Math. Struct. in Comp. Science (2012), *vol.* 22, *pp.* 651–685. © Cambridge University Press 2012
doi:10.1017/S0960129511000739 First published online 8 May 2012

Static analysis of Biological Regulatory Networks dynamics using abstract interpretation

LOÏC PAULEVÉ[†], MORGAN MAGNIN[‡] and OLIVIER ROUX[‡]

[†]*LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597*
(Institut de Recherche en Communications et Cybernétique de Nantes),
1 rue de la Noë - B.P. 92101 - 44321 Nantes Cedex 3, France
and
LIX, École Polytechnique, 91128 Palaiseau Cedex, France
Email: loic.pauleve@ens-cachan.org
[‡]*LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597*
(Institut de Recherche en Communications et Cybernétique de Nantes),
1 rue de la Noë - B.P. 92101 - 44321 Nantes Cedex 3, France
Email: {morgan.magnin;olivier.roux}@irccyn.ec-nantes.fr

Received 7 March 2011; revised 12 December 2011

The analysis of the dynamics of Biological Regulatory Networks (BRNs) requires innovative methods to cope with the state-space explosion. This paper settles an original approach for deciding reachability properties based on *Process Hitting*, which is a framework suitable for modelling dynamical complex systems. In particular, Process Hitting has been shown to be of interest in providing compact models of the dynamics of BRNs with discrete values. Process Hitting splits a finite number of processes into so-called sorts and describes the way each process is able to act upon (that is, to 'hit') another one (or itself) in order to 'bounce' it as another process of the same sort with further actions.

By using complementary abstract interpretations of the succession of actions in Process Hitting, we build a very efficient static analysis to over- and under-approximate reachability properties, which avoids the need to build the underlying states graph. The analysis is proved to have a low theoretical complexity, in particular when the number of processes per sorts is limited, while a very large number of sorts can be managed.
This makes such an approach very promising for the scalable analysis of abstract complex systems. We illustrate this through the analysis of a large BRN of 94 components. Our method replies quasi-instantaneously to reachability questions, while standard model-checking techniques regularly fail because of the combinatoric explosion of behaviours.

1. Introduction

Biological Regulatory Networks (BRNs) are a common framework for modelling the concurrent regulatory mechanisms between biological components (RNA, proteins, and so on). These regulations are generally represented as interaction graphs, where nodes are components of the system, and edges state the regulation between them, which is either positive (activation) or negative (inhibition). Each node is also assigned a numerical value representing the state (for example, the concentration) of the component of the network, at a given time. This value evolves in response to the various regulations the component is subject to. In 1973, the biologist René Thomas proposed a formalisation of BRNs in which the values of the components are boolean (Thomas 1973). His formalisation uses an interaction graph and parameters (or, equivalently, boolean functions between nodes inputs) to describe the dynamics of a BRN. A full description of the BRN formalism with discrete values for components can be found in Bernot *et al.* (2007).

The derivation of dynamical properties from the interaction graph of a BRN has been the motivation underlying a variety of mathematical studies. Twenty years ago, René Thomas conjectured that the presence of positive circuits within the interaction graph is a necessary condition to achieve systems with multi-stationarity. The conjecture has been proved in several frameworks, notably in discrete dynamical systems (Richard and Comet 2007). By using more elaborated interaction graph analyses, the maximum number of fixed points within boolean networks can be characterised (Aracena 2008). Under strong conditions, particular fixed points (qualified as topological) can be fully extracted from the interaction graph (Paulevé and Richard 2010): these points are fixed in all possible dynamics matching the interaction graph. Finally, the presence of negative circuits in interaction graphs has been proved to be necessary for there to be sustained oscillations in the dynamics (Remy *et al.* 2008; Richard 2010).

To produce more precise analyses of BRN dynamics, we need to take into account the boolean functions specified together with the interaction graph (as in Bernot *et al.* (2008), for instance). The majority of current techniques use standard model-checking methods (Richard *et al.* 2006) based on the (explicit or symbolic) exploration of the state space of the model. Such methods suffer from a state-space explosion, and are intractable for large regulatory networks. We propose in this paper a novel and original method relying on the Process Hitting framework to address this scalability issue.

Process Hitting (Paulevé *et al.* 2011b) is a recently introduced framework that is suitable for modelling BRNs with discrete values. Basically, each discrete component value is modelled as a process; at any time, one and only one process of each component (referred to as a *sort*) is present and stands for the current state of the component. A sort changes process when it is *hit* by at most one other process. Static analyses have already been developed in the Process Hitting framework, notably for obtaining all the fixed points of dynamics of a Process Hitting (Paulevé *et al.* 2011b). Being a particular restriction of Communicating Finite-State Machines (Brand and Zafiropulo 1983), Process Hitting can be applied to less specific complex dynamical systems.

The static analysis by abstract interpretation (Cousot and Cousot 1977) aims to provide an efficient analysis of a program without executing it. This is achieved by constructing one, or several, sound abstractions of the semantics of the program; these abstractions are then interpreted to decide the validity of a given property, resulting in over- or under-approximations of the validity of the property in the concrete program. From now on, we will refer to this validity as the concretisability of the abstract property.

In this paper, we present a novel static analysis by abstract interpretation of Process Hittings. We address the decision of a successive reachability of processes. Our approach is based on two complementary abstractions of a succession of actions within a Process Hitting. Several refinement operators acting on these two abstractions are then defined. These refinements detail an abstraction, with the aim of simplifying the concretisability decision. We then use the abstraction refinement operators to develop over- and under-approximations of the process reachability decision. The implementations rely on the analysis of an abstract structure, which can be represented as a graph. We show that this abstract structure always has a reasonable size (that is, it is polynomial in the total number of processes); and, while its computation can be exponential in the number of processes within a single sort, the approximations are always linear or polynomial in its size.

The scalability of our approach is illustrated by its application to the decision of reachability of gene expression levels within a BRN of 94 components. Our methods produce very fast responses for the decision, while the well-established symbolic model checking technique SDD (Hamez *et al.* 2009) regularly fails because of the state-space explosion.

A preliminary version of these results was presented in Paulevé *et al.* (2011a). The current paper extends the results given in that paper in several ways: the general framework for presenting the analyses has been substantially reworked and unified; the conclusiveness of the overall method has been greatly improved; and the BRN application case has been extended from 40 to 94 components.

Structure of the paper

The Process Hitting framework is defined in Section 2. Section 3 gives a brief and simplified overview of the results presented in this paper. Section 4 presents complementary abstractions of scenarios (that is, sequences of actions) and defines the abstraction refinement operators. These refinement operators are then applied to the over- and under-approximation of process reachability in Section 5, and details of their implementation and complexity are given. Section 6 briefly presents the encoding of BRN dynamics into Process Hittings and applies the above methods to a large BRN relating 94 components. Finally, Section 7 summarises and discusses the contributions of this paper.

Notation

Given a finite *set* $S = \{e_1, \ldots, e_n\}$, we write $|S| \triangleq n$, and we write $\wp(S)$ to denote the power set.

Given a finite *sequence* of elements $A = e_1 :: \ldots :: e_n$:

— $|A| = n$ is the length of the sequence;
— $\mathbb{I}^A \triangleq \{1, \ldots, |A|\}$ is the set of A indexes;
— $\forall i \in \mathbb{I}^A, A_i = e_i$;
— ε is the empty sequence;
— $A_{i..j}$ is the subsequence $A_i, \ldots, A_j$;
— $A_{i..j} = \varepsilon$ if $j < i$.

