S)$. By
definition of $\text{Sub}$ function and by inductive hypothesis, this means that for all $n \in S'' \cup S$ we have $\text{Sub}(P\{n/m\}, S'' \cup S) \subseteq \text{Sub}(Q, S'' \cup S)$ and hence the
Lemma follows.

(Case $P_0|P_1$) By hypothesis and $\text{Sub}$ function definition we have $\text{Sub}(P_0, S') \subseteq
\text{Sub}(Q, S'')$ and $\text{Sub}(P_1, S') \subseteq \text{Sub}(Q, S'')$. By inductive hypothesis we have
$\text{Sub}(P_0, S' \cup S) \subseteq \text{Sub}(Q, S'' \cup S)$ and $\text{Sub}(P_1, S' \cup S) \subseteq \text{Sub}(Q, S'' \cup S)$ and
therefore $\text{Sub}(P_0, S' \cup S) \cup \text{Sub}(P_1, S' \cup S) \subseteq \text{Sub}(Q, S'' \cup S)$. The Lemma follows.

The other cases are similar. \qed

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

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

\[
\text{Sub}(\text{Nil}, S) = \emptyset
\]

\[
\text{Sub}(I[P], S) = \{ I[P'] \mid P' \in \text{Sub}(P, S \cup \text{sub}(I)) \}
\]

\[
\text{Sub}(B || B', S) = \text{Sub}(B, S) \cup \text{Sub}(B', S)
\]

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

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

**Corollary 4.34.** Let $B \in Z_s$. Then $\text{Sub}(B, \text{fn}(B))$ is finite.

**Proof.** Immediate from the fact that $\text{fn}(B)$ is a finite set of names and by Lemma 4.33. \qed

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

**Proof.** By induction on the structure of $B$.
(Case $I[P]$) By Lemma 4.30 we have $\text{Sub}(P, S' \cup \text{sub}(I)) \subseteq \text{Sub}(P, S \cup \text{sub}(I))$. This means $\{ I[P'] \mid P' \in \text{Sub}(P, S' \cup \text{sub}(I)) \} \subseteq \{ I[P'] \mid P' \in \text{Sub}(P, S \cup \text{sub}(I)) \}$ and the Lemma follows.

The other cases are trivial. \qed

**Lemma 4.36.** Let $B, B' \in Z_s$ and $S, S', S'' \subseteq 2^N$. If $S' \subseteq S''$ and $\text{Sub}(B, S') \subseteq \text{Sub}(B', S'')$ then $\text{Sub}(B, S' \cup S) \subseteq \text{Sub}(B', S'' \cup S)$. 21

(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 $P \xrightarrow{a} P'$ 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 $\theta$.

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

- If $\theta = \pi y$ then $Sub(P', fn(P')) \subseteq Sub(P, fn(P))$.
- If $\theta = xy$ then $Sub(P', fn(P')) \subseteq Sub(P, fn(P) \cup \{y\})$.
- 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 $P = \pi(y).P'. \xrightarrow{\pi y} P'$. By definition we have that $Sub(P, fn(P)) = \{\pi(y).P'\} \cup Sub(P', fn(P))$. 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 $P_0 \xrightarrow{\pi y} P_0'$ 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 $P = x(w).P'. \xrightarrow{\pi y} P'[y/w]$. By definition we have that $Sub(P, fn(P) \cup \{y\}) = \{\pi(w).P'\} \cup (\bigcup_{n \in fn(P) \cup \{y\}} Sub(P'[n/w], fn(P) \cup \{y\})$ 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)
\( \text{Sub}(P' \{y/w\}, fn(P' \{y/w\})) \subseteq \text{Sub}(P' \{y/w\}, fn(P) \cup \{y\}) \). The Lemma follows.

(c) \((\text{Case } \text{intra})\) By hypothesis we have \(P_0 \xrightarrow{x} P_0'\) and \(P_1 \xrightarrow{x} P_1'\). By (a) and (b) we have that \(\text{Sub}(P_0', fn(P_0')) \subseteq \text{Sub}(P_0, fn(P_0))\) and \(\text{Sub}(P_1', fn(P_1')) \subseteq \text{Sub}(P_1, fn(P_1) \cup \{z\})\). By applying Lemma 4.10, Lemma 4.30 and Lemma 4.31 we obtain \(\text{Sub}(P_0', fn(P_0') \cup fn(P_1')) \subseteq \text{Sub}(P_0, fn(P_0) \cup fn(P_1))\) and \(\text{Sub}(P_1', fn(P_0') \cup fn(P_1')) \subseteq \text{Sub}(P_1, fn(P_0) \cup fn(P_1))\) and, by Sub function definition, the Lemma follows. \(\blacksquare\)

**Lemma 4.38.** Let \(B, B' \in \mathcal{Z}_s\). Suppose \(B \overset{\theta}{\to} B'\). Then

(a) if \(\theta = \Delta xy\) then \(\text{Sub}(B', fn(B')) \subseteq \text{Sub}(B, fn(B))\)

(b) if \(\theta = \Delta yx\) then \(\text{Sub}(B', fn(B')) \subseteq \text{Sub}(B, fn(B) \cup \{y\})\)

(c) if \(\theta = \tau\) then \(\text{Sub}(B', fn(B')) \subseteq \text{Sub}(B, fn(B))\)

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

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

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

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
\[
\mathcal{P}_B = \{B' \in \mathcal{Z}_s \mid \text{Sub}(B', fn(B')) \subseteq \text{Sub}(B, fn(B)) \land box_{\#}(B') = box_{\#}(B)\}
\]
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 \( B' \overset{\tau}{\rightarrow} B'' \) then \( B'' \in \mathcal{P}_B \).

*Proof.* \( B' \in \mathcal{P}_B \) implies \( \text{Sub}(B', fn(B')) \subseteq \text{Sub}(B, fn(B)) \). Since by hypothesis \( B' \overset{\tau}{\rightarrow} B'' \), by Lemma 4.38 we have \( \text{Sub}(B'', fn(B'')) \subseteq \text{Sub}(B', fn(B')) \). Transitivity of \( \subseteq \) proves \( \text{Sub}(B'', fn(B'')) \subseteq \text{Sub}(B, fn(B)) \), which means \( B'' \in \mathcal{P}_B \). \( \square \)

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 \( \text{Deriv}(B) \subseteq \mathcal{P}_B \).

*Proof.* Immediate from Lemma 4.40. \( \square \)

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

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

\[
B' \equiv_{dec} \prod_{i=1}^n I_i \prod_{j=1}^m Q_{i,j}
\]

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

*Proof.* Immediate from the bio-processes syntax and from the definition of \( \text{Sub} \) function. \( \square \)

In the following Lemma we define the relation \( =_s \) over bio-processes according to Def. 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 \( \text{Sub}(B, fn(B)) \). If \( I[P_1], \ldots, I[P_n] =_s I'[Q_1], \ldots, I'[Q_m] \) then \( I[I_1^m P_i] \cong_b I'[\prod_{i=1}^m Q_i] \).

*Proof.* If \( I[P_1], \ldots, I[P_n] =_s I'[Q_1], \ldots, I'[Q_m] \) then there exists an injection \( f : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\} \) 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 \( \equiv_{dec} \) definition we can write \( I'[\prod_{i=1}^m Q_i] \equiv_{dec} I'[\prod_{i=1}^n Q_i] | \prod_{i=n+1}^m Q_i \) and by the equality result we have \( I[I_1^m P_i] \equiv_{dec} I'[\prod_{i=1}^m Q_i] | \prod_{i=n+1}^m Q_i \) (notice that \( I = I' \)), which means \( I[I_1^m P_i] \cong_b I'[\prod_{i=1}^m Q_i] \). \( \square \)
The following theorem shows that the $\preceq_b$ is a wqo.

**Theorem 4.44.** Let $B \in Z_s$. The relation $\preceq_b$ is a wqo over $P_B$.

**Proof.** We take an infinite sequence $B_1, \ldots, B_i, \ldots$ such that $B_i \in P_B$ for $i > 0$. By Lemma 4.42, for any $i$ we have that:

$$B_i \equiv_b \prod_{j=1}^{n_i} I_{i,j}^{m_{i,j}} P_{i,j,k}^{l_{i,j,k}}$$

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

$$I_{i,1}[P_{i,1,1}], \ldots, I_{i,1}[P_{i,1,m_{i,1}}]$$
$$I_{i,2}[P_{i,2,1}], \ldots, I_{i,2}[P_{i,2,m_{i,2}}]$$

$$\ldots$$
$$I_{i,n}[P_{i,n,1}], \ldots, I_{i,n}[P_{i,n,m_{i,n}}]$$

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

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. $\preceq_\ast$; 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. $\preceq_\ast$. We continue for all the $n$ subsequences.

We end up with an infinite subsequence $B_{n_0}, B_{n_1}, \ldots$ (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. $\preceq_\ast$. By Lemma 4.43 we obtain:

$$I_{n_0,j}[\prod_{l=1}^{m_{n_0,j}} P_{n_0,j,l}] \preceq_b \cdots \preceq_b I_{n_i,j}[\prod_{l=1}^{m_{n_i,j}} P_{n_i,j,l}] \preceq_b \cdots$$

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 $B \in Z_s$. The transition system $(\text{Deriv}(B), \rightarrow, \preceq_b)$ is a well-structured transition system with decidable $\preceq_b$ and computable $\text{Succ}$.

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

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

**Corollary 4.46.** Let $B \in Z_s$. 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 $(j, k_1, ..., k_n)$ where $j$ is the index of the next instruction to be executed and $k_1, ..., k_n$ are the current contents of the registers. The execution is defined by a transition relation among states

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

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

A state $(j, k_1, ..., k_n)$ 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.
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:

$$\pi ::= \cdots | \overline{y}$$

which has precedence over the usual $\overline{y}$. 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. Sec. 3 and the new reduction semantics is reported in Tab. 5. Moreover, we denote with $BL^{gp}$ this extension and with $Z^{gp}$ the set of well-formed systems of $BL^{gp}$. Tab. 5 shows that intra-communications and inter-communications with outputs $\overline{y}$ can be derived only if, respectively, no intra-communications and inter-communications through immediate outputs $\overline{y}$ are enabled. Notice that in Tab. 5 with $B \not\rightarrow B'$ we mean that no bi-process $B'$ such that $B \rightarrow B'$ exists.

It is important to note that the introduction of priorities causes the gen-
ereation 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 Sec.4) that satisfies the strong compatibility property.

\[
\begin{align*}
\text{(intra)} & \quad I[Z(z), P_1 + M_1 | x(w), P_2 + M_2 | P_3] \lor B \Rightarrow I[P_1 | P_2 \{z/w\}, P_3] \lor B \\
\text{(intra)} & \quad I[Z(z), P_1 + M_1 | x(w), P_2 + M_2 | P_3] \lor B \Rightarrow I[P_1 | P_2 \{z/w\}, P_3] \lor B \\
\text{provided } \alpha(\Gamma, \Delta) > 0 \text{ and } z \notin \text{sub}(I_1) \\
\text{and where } I_1 = \beta(x, \Delta) I_1^* \text{ and } I_2 = \beta(y, \Gamma) I_2^* \\
\text{(inter)} & \quad I_1[P_1] \lor I_2[P_2] \lor B \Rightarrow I_1[R_1 | Q_1] \lor I_2[R_2 \{z/w\} | Q_2] \lor B \\
\text{provided } \alpha(\Gamma, \Delta) > 0 \text{ and } z \notin \text{sub}(I_1) \\
\text{and where } I_1 = \beta(x, \Delta) I_1^* \text{ and } I_2 = \beta(y, \Gamma) I_2^* \\
\text{(struct)} & \quad B_1 \equiv_b B_1' \quad B_1' \rightarrow B_2' \quad B_2' \equiv_b B_2 \\
\text{B_1 \rightarrow B_2} \\
\text{(struct)} & \quad B_1 \equiv_b B_1' \quad B_1' \rightarrow B_2' \quad B_2' \equiv_b B_2 \quad B_1' \not\equiv_b B_2 \\
\text{B_1 \rightarrow B_2} \\
\end{align*}
\]

Table 5: Reduction semantics of \(BL^{gp}\).