$\mathrm{lfp}\{x_0\}\ (x \mapsto x')$ is the least fixpoint greater than x_0, if it exists, of the monotonic function $f(x) \triangleq x'$; and gfp f is the greatest fixpoint of f, if it exists.

2. The Process Hitting Framework

This section presents the *Process Hitting* framework (Paulevé *et al.* 2011b) on which the methods presented in this paper rely.

Process Hitting gathers a finite number of concurrent *processes* grouped into a finite set of *sorts*. A process belongs to one and only one sort and is denoted by a_i where a is the sort and i the identifier of the process within the sort a. At any time, one and only one process of each sort is present, forming a state of the Process Hitting.

The concurrent interactions between processes are defined by a set of *actions*. Actions describe the replacement of one process by another of the same sort conditional on the presence of at most one other process in the current state of the Process Hitting. An action is denoted by $a_i \rightarrow b_j \Rsh b_k$ where a_i, b_j, b_k are processes of sorts a and b. It is required that $b_j \neq b_k$ and that $a = b \Rightarrow a_i = b_j$. An action $h = a_i \rightarrow b_j \Rsh b_k$ is read as 'a_i *hits* b_j to make it bounce to b_k', and a_i, b_j, b_k are called the *hitter*, *target* and *bounce* of the action, respectively, and will be denoted by $\mathrm{hitter}(h), \mathrm{target}(h), \mathrm{bounce}(h)$, respectively.

Definition 2.1 (Process Hitting). A *Process Hitting* is a triple $(\Sigma, L, \mathscr{H})$:

— $\Sigma = \{a, b, \dots\}$ is the finite countable set of sorts.

— $L = \prod_{a \in \Sigma} L_a$ is the set of states with $L_a = \{a_0 \dots a_{l_a}\}$ the finite and countable set of processes of sort $a \in \Sigma$ and l_a a positive integer with

$$a \neq b \Rightarrow \forall (a_i, b_j) \in L_a \times L_b, a_i \neq b_j.$$

— The finite set of actions is

$$\mathscr{H} = \{a_i \rightarrow b_j \Rsh b_k, \cdots \mid (a, b) \in \Sigma^2,\ (a_i, b_j, b_k) \in L_a \times L_b \times L_b, b_j \neq b_k, a = b \Rightarrow a_i = b_j\}.$$

Proc denotes the set of all processes ($\mathbf{Proc} = \{a_i \mid a \in \Sigma \wedge a_i \in L_a\}$).

We write $\Sigma(a_i) = a$ to denote the sort of a process a_i, and

$$\Sigma(h) = \{\Sigma(\mathrm{hitter}(h)), \Sigma(\mathrm{target}(h))\}$$

to denote the set of sorts present in an action $h \in \mathscr{H}$. Given a state $s \in L$, the process of sort $a \in \Sigma$ present in s is denoted by $s[a]$, which is the a-coordinate of the state s. We define the following notation:

— if $a_i \in L_a$, $a_i \in s \overset{\Delta}{\Leftrightarrow} s[a] = a_i$; and

— if $ps \in \wp(Procs)$, $ps \subseteq s \overset{\Delta}{\Leftrightarrow} \forall a_i \in ps, a_i \in s$.

An action $h = a_i \rightarrow b_j \Rsh b_k \in \mathscr{H}$ is *playable* in $s \in L$ if and only if $s[a] = a_i$ and $s[b] = b_j$. In such a case, $(s \cdot h)$ stands for the state resulting from the play of the action h in s, that is, $(s \cdot h)[b] = b_k$ and $\forall c \in \Sigma, c \neq b, (s \cdot h)[c] = s[c]$. For clarity, $((s \cdot h) \cdot h')$, $h' \in \mathscr{H}$ is abbreviated as $(s \cdot h \cdot h')$.

If A is a sequence of actions, the set of sorts present in A is given by $\Sigma(A) = \bigcup_{n \in \mathbb{I}^A} \Sigma(A_n)$. We write $\mathrm{fst}_a(A)$ to denote first process of sort a appearing in the sequence, and $\mathrm{last}_a(A)$

for the last one:

$$\mathrm{fst}_a(A) \triangleq \begin{cases} \emptyset & \text{if } a \notin \Sigma(A) \\ \mathrm{hitter}(A_m) & \text{if } m = \min\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathrm{hitter}(A_m)) = a \\ \mathrm{target}(A_m) & \text{if } m = \min\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathrm{target}(A_m)) = a \end{cases} \tag{1}$$

$$\mathrm{last}_a(A) \triangleq \begin{cases} \emptyset & \text{if } a \notin \Sigma(A), \\ \mathrm{bounce}(A_m) & \text{if } m = \max\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathrm{bounce}(A_m)) = a \\ \mathrm{hitter}(A_m) & \text{if } m = \max\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathrm{hitter}(A_m)) = a. \end{cases} \tag{2}$$

Among the sequences of actions, the particular sequences that are only composed of successively playable actions form *scenarios* (see Definition 2.2). A scenario δ is said to be playable in a state $s \in L$ if and only if δ_1 is playable in s and for all $n \in \mathbb{I}^\delta, n < |\delta|$, we have δ_{n+1} is playable in the state $(s \cdot \delta_1 \cdot \ldots \cdot \delta_n)$; or, equivalently, δ is playable in s if and only if support$(\delta) \subseteq s$, where support(δ) is the set of processes that are the first ones appearing in δ for their respective sort (see Equation (3)). If it is playable, the state resulting from the sequential play of the scenario in s is denoted by $s \cdot \delta$. During the play of δ, only target processes change (this set is given by $\{p \in \mathbf{Proc} \mid \exists n \in \mathbb{I}^\delta, p = \mathrm{target}(\delta_n)\}$). It is easy to show that

$$\forall a_i \in \mathrm{end}(\delta), (s \cdot \delta)[a] = a_i$$
$$\forall b \in \Sigma \setminus \Sigma(\delta), (s \cdot \delta)[b] = s[b];$$

where end(δ) is defined in Equation (4).

Definition 2.2 (Scenario (Sce)). Given a Process Hitting $(\Sigma, L, \mathscr{H})$, a *scenario* δ is a sequence of actions in $\mathscr{H}$ such that for all $n \in \mathbb{I}^\delta$, $a_i = \mathrm{hitter}(\delta_n)$ (respectively, target(δ_n)) $\Rightarrow$ last$_a(\delta_{1..n-1}) \in \{\emptyset, a_i\}$. The set of all scenarios is denoted by **Sce**. The functions support(δ) and end(δ) give the first and last processes of each sort, respectively:

$$\mathrm{support}(\delta) \triangleq \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = \mathrm{fst}_{\Sigma(p)}(\delta)\} \tag{3}$$

$$\mathrm{end}(\delta) \triangleq \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = \mathrm{last}_{\Sigma(p)}(\delta)\}. \tag{4}$$

Example 2.3. Figure 1 represents a Process Hitting $(\Sigma, L, \mathscr{H})$ where

$$\Sigma = \{a, b, c, d\}$$
$$L = \{a_0, a_1\} \times \{b_0, b_1, b_2\} \times \{c_0, c_1\} \times \{d_0, d_1, d_2\}$$
$$\mathscr{H} = \{a_0 \rightarrow c_0 \Rsh c_1, a_1 \rightarrow b_1 \Rsh b_0, c_1 \rightarrow b_0 \Rsh b_1, b_1 \rightarrow a_0 \Rsh a_1,$$
$$b_0 \rightarrow d_0 \Rsh d_1, b_1 \rightarrow d_1 \Rsh d_2, d_1 \rightarrow b_0 \Rsh b_2, c_1 \rightarrow d_1 \Rsh d_0, b_2 \rightarrow d_0 \Rsh d_2\}.$$