**Definition 5.1.** The \(BL^{gp}\) Transition System (TS\(^{gp}\)) is referred as \((Z^{gp}, \rightarrow)\), where \(Z^{gp}\) is the set of well-formed systems and \(\rightarrow \subseteq Z^{gp} \times Z^{gp}\) 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 \(Z \rightarrow^1 Z_1 \rightarrow^2 \cdots \rightarrow^{k-1} Z_{k-1} \rightarrow^k Z'\).

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:

\[
\llbracket (j, k_1, \ldots, k_n) \rrbracket_R^{gp} = B
\]

28
where,
\[ B \triangleq \text{Switch}_j \parallel [(1, I_1)]^{op}_{\mathcal{R}} \parallel \cdots \parallel [(m, I_m)]^{op}_{\mathcal{R}} \parallel [r_1 = k_1]^{op}_{\mathcal{R}} \parallel \cdots \parallel [r_n = k_n]^{op}_{\mathcal{R}} \]

The Switch\(_j\) bio-process is defined in the following way
\[ \text{Switch}_j \triangleq \beta(\text{ins} : \text{Ins})\beta(i_1 : \text{Ins}_1)\beta(i_2 : \text{Ins}_2)\cdots\beta(i_m : \text{Ins}_m) \]
\[ [ !x(e).\text{Switch} \parallel \text{Switch} \parallel \bar{i}_j(e).\text{nil} ] \]

Switch \(\triangleq_p \text{ins(type)}.(\text{type} \langle e \rangle .\text{nil} | (\sum_{i=1}^m \text{ins}_i(e).\bar{x}(e).\bar{i}_j(e).\text{nil}))\)

and its graphical representation is

\[ (\text{ins} : \text{Ins}) (i_1 : \text{Ins}_1) (i_2 : \text{Ins}_2) \cdots (i_m : \text{Ins}_m) \]
\[ !x(e).\text{Switch} \parallel \text{Switch} \parallel \bar{i}_j(e).\text{nil} \]

The encoding of the instruction \((i, \text{Incr}(r_j))\) is
\[ [(i, \text{Incr}(r_j))]^{op}_{\mathcal{R}} = \beta(\text{act} : \text{Ins}_i)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_j) \]
\[ [ !x(e).\text{Inc}_i | \text{Inc}_i ] \]

Inc\(_i\) \(\triangleq_p\) act(e).(inc(e).\bar{x}(e).next(ins_{i+1}).\text{nil})

and its graphical representation is

\[ (\text{act} : \text{Ins}_i) (\text{next} : \text{Ins}) (\text{inc} : \text{IReg}_j) \]
\[ !x(e).\text{Inc}_i | \text{Inc}_i \]

The encoding of the instruction \((i, \text{DecJump}(r_j, s))\) is
\[ [(i, \text{DecJump}(r_j, s))]^{op}_{\mathcal{R}} = \beta(\text{act} : \text{Ins}_i)\beta(\text{next} : \text{Ins})\beta(t_y : \text{Test}_j^{yes}) \]
\[ \beta(t_n : \text{Test}_j^{no}) \]
\[ [ !x(e).\text{DecJump}_i | \text{DecJump}_i ] \]

DecJump\(_i\) \(\triangleq_p\) act(e).(t_y(e).Dec\(_i\).t_n(e).Jump\(_i\))

Dec\(_i\) \(\triangleq_p\) \bar{x}(e).next(ins_{i+1}).\text{nil}

Jump\(_i\) \(\triangleq_p\) \bar{x}(e).next(ins_s).\text{nil}

and its graphical representation is

\[ (\text{act} : \text{Ins}_i) (\text{next} : \text{Ins}) (t_y : \text{Test}_j^{yes}) (t_n : \text{Test}_j^{no}) \]
\[ !x(e).\text{DecJump}_i | \text{DecJump}_i \]
The encoding of a register \( r_j \) with content \( l \) is defined as follows

\[
[r_j = l]_R^{gp} \triangleq_B \beta(t_y : Test^\text{yes})\beta(t_n : Test^\text{no})\beta(inc : IReg_j)
\]

\[
{} !inc(e).Decrement | !\lnot_n(e).nil | \text{Val}_l
\]

where,

\[
\text{Val}_l \triangleq_P \{ \text{Decrement} | \cdots | \text{Decrement} \}
\]

\[
\text{Decrement} \triangleq_P \lnot_y(e).nil
\]

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

\[
(t_y : Test^\text{yes}) (t_n : Test^\text{no}) (inc : IReg_j)
\]

\[
{} !inc(e).Decrement | !\lnot_n(e).nil | \text{Val}_l
\]

Finally, the function \( \alpha \) is defined in the following way

\[
\alpha(\text{Type}_1, \text{Type}_2) = \begin{cases} 
1 & \text{if } \text{Type}_1 = \text{Type}_2 \\
0 & \text{otherwise}
\end{cases}
\]

The encoding produces a system in \( Z^{gp} \) due to the unique immediate output in the definition of the process \( \text{Decrement} \). 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 \( \text{act} \) with the switching box \( \text{Switch}_j \).

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 \( \text{Decrement} \) 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 \( \text{Decrement} \), 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 \( \text{Ins} \)) for the activation of instruction \( i+1 \).
The box encoding the instruction \((i, DecJump(r_j, s))\), after its activation, presents an alternative behavior (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 \(Test^{\text{yes}}\) and \(Test^{\text{no}}\), respectively. In the register box, outputs on channel \(t_y\), if present, are of the form \(\overline{t_y}(e)\) and hence generate immediate inter-communications.

If the encoded register contains a value \(n > 0\), then \(n\) parallel compositions of process \(Decrement \triangleq P_{t_y}(e)\).\text{nil}\) are present and hence inter-communications on output \(\overline{t_y}(e)\) have always precedence w.r.t. the inter-communications that the process \(!\overline{t_n}(e).\text{nil}\) 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 \(Dec\), 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 \(!\overline{t_n}(e).\text{nil}\) 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 \((j, k_1, \ldots, k_n) \xrightarrow{R} (j', k'_1, \ldots, k'_n)\), then there exists a well-formed system \(Z \in \mathcal{Z}^{gp}\) such that \(\llbracket (j, k_1, \ldots, k_n) \rrbracket^R_R \xrightarrow{+} Z\) and \(Z \equiv_b \llbracket (j', k'_1, \ldots, k'_n) \rrbracket^R_R\).