Playing the action $b_1 \rightarrow a_0 \Rsh a_1$ in the state $\langle a_0, b_1, c_0, d_0 \rangle$ results in the state $\langle a_1, b_1, c_0, d_0 \rangle$. Then

$$\delta = a_0 \rightarrow c_0 \Rsh c_1 :: b_1 \rightarrow a_0 \Rsh a_1 :: a_1 \rightarrow b_1 \Rsh b_0 :: b_0 \rightarrow d_0 \Rsh d_1 :: d_1 \rightarrow b_0 \Rsh b_2$$

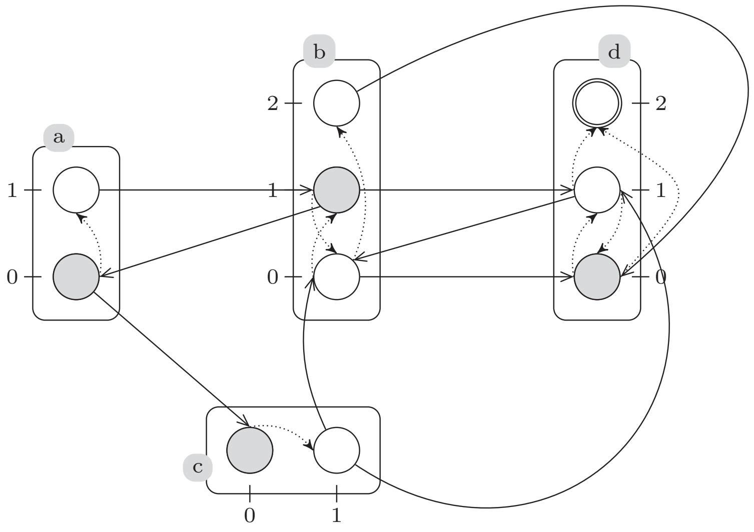

Figure 1. A Process Hitting example: sorts are represented by labelled boxes, and processes by circles (ticks are the identifiers of the processes within the sort, for instance, a_0 is the process ticked 0 in the box a). An action (for instance $a_0 \rightarrow c_0 \upharpoonright c_1$) is represented by a pair of directed arcs, having the hit part (a_0 to c_0) in plain line and the bounce part (c_0 to c_1) in dotted line. The reachability of the process d_2 (double circled) is studied in next sections. The current state is represented by the grayed processes: $\langle a_0, b_1, c_0, d_0 \rangle$.

is a scenario playable in the state

$$s = \langle a_0, b_1, c_0, d_0 \rangle$$
$$s \cdot \delta = \langle a_1, b_2, c_1, d_1 \rangle$$

with

$$\text{support}(\delta) = \{a_0, b_1, c_0, d_0\}$$
$$\text{end}(\delta) = \{a_1, b_2, c_1, d_1\}.$$

Remark 2.4. The Process Hitting framework can be considered as a class of Communicating Finite-State Machines (Brand and Zafiropulo 1983) where at most two machines (sorts) share a synchronisation label (action) and one and only one machine changes its state (process) at each synchronisation (action play).

3. Overview of the results obtained

In this section we give a brief overview of the results obtained on the static analysis of reachability properties using Process Hitting. For simplicity, we focus on the practical use of the analyses developed, skipping the detailed definitions and proofs, which will be given in the following sections.

3.1. *General approach*

We define the Process Hitting $\mathcal{PH} = (\Sigma, L, \mathcal{H})$ on which we want to check the successive reachability property $\mathcal{R}$ as follows: given a state $s \in L$, there exists a scenario such that whenever some actions are performed, the process a_i is present, then, after some other actions, the process b_j is eventually present, and so on. This can be compared with CTL (Clarke and Emerson 1981), with a recursive EF (exists eventually) property where atoms simply check the presence of a given process.

In the context of Process Hitting, the current paper proves some properties, say $\mathcal{P}$ and $\mathcal{Q}$, respectively, allowing the over- and under-approximation of the property $\mathcal{R}$:

— *If $\mathcal{PH}$ does not verify $\mathcal{P}$, the reachability $\mathcal{R}$ is impossible* (over-approximation of $\mathcal{R}$).

— *If $\mathcal{PH}$ verifies $\mathcal{Q}$, the reachability $\mathcal{R}$ is possible* (under-approximation of $\mathcal{R}$).

The properties $\mathcal{P}$ and $\mathcal{Q}$ can be checked statically and efficiently, allowing a fast over- and under-approximation of $\mathcal{R}$. We will now give a brief sketch of these properties.

3.2. *Complementary abstractions of scenarios*

We develop two complementary scenario abstractions in Section 4. Both are based on the notion of an *objective*, which specifies the bounce (after some actions) from one process to another of the same sort (for example, $d_0 \Rsh^* d_2$ is an objective).

The first abstraction, which is called the *objective sequences*, gives a sparse description of the successive reachability of processes within a scenario, and can be used to stand for the reachability property $\mathcal{R}$. For example, $b_0 \Rsh^* b_2 :: d_0 \Rsh^* d_2$ abstracts all scenarios such that, starting with b_0 and d_0 present, and after some actions, b_2 is present, and then, after some actions, d_2 is present, with any other actions taking place in between ignored.

The second abstraction, which is called the *bounce sequences*, describes the *local* actions resolving a given objective (that is, hitting processes of the sort of the objective). For example, $b_0 \rightarrow d_0 \Rsh d_1 :: b_1 \rightarrow d_1 \Rsh d_2$ is a bounce sequence for the objective $d_0 \Rsh^* d_2$ and abstracts any scenario using these actions in the same order. The results of this paper mainly use *abstract bounce sequences*, which are the set of hitters of the bounce sequence (for example, $\{b_0, b_1\}$ for the previous example), thereby abstracting the order of actions. An interesting feature of such abstractions is that they are statically completely computable from the definition of the Process Hitting, thus providing the necessary (local) actions for resolving any given objective.

Such abstractions are complementary in the sense that they are concerned with different information, which may be combined to *refine* a given abstraction, that is, to incorporate some additional knowledge into it. Basically, while objective sequences describe successions of objectives to perform, bounce sequences describe the necessary steps to achieve such objectives. The refinement operators (see Section 4.4) take advantage of such information to refine a given objective sequence by merging into it the necessary intermediate objectives given by the associated bounce sequences.

3.3. *Abstract structures of process hitting*

In order to give an efficient representation of the relations between the various objectives and their associated bounce sequences, we will give several graphical representations describing the abstract structures of a given Process Hitting.

Figure 2 gives two instances of such graphs for the Process Hitting drawn in Figure 1. These graphs relate three types of nodes: *processes* (square nodes), *objectives* and *solutions* (circles) representing an abstracted bounce sequence. In particular:

— processes are related to the objectives describing their reachability;
— objectives are related to solutions corresponding to associated abstract bounce sequences (objectives can also be related to other objectives sharing the same bounces); and
— solutions are related to processes composing the associated abstract bounce sequence.

Objectives without solutions are annotated with the $\perp$ symbol.

Such structures have the advantage of having a limited size with a low complexity for their construction: polynomial in the number of processes and exponential in the number of processes within one sort (which is expected to be very small).

In this paper, we use the two abstract structures $\mathscr{A}$ and $\lceil\mathscr{B}\rceil$ to compute $\mathscr{P}$ (over-approximation) and $\mathscr{Q}$ (under-approximation) properties, respectively. The construction of $\mathscr{A}$ is recursive: starting from processes to which the reachability is specified by $\mathscr{R}$, we relate processes to the objective reaching them from the initial state (imposed by $\mathscr{R}$); we then compute all the solutions for each objective (that is, the associated abstract bounce sequences). The construction of $\lceil\mathscr{B}\rceil$ is similar, except that some solutions may be ignored: if a process a_i is referenced in $\lceil\mathscr{B}\rceil$, all the processes of sort a in $\lceil\mathscr{B}\rceil$ are related to an objective starting from a_i.

The top and bottom diagrams in Figure 2(top) show the structures $\mathscr{A}$ and $\lceil\mathscr{B}\rceil$, respectively, computed from the Process Hitting drawn in Figure 1 for the reachability of process d_2 from the initial state $\langle a_1, b_0, c_0, d_1\rangle$ for $\mathscr{A}$ and $\langle a_1, b_1, c_1, d_0\rangle$ for $\lceil\mathscr{B}\rceil$.

3.4. *Main static properties for successive reachability analysis*

Using the previously introduced abstract structures, we will describe in Section 5 the three over-approximation properties $\mathscr{P}_1$, $\mathscr{P}_2$, $\mathscr{P}_3$ ($\mathscr{P} = \mathscr{P}_1 \vee \mathscr{P}_2 \vee \mathscr{P}_3$) and the two under-approximation properties $\mathscr{Q}_1$, $\mathscr{Q}_2$ ($\mathscr{Q} = \mathscr{Q}_1 \vee \mathscr{Q}_2$) developed in this paper for deciding the possibility of the successive reachability property $\mathscr{R}$.

— $\mathscr{P}_1$ *(see Theorem 5.7): there exists a traversal of* $\mathscr{A}$ *starting from each process in* $\mathscr{R}$ *such that: there is no loop; and from an objective node, at least one related solution is traversed, and from any other node, all related nodes are traversed.*

The top diagram in Figure 2 gives an example of an abstract structure $\mathscr{A}$ that does not verify $\mathscr{P}_1$: the reachability is then proved to be impossible.

— $\mathscr{Q}_1$ *(see Theorem 5.29):* $\lceil\mathscr{B}\rceil$ *has no cycle and all its leaves are solution nodes.*

The bottom diagram in Figure 2 shows an example of an abstract structure $\lceil\mathscr{B}\rceil$ that verifies $\mathscr{Q}_1$: the reachability is then proved to be possible.

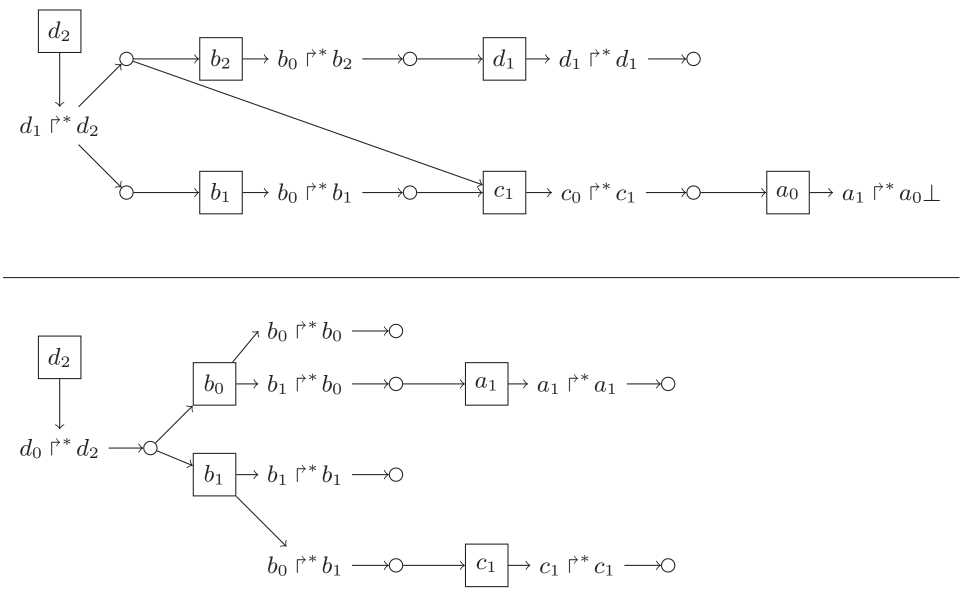

Figure 2. The top diagram shows the abstract structure $\mathscr{A}$ of the Process Hitting in Figure 1 for the reachability of d_2 from the state $\langle a_1, b_0, c_0, d_1 \rangle$. The bottom diagram shows the abstract structure $\lceil \mathscr{B} \rceil$ for the reachability of d_2 from the state $\langle a_1, b_1, c_1, d_0 \rangle$.

Properties $\mathscr{P}_1$ and $\mathscr{Q}_1$ are said to be *unordered* since the order of the requested reachabilities in $\mathscr{R}$ is ignored. Properties $\mathscr{P}_2$ and $\mathscr{Q}_2$ take advantage of this sequentiality through an iterative reasoning procedure. If $\mathscr{R}_1$ denotes $\mathscr{R}$ truncated at its first reachability and $\mathscr{R}_{2..}$ denotes the remaining reachabilities (with a modified initial state, which we do not describe here), we have the following results:

— *$\mathscr{P}_2$ (see Theorem 5.15): $\mathscr{P}_1$ is verified with $\mathscr{R}_1$ and $\mathscr{P}_2$ is verified with $\mathscr{R}_{2..}$.*
— *$\mathscr{Q}_2$ (see Corollary 5.32: $\mathscr{Q}_1$ is verified with $\mathscr{R}_1$ and $\mathscr{Q}_2$ is verified with $\mathscr{R}_{2..}$.*

Finally, in the case of the over-approximation of $\mathscr{R}$, a property $\mathscr{P}_3$ exploits ordering constraints between occurrences of processes that are statically extracted from the Process Hitting. We write $q \lhd p$ if the process q cannot occur after p:

— *$\mathscr{P}_3$ (see Theorem 5.23): For all $n < m$, for each $p \in \mathrm{minProc}(\mathscr{R}_{n..})$ and $q \in \mathrm{minProc}(\mathscr{R}_{m..})$, we have $\neg(q \lhd p)$,*
where minProc denotes the processes necessary for achieving $\mathscr{R}_{n..}$ (respectively, $\mathscr{R}_{m..}$).

The computation of $\mathscr{P}$ also provides a set of so-called *key processes* for satisfying $\mathscr{R}$. If z_l is such a key process, and $\mathscr{PH} \setminus z_l$ denotes the Process Hitting $\mathscr{PH}$ without the actions related to z_l, then:

— *If $\mathscr{PH}$ verifies $\mathscr{P}$, then $\mathscr{PH} \setminus z_l$ does not satisfy $\mathscr{P}$.*

Such a property is very interesting in the context of controlling the reachability $\mathscr{R}$: disabling a key process ensures that no scenario can concretise $\mathscr{R}$.