**Proof.** The proof is by case analysis. There are three cases: (i) Instruction \(I_j = DecJump(r_1, s)\) with \(r_1\) content greater than zero; (ii) Instruction \(I_j = DecJump(r_1, s)\) with \(r_1\) content equal to zero; (iii) Instruction \(I_j = Incr(r_1)\). We only prove case (i), because the other cases can be proved similarly.

(i) We consider the computation of the bio-process \(\llbracket (j, k_1, \ldots, k_n) \rrbracket^R_R\). An inter-communication between the component \(Switch_j\) and the component \(\llbracket (j, DecJump(r_1, s)) \rrbracket^R_R\) is consumed. In particular, the two boxes synchronize on output \(\overline{t_j}(e)\) and input \(act(e)\) through their binders of type \(Ins_j\).
This cause the activation of the $\mathbb{J}(j,\text{DecJump}(r_1, s))^{op}_R$ 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 $\mathbb{J}(j,\text{DecJump}(r_1, s))^{op}_R$ 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 $\text{Test}^{\text{yes}}$ and $\text{Test}^{\text{no}}$, respectively. By hypothesis the content of the register $r_i$ is greater than 0 and hence the internal structure of the box $[r_i = k_i]^{op}_R$ 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 $\text{Test}^{\text{no}}$ and $k_j$ inter-communications through interfaces with type $\text{Test}^{\text{yes}}$. However, since the inter-communications through interfaces with type $\text{Test}^{\text{yes}}$ are immediate, they have precedence w.r.t. the one through interfaces with type $\text{Test}^{\text{no}}$.

The consumption of one of the immediate inter-communications deletes one of the $k_i$ Decrement processes in $[r_i = k_i]^{op}_R$, resulting (for all the possible inter-communications of this type) in a bio-process structurally congruent to $[r_i = k_i - 1]^{op}_R$, and activates process $\text{Dec}_j$ 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 $\text{next} \langle \text{ins}_j+1 \rangle.\text{nil}$. This produces a synchronization between the instruction box and the switching box, which generates an inter-communication on output $\text{next} \langle \text{ins}_j+1 \rangle$ and input $\text{ins}(\text{type})$ through interfaces of type $\text{Ins}$.

The instruction box is now returned in its form $\mathbb{J}(j,\text{DecJump}(r_1, s))^{op}_R$, while in the switching box the process

$$(\text{ins}_{j+1}(e).\text{nil} | (\sum_{o=1}^m \text{ins}_o(e).\text{x}(e).\text{ins}_o(e).\text{nil}))$$

is enabled. An intra-communication on channel $\text{ins}_{j+1}$ 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 $\text{Switch}_{j+1}$.

**Lemma 5.3.** Let $R$ be a RAM with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$. If the system $Z = \mathbb{J}(j, k_1, ..., k_n)_{op}^{op}_R$ can produce a transition $Z \rightarrow Z_1$, then there exists a computation $Z \rightarrow Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_l$ such that $Z_l = \mathbb{J}(j', k_1', ..., k_n')_{op}^{op}_R$ and $(j, k_1, ..., k_n) \rightarrow R (j', k_1', ..., k_n')$

**Proof.** Consider the structure of the bio-process $Z = \mathbb{J}(j, k_1, ..., k_n)_{op}^{op}_R$. If the bio-process $Z$ can perform a first step $Z \rightarrow Z_1$, this corresponds to an inter-communication between the box $\text{Switch}_j$ 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 \((j, k_1, ..., k_n)\) of the RAM \(R\), generating a new state \((j', k'_1, ..., k'_n)\).

There are three cases: (i) Instruction \(I_j = \text{DecJump}(r_l, s)\) with \(r_l\) content greater than zero; (ii) Instruction \(I_j = \text{DecJump}(r_l, s)\) with \(r_l\) content equal to zero; (iii) Instruction \(I_j = \text{Incr}(r_l)\). 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

\[
[(j, \text{Incr}(r_l))]_{R}^{op} = \beta(\text{act} : \text{Ins}_j)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_l)
\]

\[
[!x(e).\text{Inc}_j | \text{act}(e)\text{.Inc}(e)\text{.Val}(e)\text{.next}(\text{ins}_{j+1}).\text{nil}]
\]

This box is the only one able to synchronize with the box \(\text{Switch}_j\) for an inter-communication through interfaces of type \(\text{Ins}_j\). After the inter-communication we have that in \(Z_1\) the box encoding the instruction \(j\) becomes structurally congruent to

\[
B' = \beta(\text{act} : \text{Ins}_j)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_l)
\]

\[
[!x(e).\text{Inc}_j | \text{inc}(e)\text{.Val}(e)\text{.next}(\text{ins}_{j+1}).\text{nil}]
\]

At this point, the only possible action \(Z_1 \rightarrow Z_2\) is the inter-communication between the box \(B'\) and the box \([r_l = k]_{R}^{op}\) through their interfaces of type \(\text{IReg}_l\) on output \(\text{inc}(e)\) and input \(\text{inc}(e)\), respectively. After the inter-communication the box encoding for the register \(r_l\) becomes structurally congruent to

\[
\beta(t_y : \text{Test}_{j_1}^{yes})\beta(t_n : \text{Test}_{j_1}^{no})\beta(\text{inc} : \text{IReg}_l)
\]

\[
[!\text{inc}(e)\text{.Decrement} | !\text{val}(e)\text{.nil} | \text{val}_k | \text{Decrement}]
\]

which corresponds to the box \([r_l = k + 1]_{R}^{op}\). Instead the box encoding for the register \(r_{l}\) becomes structurally congruent to

\[
B'' = \beta(\text{act} : \text{Ins}_j)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_l)
\]

\[
[!x(e).\text{Inc}_j | \text{inc}(e)\text{.next}(\text{ins}_{j+1}).\text{nil}]
\]

Now, the action \(Z_2 \rightarrow Z_3\) is the intra-communication of \(B''\) on channel \(x\) which becomes a box \(B'''\) with internal structure \(!x(e).\text{Inc}_j | \text{next}(\text{ins}_{j+1}).\text{nil}\).

and the action \(Z_3 \rightarrow Z_4\) is the inter-communication between the box \(B'''\)
and the switching box. After the inter-communication, the instruction box returns in its initial form \([(j, \text{Incr}(r_j))]_{R}^{gp}\) and the switching box starts a sequence of intra-communications which produces a box structurally congruent to \(\text{Switch}_{j+1}\) and representing the sequence of actions \(Z_4 \rightarrow Z_5 \rightarrow Z_6\). It is easy to see that \(Z_6\) is congruent to \([(j+1, k_1, ..., k_{i-1}, k+1, k_{i+1}, ..., k_n)]_{R}^{gp}\).