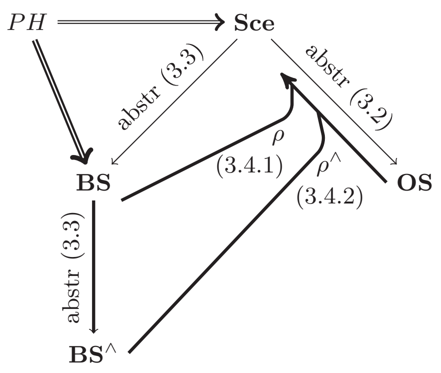

Figure 3. Derivation relations between a Process Hitting (PH), scenarios (**Sce**), objective sequences (**OS** – see Section 4.2), bounce sequences (**BS** and **BS**$^\wedge$ – see Section 4.3) and refinement operators (for example, ρ and $\rho^\wedge$ – see Section 4.4). The relations shown with thick lines are used in Section 5 to decide the concretisability of an objective sequence.

As for the complexity, these properties can be checked in time polynomial in the size of the abstract structures. In the case of the Q_1 (under-approximation), a more conclusive checking can be achieved, but at the cost of being exponential in the number of solutions for a single objective.

Therefore, we expect these new analyses will be very efficient using Process Hitting since they have a limited number of processes per sort, but a very large number of sorts.

In the folowing sections, we will specify successive reachability properties by objective sequences, and extend the notion of an initial state to the notion of a context (a set of initial states).

4. Abstract interpretation of scenarios

Having introduced the preliminary definitions in Section 4.1, we will now establish two orthogonal abstractions of scenarios using *objective sequences* (see Section 4.2) and *bounce sequences* (see Section 4.3). The first of these abstractions describes a succession of process changes per sort (called objectives), and the second describes the actions actually played to resolve these objectives. While objective sequences can be seen as a sparse representation of a scenario, bounce sequences emphasise the necessary actions required to resolve an objective.

Using these two complementary abstractions, we will derive several objective sequence refinement operators in Section 4.4. The aim of these refinements is to provide more precise abstractions through which the concretisability may be easier to decide. Figure 3 summarises the possible derivations between a Process Hitting and the different scenario representations.

4.1. *Preliminaries*

This section introduces the notions of an *objective* and a *context* that we will use in developing the abstractions.

Definition 4.1 (objective (Obj)). The reachability of process a_j from a process a_i is called an *objective*, and is denoted by $a_i {\Rsh}^* a_j$. The set of objectives is denoted by

$$\mathbf{Obj} \triangleq \{a_i {\Rsh}^* a_j \mid a \in \Sigma \wedge (a_i, a_j) \in L_a^2\}.$$

Given an objective $P \in \mathbf{Obj}$, where $P = a_i {\Rsh}^* a_j$, we have

$$\begin{aligned} \Sigma(P) &= a \\ \mathrm{target}(P) &= a_i \\ \mathrm{bounce}(P) &= a_j. \end{aligned}$$

An objective P is *trivial* if $\mathrm{target}(P) = \mathrm{bounce}(P)$.

We will now extend the notion of a state to the notion of a context. A context references the set of processes per sort that can serve as an initial state.

Definition 4.2 (context ς (Ctx)). A *context* ς associates with each sort in Σ a non-empty subset of its processes:

$$\forall a \in \Sigma, \varsigma[a] \subseteq L_a \wedge \varsigma[a] \neq \varnothing.$$

We use **Ctx** to denote the set of contexts.

Given a context ς, we write $a_i \in \varsigma \overset{\Delta}{\Leftrightarrow} a_i \in \varsigma[a]$, and, moreover, given $ps \in \wp(\mathbf{Proc})$, we write $ps \subseteq \varsigma \overset{\Delta}{\Leftrightarrow} \forall a_i \in ps, a_i \in \varsigma$. The override of a context ς by a set of processes ps is denoted by $\varsigma \Cap ps$ (see Definition 4.3). For instance,

$$\langle a_1, a_2, b_1, c_1\rangle \Cap \{a_3, b_2, b_3\} = \langle a_3, b_2, b_3, c_1\rangle.$$

Definition 4.3 ($\Cap$: $\mathbf{Ctx} \times \wp(\mathbf{Proc}) \mapsto \mathbf{Ctx}$). Given a context $\varsigma \in \mathbf{Ctx}$ and $ps \in \wp(\mathbf{Proc})$, the override of ς by ps is denoted by $\varsigma \Cap ps$ and is defined by

$$\forall a \in \Sigma, (\varsigma \Cap ps)[a] \triangleq \begin{cases} \{p \in ps \mid \Sigma(p) = a\} & \text{if } \exists p \in ps, \Sigma(p) = a \\ \varsigma[a] & \text{otherwise.} \end{cases}$$

A scenario $\delta \in \mathbf{Sce}$ is playable in the context ς if and only if $\mathrm{support}(\delta) \subseteq \varsigma$. The play of δ in ς is denoted by $\varsigma \cdot \delta$ where $\varsigma \cdot \delta = \varsigma \Cap \mathrm{end}(\delta)$.

Example 4.4 (Process Hitting in Figure 1). Given the context

$$\varsigma = \langle a_0, b_0, b_1, c_0, d_1\rangle$$

and the scenario

$$\delta = a_0 {\rightarrow} c_0 {\Rsh} c_1 :: b_1 {\rightarrow} a_0 {\Rsh} a_1 :: b_1 {\rightarrow} d_1 {\Rsh} d_2,$$

δ is playable in ς and

$$\varsigma \cdot \delta = \langle a_0, b_0, b_1, c_0, d_1\rangle \Cap \{a_1, b_1, c_1, d_2\} = \langle a_1, b_1, c_1, d_2\rangle.$$

4.2. *Abstraction of scenarios into objective sequences*

During the execution of a scenario, processes of different sorts bounce one after the other, following the play of the actions. An abstraction of such an execution is a succession of objectives: the process a_j is reached (after a certain number of actions) from a_i, then the process b_j is reached from b_i, and so on. This forms an objective sequence (see Definition 4.5). The appending of an objective to an objective sequence is defined in Definition 4.6.

Definition 4.5 (objective sequence (OS)). An *objective sequence* is a sequence $\omega = P_1 :: \ldots :: P_{|\omega|}$, where:

$$\forall n \in \mathbb{I}^{\omega}, \omega_n \in \mathbf{Obj}$$

and

$$a_i = \text{target}(\omega_n) \Rightarrow \text{last}_a(\omega_{1..n-1}) \in \{\emptyset, a_i\}.$$

The set of objective sequences is denoted by **OS**. The definitions of last_a (see Equation (2)), fst_a (see Equation (1)), support (see Equation (3)) and end (see Equation (4)) are straightforwardly extended to objective sequences by omitting the hitter case.

Definition 4.6 ($\oplus$: OS $\times$ Obj $\mapsto$ OS). The join between an objective sequence $\omega \in \mathbf{OS}$ and an objective $a_i \Rsh^* a_j \in \mathbf{Obj}$ is defined by

$$\omega \oplus a_i \Rsh^* a_j \triangleq \begin{cases} \omega :: a_i \Rsh^* a_j & \text{if } a \notin \Sigma(\omega) \\ \omega :: \text{last}_a(\omega) \Rsh^* a_j & \text{otherwise.} \end{cases}$$

Example 4.7. The sequence

$$\omega = a_0 \Rsh^* a_1 :: b_0 \Rsh^* b_2 :: d_0 \Rsh^* d_1$$

is an objective sequence where

$$\begin{aligned} \text{support}(\omega) &= \{a_0, b_0, d_0\} \\ \text{end}(\omega) &= \{a_1, b_2, d_1\} \\ \omega \oplus d_0 \Rsh^* d_2 &= \omega :: d_1 \Rsh^* d_2. \end{aligned}$$

A scenario can be abstracted by several objective sequences, which describe process changes more or less sparsely. For instance, the scenario

$$a_0 \rightarrow \mathbf{c}_0 \Rsh \mathbf{c}_1 :: b_1 \rightarrow \mathbf{a}_0 \Rsh \mathbf{a}_1 :: a_1 \rightarrow \mathbf{b}_1 \Rsh \mathbf{b}_0 :: b_0 \rightarrow \mathbf{d}_0 \Rsh \mathbf{d}_1 :: d_1 \rightarrow \mathbf{b}_0 \Rsh \mathbf{b}_2$$

can be abstracted to

$$c_0 \Rsh^* c_1 :: \mathbf{a}_0 \Rsh^* \mathbf{a}_1 :: \mathbf{b}_1 \Rsh^* b_0 :: d_0 \Rsh^* d_1 :: b_0 \Rsh^* \mathbf{b}_2;$$

or, more sparsely, to

$$a_0 \Rsh^* a_1 :: \mathbf{b}_1 \Rsh^* \mathbf{b}_2;$$

or to

$$b_1 \Rsh^* b_2;$$

and so on (processes written in bold are the ones kept in the succeeding abstraction).

The set of scenarios concretising an objective sequence ω in a context ς is given by $\gamma_\varsigma(\omega)$ (see Definition 4.8). We can also define the concretisation of a set of objective sequences as the union of their concretisations (see Definition 4.9).

Definition 4.8 ($\gamma_\varsigma : \mathbf{OS} \mapsto \wp(\mathbf{Sce})$)**.** Given $\omega \in \mathbf{OS}$, $\gamma_\varsigma(\omega)$ is the set of scenarios concretising ω in the context ς:

$$\begin{aligned}\gamma_\varsigma(\omega) \triangleq \{\delta \in \mathbf{Sce} \mid & (\omega^\triangle = \varepsilon \wedge \delta = \varepsilon) \vee (\omega^\triangle \neq \varepsilon \wedge \mathrm{support}(\delta) \subseteq \varsigma \\ & \wedge \exists \phi : \mathbb{I}^\omega \mapsto \mathbb{I}^\delta, (\forall n, m \in \mathbb{I}^\omega, n < m \Leftrightarrow \phi(n) \leqslant \phi(m)) \\ & \wedge \forall n \in \mathbb{I}^\omega, \mathrm{bounce}(\omega_n) \in \varsigma \cdot \delta_{1..\phi(n)})\}\end{aligned}$$

where we write $\omega^\triangle$ for the objective sequence ω with the trivial objectives removed.

Definition 4.9 ($\gamma_\varsigma : \wp(\mathbf{OS}) \mapsto \wp(\mathbf{Sce})$)**.** $\gamma_\varsigma(\Omega) \triangleq \{\delta \in \gamma_\varsigma(\omega) \mid \omega \in \Omega\}$.

Note that the concretisation of an objective sequence ω does not depend on its support support(ω) since it is imposed by the context. In this way, we use $\star \Rsh^* a_i$ to denote explicitly the fact that the target of the objective can be any process of sort a present in the context:

$$\forall a_j \in \varsigma[a], \gamma_\varsigma(a_j \Rsh^* a_i) = \gamma_\varsigma(\star \Rsh^* a_i). \qquad (5)$$

Example 4.10 (Process Hitting in Figure 1). Given the context

$$\varsigma = \langle a_0, b_0, b_1, c_1, d_1 \rangle$$

and the objective sequence $\omega = \star \Rsh^* b_2 :: \star \Rsh^* d_2$, the following is an extract of the (infinite) set of concrete scenarios:

$$\begin{aligned}\gamma_\varsigma(\omega) = \{ & d_1 \to b_0 \Rsh b_2 :: c_1 \to d_1 \Rsh d_0 :: b_2 \to d_0 \Rsh d_2, \\ & b_1 \to a_0 \Rsh a_1 :: a_1 \to b_1 \Rsh b_0 :: d_1 \to b_0 \Rsh b_2 :: c_1 \to d_1 \Rsh d_0 :: b_2 \to d_0 \Rsh d_2, \\ & c_1 \to b_0 \Rsh b_1 :: b_1 \to a_0 \Rsh a_1 :: a_1 \to b_1 \Rsh b_0 :: d_1 \to b_0 \Rsh b_2 :: c_1 \to d_1 \Rsh d_0 :: b_2 \to d_0 \Rsh d_2, \\ & \ldots\}.\end{aligned}$$

Given the context $\varsigma = \langle a_0, b_1, b_2, c_0, d_2 \rangle$, we get $\gamma_\varsigma(\star \Rsh^* c_1 :: \star \Rsh^* a_0) = \emptyset$.

Finally, we use α_ς to denote the reverse operation of concretisation of a set of scenarios (see Definition 4.11): the abstraction of a scenario δ is the set of objective sequences ω for which δ is a concretisation ($\delta \in \gamma_\varsigma(\omega)$).

Definition 4.11 ($\alpha_\varsigma : \wp(\mathbf{Sce}) \mapsto \wp(\mathbf{OS})$)**.**

$$\alpha_\varsigma(\Delta) \triangleq \{\omega \in \mathbf{OS} \mid \exists \delta \in \Delta, \delta \in \gamma_\varsigma(\omega)\}.$$

The relation between γ_ς and α_ς is emphasised by Property 4.12: the concretisation of the abstraction of a set of scenarios Δ includes Δ. This relation gives an abstract interpretation framework (Cousot and Cousot 1992) that is sufficient for the results presented in this paper.

Property 4.12. $\forall \Delta \in \wp(\mathbf{Sce}), \Delta \subseteq \gamma_\varsigma(\alpha_\varsigma(\Delta))$.

Proof. $\forall \delta \in \Delta$, there exists $\omega \in \mathbf{OS}$ such that $\delta \in \gamma_\varsigma(\omega)$ (see Definition 4.8). Hence, $\omega \in \alpha_\varsigma(\Delta)$ (see Definition 4.11), therefore $\delta \in \gamma_\varsigma(\alpha_\varsigma(\Delta))$. □

4.3. *Abstraction of scenarios into bounce sequences*

Bounce sequences result from a local reasoning on a single sort a. Bouncing from a_i to a_j (that is, resolving the objective $a_i \Rsh^* a_j$) may require the play of several actions on processes of sort a, forming a *bounce sequence* (see Definition 4.13). Note that bounce sequences are generally not scenarios: for example, $b_i \rightarrow a_i \Rsh a_j :: b_j \rightarrow a_j \Rsh a_k$ is a bounce sequence but not a scenario if $b_i \neq b_j$.

The targets and bounces of all actions in a bounce sequence ζ share the same sort $\Sigma(\zeta)$. We will not consider bounce sequences that contain cycles between targets and bounces of actions. In this way, the maximum length of a bounce sequence for a sort a is the number of processes of sort a.