Lemma 5.2 and Lemma 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 \((1, I_1), ..., (m, I_m)\) and initial state \((j, k_1, ..., k_n)\). Then the computation of the RAM \(R\) terminates if and only if the computation of the system \(Z = [(j, k_1, ..., k_n)]_{R}^{gp}\) terminates.

**Proof.** (⇒) 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 \((j', k_1', ..., k_n')\), 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 \(Z \rightarrow Z_1 \rightarrow \cdots \rightarrow Z_i \rightarrow \cdots\). By Lemma 5.2 we have that there exists a well-formed system \(Z' \in Z^{gp}\) such that \(Z \rightarrow^{*} Z'\) and \(Z' \equiv_b [(j', k_1', ..., k_n')]_{R}^{gp}\). By assumption, \(Z\) does not terminate and hence there exists \(Z''\) such that \(Z' \rightarrow Z''\). By Lemma 5.3 we have that there exists a computation \(Z' \rightarrow Z'' \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_i\) such that \(Z_l = [(j'', k_1'', ..., k_n'')]_{R}^{gp}\) and \((j', k_1', ..., k_n') \rightarrow_{R} (j'', k_1'', ..., k_n'')\). But this contradicts our hypothesis, which states that \((j', k_1', ..., k_n')\) is a terminated state and therefore the implication holds.

(⇐) By hypothesis we have that the system \(Z\) terminates. This means that there exists a computation \(Z \rightarrow^{i} Z'\) such that \(Z' \not\rightarrow^{i}\). The proof is by contradiction assuming that the RAM \(R\) does not terminate. By applying Lemma 5.3 we have that \(Z' \equiv_b [(j', k_1', ..., k_n')]_{R}^{gp}\) and that \((j, k_1, ..., k_n) \rightarrow_{R} \cdots \rightarrow_{R} (j', k_1', ..., k_n')\). By assumption we have that \((j', k_1', ..., k_n') \rightarrow_{R} (j'', k_1'', ..., k_n'')\) and hence by Lemma 5.2 we have that there exists a well-formed system \(Z'' \in Z^{gp}\) such that \(Z' \rightarrow^{+} Z''\) and \(Z'' \equiv_b [(j'', k_1'', ..., k_n'')]_{R}^{gp}\). But this contradicts our hypothesis, which states that \(Z' \not\rightarrow^{i}\) 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 \(f_{\text{split}}\) and \(f_{\text{join}}\) axioms present of the original version of Beta-bindners [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:

\[
\begin{align*}
\text{cond} &::= I[P] \mid I[P], I[P] \\
\text{verb} &::= \text{split}(I[P], I[P]) \mid \text{join}(I[P]) \\
\text{event} &::= \bullet \mid (\text{cond}) \text{ verb} \\
E &::= \text{event} \mid E :: E
\end{align*}
\]

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 \( \text{event} \) is the composition of a condition \( \text{cond} \) and an action \( \text{verb} \). 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 Sec. 3, while a new definition of structural congruence over systems (Def. 5.5) and a new reduction semantics (Tab. 6) are introduced. In Tab. 6, a \( \text{join} \) event substitutes two boxes with a single one, while a \( \text{split} \) event substitutes a box with two boxes.

We denote with \( BL^e \) this extension and with \( Z^e \) 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 \( E \equiv_e E' \).

\[
\begin{align*}
- (B_0) \text{ split}(B_1, B_2) &\equiv_e (B'_0) \text{ split}(B'_1, B'_2) \\
&\text{if } B_0 \equiv_b B'_0, B_1 \equiv_b B'_1 \text{ and } B_2 \equiv_b B'_2 \\
- (B_0, B_1) \text{ join}(B_2) &\equiv_e (B'_0, B'_1) \text{ join}(B'_2) \\
&\text{if } B_0 \equiv_b B'_0, B_1 \equiv_b B'_1 \text{ and } B_2 \equiv_b B'_2 \\
- E::\bullet &\equiv_e E \\
- E_0::E_1 &\equiv_e E_1::E_0 \\
- E_0::(E_1::E_2) &\equiv_e (E_0::E_1)::E_2
\end{align*}
\]

Figure 3: Structural laws for events.
Definition 5.6. The $BL^e$ Transition System ($TS^e$) is referred as $(Z^e, \rightarrow)$, where $Z^e$ is the set of well-formed systems and $\rightarrow \subseteq Z^e \times Z^e$ 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 $Z \rightarrow^1 Z_1 \rightarrow^2 \cdots \rightarrow^{k-1} Z_{k-1} \rightarrow^k Z'$.

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:

$$\llbracket (j, k_1, \ldots, k_n) \rrbracket^e_R = \langle B, E \rangle$$

where,

$$B \triangleq_B Switch_j \ || \ \llbracket (1, I_1) \rrbracket^e_R \ || \ \cdots \ || \ \llbracket (m, I_m) \rrbracket^e_R \ || \ \llbracket r_1 = k_1 \rrbracket^e_R \ || \ \cdots \ || \ \llbracket r_n = k_n \rrbracket^e_R$$

$$E \triangleq_E ZeroToOne \ || \ ZeroToZero_1 \ || \ \cdots \ || \ ZeroToZero_n$$

The differences w.r.t. the encoding presented in Subsec. 5.2 concern the encoding of instructions and registers. The encoding of the instruction