Definition 4.13 (bounce sequence (BS)). A *bounce sequence* ζ is a sequence of actions such that $\forall n \in \mathbb{I}^\zeta, n < |\zeta|, \text{bounce}(\zeta_n) = \text{target}(\zeta_{n+1})$. We will write **BS** to denote the set of bounce sequences, and $\mathbf{BS}(P)$ to denote the set of bounce sequences *resolving* the objective P:

$$\mathbf{BS}(a_i \Rsh^* a_j) \stackrel{\Delta}{=} \{\zeta \in \mathbf{BS} \mid \text{target}(\zeta_1) = a_i \wedge \text{bounce}(\zeta_{|\zeta|}) = a_j \\ \wedge \forall m, n \in \mathbb{I}^\zeta, n > m, \text{bounce}(\zeta_n) \neq \text{target}(\zeta_m)\}.$$

It is obvious that

$$\mathbf{BS}(a_i \Rsh^* a_i) = \{\varepsilon\},$$

and that

$$\mathbf{BS}(a_i \Rsh^* a_j) = \emptyset$$

if there is no possibility of reaching a_j from a_i.

The full set of bounce sequences can be computed directly from the set of actions $\mathscr{H}$ of the Process Hitting without any enumeration of scenarios. Given an objective $a_i \Rsh^* a_j$, the computation of bounce sequences $\mathbf{BS}(a_i \Rsh^* a_j)$ (see Definition 4.13) works by a depth-first research between actions on the sort a to form a bounce sequence without cycles. Such a computation is exponential in the number of actions on the sort a, and is thus efficient when this number is small compared with the total number of actions.

We will also consider a sparse representation of a bounce sequence ζ resolving an objective P by considering the set of hitters of its actions that have a different sort than that of P. We write $\mathbf{BS}^\wedge(P)$ to denote the set of such abstracted bounce sequences.

Definition 4.14 ($\mathbf{BS}^\wedge : \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$).

$$\mathbf{BS}^\wedge(P) \stackrel{\Delta}{=} \{\zeta^\wedge \mid \zeta \in \mathbf{BS}(P) \wedge \nexists \zeta' \in \mathbf{BS}(P), \zeta'^\wedge \subsetneq \zeta^\wedge\}$$

where

$$\zeta^\wedge \stackrel{\Delta}{=} \{\text{hitter}(\zeta_n) \mid n \in \mathbb{I}^\zeta \wedge \Sigma(\text{hitter}(\zeta_n)) \neq \Sigma(P)\}.$$

Note that $\mathbf{BS}^\wedge(P)$ can be computed directly from the Process Hitting in the same way as $\mathbf{BS}(P)$, but even more efficiently since only minimal sets of hitters are kept after pruning redundant explorations.

The relations of abstractions and concretisations between scenarios and (abstracted) bounce sequences can be derived easily, so the details are not given here.

Example 4.15 (Process Hitting in Figure 1). $\zeta = a_1 \rightarrow b_1 \Rsh b_0 :: d_1 \rightarrow b_0 \Rsh b_2$ is the only bounce sequence resolving the objective $b_1 \Rsh^* b_2$:

$$\mathbf{BS}(b_1 \Rsh^* b_2) = \{\zeta\}$$

and

$$\mathbf{BS}^\wedge(b_1 \Rsh^* b_2) = \{\{a_1, d_1\}\}.$$

4.4. *Objective sequence refinements*

Before introducing the objective sequence refinement operators, we will define the relation $\preccurlyeq_{\mathbf{OS}}$ between two objective sequences (see Definition 4.16). Basically, if $\omega \preccurlyeq_{\mathbf{OS}} \omega'$, we say ω is a more precise abstraction than ω', hence $\gamma_\varsigma(\omega) \subseteq \gamma_\varsigma(\omega')$ (see Property 4.18). In such a setting, an objective sequence joined to an objective is a more precise abstraction than the objective alone (see Property 4.19).

Definition 4.16 ($\preccurlyeq_{\mathbf{OS}} \subset \mathbf{OS} \times \mathbf{OS}$). $\omega \preccurlyeq_{\mathbf{OS}} \omega'$ if and only if the following properties are satisfied:

— $|\omega| \geqslant |\omega'|$;
— there exists a mapping $\phi : \mathbb{I}^{\omega'} \mapsto \mathbb{I}^{\omega}$ such that

$$\forall n \in \mathbb{I}^{\omega'}, \text{bounce}(\omega'_n) = \text{bounce}(\omega_{\phi(n)})$$

and

$$\forall n, m \in \mathbb{I}^{\omega'}, n < m \Leftrightarrow \phi(n) < \phi(m).$$

Example 4.17. We have

$$b_0 \Rsh^* b_1 :: a_0 \Rsh^* a_1 :: b_1 \Rsh^* b_2 \preccurlyeq_{\mathbf{OS}} a_0 \Rsh^* a_1 :: b_0 \Rsh^* b_2 \preccurlyeq_{\mathbf{OS}} b_0 \Rsh^* b_2,$$

but

$$b_0 \Rsh^* b_1 \not\preccurlyeq_{\mathbf{OS}} b_0 \Rsh^* b_2.$$

Property 4.18. $\omega \preccurlyeq_{\mathbf{OS}} \omega' \implies \gamma_\varsigma(\omega) \subseteq \gamma_\varsigma(\omega')$.

Property 4.19. Given $\omega \in \mathbf{OS}$ and $P \in \mathbf{Obj}$, we have $\omega \oplus P \preccurlyeq_{\mathbf{OS}} P$.

Finally, given an objective P, $\mathbf{BS}_\varsigma(P)$ (see Definition 4.20) and $\mathbf{BS}_\varsigma^\wedge(P)$ (see Definition 4.21), we can generalise $\mathbf{BS}(P)$ and $\mathbf{BS}^\wedge(P)$ to the scope of the context ς, respectively.

Definition 4.20 ($\mathbf{BS}_\varsigma : \mathbf{Obj} \mapsto \wp(\mathbf{BS})$).

$$\mathbf{BS}_\varsigma(\star \Rsh^* a_j) \triangleq \bigcup_{a_i \in \varsigma[a]} \mathbf{BS}(a_i \Rsh^* a_j).$$

Definition 4.21 ($\mathbf{BS}_\varsigma^\wedge : \mathbf{Obj} \mapsto \wp(\wp(\mathbf{Proc}))$).

$$\mathbf{BS}_\varsigma^\wedge(\star \Rsh^* a_j) \triangleq \bigcup_{a_i \in \varsigma[a]} \mathbf{BS}^\wedge(a_i \Rsh^* a_j).$$

4.4.1. *Objective refinement by* **BS** *(ρ).* We will now build the function β such that, given an objective sequence P, a bounce sequence $\zeta \in \mathbf{BS}(P)$ is abstracted by $\beta(\zeta)$ to the objective sequence describing the successive reachability of its hitters (see Definition 4.22). From the definition of $\mathbf{BS}_\varsigma$, if a scenario concretises P in context ς, it necessarily concretises one bounce sequence $\zeta \in \mathbf{BS}_\varsigma(P)$, so $\beta(\zeta)$. The refinement operator $\rho(P, \mathbf{BS}_\varsigma(P))$ extends P to the set of objective sequences where P is prefixed by each $\beta(\zeta)$, $\zeta \in \mathbf{BS}_\varsigma(P)$ (see Definition 4.23). Finally, Lemma 4.25 states the correctness of this refinement, which ensures the preservation of the concretisation set.