---

| (intra) | \( \langle I[z(x)].P_1 + M_1 \ | \ x(w).P_2 + M_2 \ | P_3, E \rangle \rightarrow \langle I[z(w)].P_1 \ | P_2 \ | P_3, E \rangle \) |
|--------|------------------------------------------------------------------------------------------------------------------|
|        | \( P_1 \equiv_p x(z).P_1 + M_1 \ | Q_1 \quad P_2 \equiv_p y(w).P_2 + M_2 \ | Q_2 \)                                                                |
| (inter) | \( \langle I_1[P_1] \ | I_2[P_2], E \rangle \rightarrow \langle I_1[R_1 \ | Q_1] \ | I_2[R_2 \ | \{z/w\} \ | Q_2], E \rangle \) |
|        | provided \( \alpha(\Gamma, \Delta) > 0 \) and \( z \notin \text{sub}(I_1) \)                                           |
|        | and where \( I_1 = \beta(x, \Delta) I_1^* \) and \( I_2 = \beta(y, \Gamma) I_2^* \)                                |
| (split) | \( \langle I[P], E \rangle \rightarrow \langle I_0[P_0] \ | I_1[P_1], E \rangle \)                                        |
|        | where \( E = (I[P]) \) split(I_0[P_0], I_1[P_1]) :: E'                                                            |
| (join)  | \( \langle I_0[P_0] \ | I_1[P_1], E \rangle \rightarrow \langle I[P], E \rangle \)                                      |
|        | where \( E = (I_0[P_0], I_1[P_1]) \) join(I[P]) :: E'                                                             |
| (struct) | \( \frac{Z_1 \equiv Z_1' \quad Z_1' \rightarrow Z_2' \quad Z_2' \equiv Z_2}{Z_1 \rightarrow Z_2} \)                |
| (redux) | \( \frac{\langle B, E \rangle \rightarrow \langle B', E \rangle}{\langle B \ | B_1, E \rangle \rightarrow \langle B' \ | B_1, E \rangle} \) |

Table 6: Operational semantics of $BL^e$. 

\( (i, \text{Incr}(r_j)) \) is
\[
\llbracket (i, \text{Incr}(r_j)) \rrbracket = \beta(\text{act} : \text{Ins}_i)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_j) \\
\beta(\text{ack} : \text{IAck}_j)!x(e).\text{Inc}_i \mid \text{Inc}_i
\]
\( \text{Inc}_i \triangleq_P \text{act}(e)\overline{\text{inc}(e)}\cdot\text{ack}(e)\cdot\overline{x(e)}\cdot\overline{\text{next}(\text{ins}_{i+1})}\cdot\text{nil} \)

and its graphical representation is

\[
(\text{act} : \text{Ins}_i) \quad (\text{next} : \text{Ins}) \quad (\text{inc} : \text{IReg}_j) \quad (\text{ack} : \text{IAck}_j) \\
!x(e).\text{Inc}_i \mid \text{Inc}_i
\]

The encoding of the instruction \( (i, \text{DecJump}(r_j, s)) \) is
\[
\llbracket (i, \text{DecJump}(r_j, s)) \rrbracket = \beta(\text{act} : \text{Ins}_i)\beta(\text{next} : \text{Ins})\beta(\text{dec} : \text{DReg}_j) \\
\beta(\text{ack} : \text{DAck}_j)\beta(\text{test} : \text{Test}_j) \\
[ !x(e).\text{DecJump}_i \mid \text{DecJump}_i ]
\]
\( \text{DecJump}_i \triangleq_P \text{act}(e)\cdot\text{test}(t)\cdot(\overline{t(e)}\cdot\text{nil} \mid (\text{yes}(e)\cdot\text{Dec}_i + \text{no}(e)\cdot\text{Jump}_i)) \\
\text{Dec}_i \triangleq_P \text{dec}(e)\cdot\text{ack}(e)\cdot\overline{x(e)}\cdot\overline{\text{next}(\text{ins}_{i+1})}\cdot\text{nil} \\
\text{Jump}_i \triangleq_P \overline{x(e)}\cdot\overline{\text{next}(\text{ins}_s)}\cdot\text{nil} \)

and its graphical representation is

\[
(\text{act} : \text{Ins}_i) \quad (\text{next} : \text{Ins}) \quad (\text{dec} : \text{DReg}_j) \quad (\text{ack} : \text{DAck}_j) \quad (\text{test} : \text{Test}_j) \\
!x(e).\text{DecJump}_i \mid \text{DecJump}_i
\]

The encoding of a register \( r_j \) with content \( l \) is defined as follows
\[
\llbracket r_j = l \rrbracket = \begin{cases} 
B^0_j & \text{if } l = 0 \\
B^\text{test}_j \mid B^l_j & \text{otherwise}
\end{cases}
\]

where the boxes \( B^0_j \), \( B^\text{test}_j \), \( B^l_j \) are defined as follows
\[
B^0_j \triangleq_B \beta(\text{test} : \text{Test}_j)\beta(\text{inc} : \text{IReg}_j)\beta(\text{ack}_i : \text{IAck}_j) \\
\beta(\text{ack}_d : \text{DAck}_j)[ \overline{\text{inc}(e)}\cdot\text{nil} \mid \overline{\text{test}(\overline{\text{no}})}\cdot\text{nil} ]
\]
\[
B^\text{test}_j \triangleq_B \beta(\text{test} : \text{Test}_j)[ \overline{\text{test}(\text{yes})}\cdot\text{nil} ]
\]
\[
B^l_j \triangleq_B \beta(\text{inc} : \text{IReg}_j)\beta(\text{dec} : \text{DReg}_j)\beta(\text{ack}_i : \text{IAck}_j) \\
\beta(\text{ack}_d : \text{DAck}_j)[ \text{inc}(e)\cdot\text{Increment} \mid \text{Ack}_L \mid \text{Dec}_L ]
\]
where,
\[
\begin{align*}
DecL_l & \triangleq_p Decrement | \cdots | Decrement \\
AckL_l & \triangleq_p \text{ack}_d(e).\text{nil} | \cdots | \text{ack}_d(e).\text{nil} \\
Increment & \triangleq_p \text{ack}_d(e).\text{nil} | \text{ack}_i(e).\text{Decrement} \\
\text{Decrement} & \triangleq_p \text{dec}(e).\text{nil}
\end{align*}
\]

If \( l = 1 \) the process \( \text{AckList}_l \equiv_p \text{nil} \). Moreover, the processes \( \text{Increment} \) and \( \text{Decrement} \) are equal for the encoding of all the registers. The graphical representation of the bio-process \( B^0_j \) is

\[
(inc : \text{Test}_j) (test : I\text{Reg}_j) (ack : I\text{Ack}_j) (ack_d : D\text{Ack}_j)
\]

\[
\begin{array}{c}
inc(e).\text{nil} | \text{!test(no).nil} \\
\end{array}
\]

and the graphical representation of the bio-process \( B^\text{test}_i \parallel B^\text{value}_i \) is

\[
(\text{test} : \text{Test}_j) (inc : I\text{Reg}_j) (dec : D\text{Reg}_j) (ack_i : I\text{Ack}_j) (ack_d : D\text{Ack}_j)
\]

\[
\begin{array}{c}
\text{!test(yes).nil} \\
\text{!inc(e).Increment} | DecL_l | AckL_l \\
\end{array}
\]

The events \( \text{ZeroToOne}_j \) and \( \text{OneToZero}_j \), 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

\[
\begin{align*}
\text{ZeroToOne}_j & \triangleq_E (B^{\text{split0}}_j) \text{split}(B^{\text{test}}_j, B^{\text{split1}}_j) \\
\text{OneToZero}_j & \triangleq_E (B^{\text{test}}_j, B^{\text{join1}}_j) \text{join}(B^{\text{join0}}_j)
\end{align*}
\]

where,

\[
\begin{align*}
B^{\text{split0}}_j & \triangleq_B \beta(\text{test} : \text{Test}_j)\beta(\text{inc} : I\text{Reg}_j)\beta(\text{ack}_i : I\text{Ack}_j)\beta(\text{ack}_d : D\text{Ack}_j) \\
& [ \text{!test(no).nil} ] \\
B^{\text{join0}}_j & \triangleq_B \beta(\text{test} : \text{Test}_j)\beta(\text{inc} : I\text{Reg}_j)\beta(\text{ack}_i : I\text{Ack}_j)\beta(\text{ack}_d : D\text{Ack}_j) \\
& [ \text{ack}_d(e).\text{inc}(e).\text{nil} | \text{!test(no).nil} ] \\
B^{\text{split1}}_j & \triangleq_B \beta(\text{inc} : I\text{Reg}_j)\beta(\text{dec} : D\text{Reg}_j)\beta(\text{ack}_i : I\text{Ack}_j)\beta(\text{ack}_d : D\text{Ack}_j) \\
& [ \text{!inc(e).Increment} | \text{ack}_i(e).\text{Decrement} ] \\
B^{\text{join1}}_j & \triangleq_B \beta(\text{inc} : I\text{Reg}_j)\beta(\text{dec} : D\text{Reg}_j)\beta(\text{ack}_i : I\text{Ack}_j)\beta(\text{ack}_d : D\text{Ack}_j) \\
& [ \text{!inc(e).Increment} ]
\end{align*}
\]
The function $\alpha$ is defined in the following way:

$$
\alpha(\text{Type}_1, \text{Type}_2) = \begin{cases} 
1 & \text{if } \text{Type}_1 = \text{Type}_2 \\
0 & \text{otherwise}
\end{cases}
$$

The encoding produces a system $Z = \langle B, E \rangle$ in $Z^e$. 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 $\text{Switch}_j$. 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 modeling of the register $r_j$ depends on its content. If the content of the register is 0, then the box $B^0_j$ is used; if the content of the register is greater than zero, then bio-process $B^{\text{test}}_j \parallel B^1_j$ is used.

The box encoding the instruction $(i, \text{Incr}(r_j))$, after its activation, consumes an inter-communication with the box encoding the register $r_j$ (through the interfaces of type $I\text{Reg}_j$), representing a request for its increment; then the instruction box waits for another inter-communication (through the interfaces of type $I\text{Ack}_j$) 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 $I\text{Ins}$) 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 $B^{\text{split}}_j$, causing the activation of event $\text{ZeroToOne}_j$. This event substitutes $B^{\text{split}}_j$ with the bio-process $B^{\text{test}}_j \parallel B^{\text{split}}_j$. After consuming the acknowledgment inter-communication on the channel $\text{ack}_i$, the box becomes structurally congruent to the box $B^1_j$, indicating that the register has been correctly incremented. Notice that when the event $\text{ZeroToOne}_j$ 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 $\text{Increment}$, 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 $\text{ack}_i$. 
The box encoding the instruction \((i, \text{DecJump}(r_j, s))\), 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 \(\text{Test}_j\)). In particular, the instruction box receives a name \(\text{yes}\) if the content of the register \(r_j\) is greater than zero, receives the name \(\text{no}\) otherwise. With the choice operator two alternative behaviors are encoded, depending on the result of the testing communication. In case of \(\text{yes}\) 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 \(\text{DReg}_j\)), then waits for an acknowledgment indicating that the decrement has been executed (through the interfaces of types \(\text{DAck}_j\)) and finally replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of type \(\text{Ins}\)) for the activation of instruction \(i + 1\). Instead, in case of \(\text{no}\) name reception, the box simply replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of type \(\text{Ins}\)) 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 \(B^\text{join1}_j\). This box activates the event \(\text{OneToZero}_j\), which substitutes the bio-process \(B^\text{test} \parallel B^\text{join1}_j\) with the box \(B^\text{join0}_j\). After consuming the decrement inter-communication on the channel \(\text{ack}_d\), the box becomes structurally congruent to the box \(B^\text{0j}_j\), indicating that the register has been correctly decremented. Notice that, also in this case, when the event \(\text{OneToZero}_j\) 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 \(\text{AckList}_l\) 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 \([[(j, k_1, \ldots, k_n)]^c_R] \rightarrow^+ i_Z^+ Z\) and \(Z \equiv [[(j', k'_1, \ldots, k'_n)]^c_R]_R^c\).

**Proof.** The proof is by case analysis. There are five cases: (i) Instruction \(I_j = \text{DecJump}(r_1, s)\) and \(r_1\) value greater than one; (ii) Instruction \(I_j = \text{DecJump}(r_1, s)\) and \(r_1\) value equal to one; (iii) Instruction \(I_j = \text{DecJump}(r_1, s)\) and \(r_1\) value equal to zero; (iv) Instruction \(I_j = \text{Incr}(r_1)\) and \(r_1\) value greater
than zero; (v) Instruction $I_j = Incr(r_1)$ and $r_1$ value equal to zero. We prove only case (ii), because the other cases are similar.

(ii) We consider the computation of the bio-process $\langle (j, k_1, ..., k_n) \rangle_R^*$. As in Lemma 5.2 an inter-communication between the component $\text{Switch}_j$ and the component $\langle (j, DecJump(r_1, s)) \rangle_R^*$ is the first consumed action. The two boxes synchronize on output $\tau_j^L(e)$ and input $act(e)$ through their binders of type $Ins_j$. This cause the activation of the $\langle (j, DecJump(r_1, s)) \rangle_R^*$ 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 $B_{test}^1 || B_1^1$. The activation of the $\langle (j, DecJump(r_1, s)) \rangle_R^*$ box causes the consumption of an inter-communication between the box $B_{test}^1$ 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 $B_1^1$, consumes an inter-communication on output $\overline{dec(e)}$ through interface of type $DReg$ and remains blocked on input $ack(e)$; after that inter-communication, the box $B_1^1$ becomes structurally congruent to the box $B_{join}^{join1}$ and the event $OneToZero_l$ becomes active. The execution of the event substitutes the bio-process $B_{test}^1 || B_{join}^{join1}$ with the box $B_{join}^{join0}$, which consumes an intra-communication with the instruction box (on interfaces of type $DAck_i$) and becomes structurally congruent to $\langle r_1 = 0 \rangle_R^*$. 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 $next(\langle ins_{j+1} \rangle$.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 $\langle (j, DecJump(r_1, s)) \rangle_R^*$, while in the switching box the process

$$(\langle ins_{j+1} \rangle(e).nil | (\sum_{o=1}^{m} ins_o(e).x(e)\cdot \tau_o^L(e).nil))$$

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 $\text{Switch}_{j+1}$. □
**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 $Z = [(j, k_1, \ldots, k_n)]_R$ can produce a transition $Z \rightarrow Z_1$, then there exists a computation $Z \rightarrow Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_l$ such that $Z_l = [(j', k'_1, \ldots, k'_n)]_R$ and $(j, k_1, \ldots, k_n) \rightarrow_R (j', k'_1, \ldots, k'_n)$.

**Proof.** Consider the structure of the bio-process $Z = [(j, k_1, \ldots, k_n)]^{op}_R$. As in Lemma 5.3, if the bio-process $Z$ can perform a first step $Z \rightarrow Z_1$, 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 $I_j = DecJump(r, s)$ and $r_1$ value greater than one; (ii) Instruction $I_j = DecJump(r, s)$ and $r_1$ value equal to one; (iii) Instruction $I_j = DecJump(r, s)$ and $r_1$ value equal to zero; (iv) Instruction $I_j = Incr(r)$ and $r_1$ value greater than zero; (v) Instruction $I_j = Incr(r)$ and $r_1$ 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

\[
[(j, Incr(r))]_R = \beta(\text{act} : Ins_j) \beta(\text{next} : Ins) \beta(\text{inc} : IReg_l) \beta(\text{ack} : IAck_j) [lx(e).Incj | \text{act}(e).\text{inc}(e).\text{ack}(e).\text{x}(e).\text{next}I(\text{ins}_{j+1}).\text{nil}]
\]

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

\[
B' = \beta(\text{act} : Ins_j) \beta(\text{next} : Ins) \beta(\text{inc} : IReg_l) \beta(\text{ack} : IAck_j) [lx(e).Incj | \text{inc}(e).\text{ack}(e).\text{x}(e).\text{next}I(\text{ins}_{j+1}).\text{nil}]
\]

Now, the only possible action $Z_1 \rightarrow Z_2$ is the inter-communication between the box $B'$ and the box $[r_1 = 0]_R$ through their interfaces of type $IReg_l$ on output $\text{inc}(e)$ and input $\text{inc}(e)$, respectively. After the inter-communication the instruction box remains blocked on input $\text{ack}(e)$ over interface of type $IAck_l$ and the box encoding for the register $r_1$ becomes structurally congruent.
to the box $B^{split}_0$; the event $ZeroToOne_j$ is now active. The execution of the event, which correspond to the action $Z_2 \rightarrow Z_3$, substitutes in $Z_2$ the box $B^{split}_i$ with the bio-process $B^{test}_i \parallel B^{split}_1$. At this point, the action $Z_3 \rightarrow Z_4$ is an inter-communication between the box $B^{split}_1$ and the instruction box; the register box becomes structurally congruent to $[r_1 = 1]^e_R$, while the instruction box is unblocked and structurally congruent to

$$B'' = \beta(\text{act} : \text{Ins}_j)\beta(\text{next} : \text{Ins})\beta(\text{inc} : \text{IReg}_j)\beta(\text{ack} : \text{IAck}_j)
\left[!x(e).\text{Inc}_j \mid \overline{x}(e).\overline{\text{next}}(\text{ins}_j + 1).\text{nil}\right]$$

Now, the action $Z_4 \rightarrow Z_5$ is the intra-communication of $B''$ on channel $x$ which becomes a box $B''''$ with internal structure $!x(e).\text{Inc}_j \mid \overline{x}(e).\overline{\text{next}}(\text{ins}_j + 1).\text{nil}$, and the action $Z_5 \rightarrow Z_6$ is the inter-communication between the box $B''''$ and the switching box. After the inter-communication, the instruction box returns in its initial form $[(j, \text{Incr}(r_i))]^e_R$ and the switching box starts a sequence of intra-communications which produces a box structurally congruent to $\text{Switch}_{j+1}$ and representing the sequence of actions $Z_6 \rightarrow Z_7 \rightarrow Z_8$. It is easy to see that $Z_8$ is congruent to $[(j + 1, k_1, ..., k_{i-1}, 1, k_{i+1}, ..., k_n)]^e_R$. □

Lemma 5.7 and Lemma 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 $(1, I_1), ..., (m, I_m)$ and initial state $(j, k_1, ..., k_n)$. The the computation of the RAM $R$ terminates if and only if the computation of the system $Z = [(j, k_1, ..., k_n)]^e_R$ terminates.

**Proof.** The Theorem can be proved similarly to Theorem 5.4 and by using Lemma 5.7 and Lemma 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 $BL^e$. 43
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 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.

7 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 fruitful discussions about the topic and Alida Palmisano for reviewing preliminary versions of the document. Finally, we thank the anonymous referees for useful comments and suggestions.

References