Definition 4.22 ($\beta : \mathbf{BS} \mapsto \wp(\mathbf{OS})$).

$$\beta(\zeta) \triangleq \{\omega \in \mathbf{OS} \mid |\omega| = |\zeta| \wedge \forall n \in \mathbb{I}^\zeta, \mathrm{bounce}(\omega_n) = \mathrm{hitter}(\zeta_n)\}.$$

Definition 4.23 ($\rho : \mathbf{Obj} \times \wp(\mathbf{BS}) \mapsto \wp(\mathbf{OS})$).

$$\rho(P, zs) \triangleq \{\omega \oplus P \mid \omega \in \beta(\zeta), \zeta \in zs\}.$$

Example 4.24 (Process Hitting in Figure 1). Given the objective $d_0 \Rsh^* d_2$, from Definition 4.13, we know that

$$\mathbf{BS}(d_0 \Rsh^* d_2) = \{b_0 \to d_0 \Rsh d_1 :: b_1 \to d_1 \Rsh d_2, b_2 \to d_0 \Rsh d_2\}.$$

From Definition 4.22, we obtain

$$\beta(b_0 \to d_0 \Rsh d_1 :: b_1 \to d_1 \Rsh d_2) = \{\star \Rsh^* b_0 :: b_0 \Rsh^* b_1\}$$
$$\beta(b_2 \to d_0 \Rsh d_2) = \{\star \Rsh^* b_2\}.$$

Hence,

$$\rho(d_0 \Rsh^* d_2, \mathbf{BS}(d_0 \Rsh^* d_2)) = \{\star \Rsh^* b_0 :: b_0 \Rsh^* b_1 :: d_0 \Rsh^* d_2, \star \Rsh^* b_2 :: d_0 \Rsh^* d_2\}.$$

Lemma 4.25. $\gamma_\varsigma(P) = \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)))$.

Proof.

($\supseteq$) We have

$$\forall \omega \in \rho(P, \mathbf{BS}_\varsigma(P)), \omega \preccurlyeq_{\mathbf{OS}} P,$$

so

$$\gamma_\varsigma(P) \supseteq \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P))).$$

($\subseteq$) By the definition of $\mathbf{BS}_\varsigma(P)$, we have

$$\forall \delta \in \gamma_\varsigma(P), \exists \omega \in \rho(P, \mathbf{BS}_\varsigma(P)), \delta \in \gamma_\varsigma(\omega),$$

so

$$\gamma_\varsigma(P) \subseteq \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P))). \qquad \square$$

4.4.2. *Objective refinement by* $\mathbf{BS}^\wedge$ *(*$\rho^\wedge$*).* The refinement of an objective P by $\mathbf{BS}^\wedge$ is done in a similar way. A set of hitters $ps \in \mathbf{BS}^\wedge(P)$ is abstracted by $\beta^\wedge(ps)$ into the set of objective sequences describing any ordering of the reach of these hitters. The relation between objective sequences in $\beta(\zeta)$ and in $\beta^\wedge(ps)$ is emphasised in Property 4.27. The refinement $\rho^\wedge(P, \mathbf{BS}_\varsigma^\wedge(P))$ is presented in Definition 4.28, and the preservation of concretisations is stated in Lemma 4.30.

Definition 4.26 **(**$\beta^\wedge : \wp(\mathbf{Proc}) \mapsto \wp(\mathbf{OS})$**).**

$$\beta^\wedge(ps) \triangleq \{\omega \in \mathbf{OS} \mid |\omega| = |ps| \wedge \forall p \in ps, \exists n \in \mathbb{I}^\omega, \mathrm{bounce}(\omega_n) = p\}.$$

Property 4.27. $\forall \zeta \in \mathbf{BS}(P), \forall \omega \in \beta(\zeta), \exists \omega' \in \beta^\wedge(\zeta^\wedge), \omega \preccurlyeq_{\mathbf{OS}} \omega'$.

Definition 4.28 **(**$\rho^\wedge : \mathbf{Obj} \times \wp(\wp(\mathbf{Proc})) \mapsto \wp(\mathbf{OS})$**).**

$$\rho^\wedge(P, pss) \triangleq \{\omega \oplus P \mid \omega \in \beta^\wedge(ps), ps \in pss\}.$$

Example 4.29 (Process Hitting in Figure 1). Given the objective $d_0 \Rsh^* d_2$, from Definition 4.14, we know that $\mathbf{BS}^\wedge(d_0 \Rsh^* d_2) = \{\{b_0, b_1\}, \{b_2\}\}$. From Definition 4.26, we obtain

$$\beta^\wedge(\{b_0, b_1\}) = \{\star \Rsh^* b_0 :: b_0 \Rsh^* b_1, \star \Rsh^* b_1 :: b_1 \Rsh^* b_0,\}$$
$$\beta^\wedge(\{b_2\}) = \{\star \Rsh^* b_2\}.$$

Hence,

$$\begin{aligned}\rho^\wedge(d_0 \Rsh^* d_2, \mathbf{BS}^\wedge(d_0 \Rsh^* d_2)) = \{&\star \Rsh^* b_0 :: b_0 \Rsh^* b_1 :: d_0 \Rsh^* d_2,\\ &\star \Rsh^* b_1 :: b_1 \Rsh^* b_0 :: d_0 \Rsh^* d_2,\\ &\star \Rsh^* b_2 :: d_0 \Rsh^* d_2\}.\end{aligned}$$

Lemma 4.30. $\gamma_\varsigma(P) = \gamma_\varsigma(\rho^\wedge(P, \mathbf{BS}_\varsigma^\wedge(P)))$.

Proof.
($\supseteq$) We have

$$\forall \omega \in \rho^\wedge(P, \mathbf{BS}_\varsigma^\wedge(P)), \omega \preccurlyeq_{\mathbf{OS}} P.$$

($\subseteq$) By Property 4.27, we have

$$\forall \omega \in \rho(P, \mathbf{BS}_\varsigma(P)), \exists \omega' \in \rho^\wedge(P, \mathbf{BS}_\varsigma^\wedge(P)), \omega \preccurlyeq_{\mathbf{OS}} \omega',$$

so $\gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)) \subseteq \gamma_\varsigma(\rho^\wedge(P, \mathbf{BS}_\varsigma^\wedge(P))$. $\square$

4.4.3. *Objective sequence refinements (*$\widetilde{\rho}$*).* Finally, to generalise the refinements defined on an objective to objective sequence, we show an objective sequence refinement operator that uses any of the above refinements. We take the operator ρ as an example and define the refinement $\widetilde{\rho}(\omega, \mathbf{BS})$ (see Definition 4.32). Basically, this refinement chooses any objective ω_n of the objective sequence, refines it using ρ and returns all the interleaving of the resulting refined sequences with the objective sequence $\omega_{1..n-1}$ (see Definition 4.31). The concretisation set is then preserved (see Lemma 4.34).

Definition 4.31 (interleave : $\mathbf{OS} \times \mathbf{OS} \mapsto \wp(\mathbf{OS})$).

$$\begin{aligned}\text{interleave}(\omega^1, \omega^2) \triangleq \{\omega \in \mathbf{OS} \mid |\omega| = |\omega^1| + |\omega^2| \wedge \exists \phi^1 : \mathbb{I}^{\omega^1} \mapsto \mathbb{I}^{\omega}, \phi^2 : \mathbb{I}^{\omega^2} \mapsto \mathbb{I}^{\omega},\\ (\forall n, m \in \mathbb{I}^{\omega^1}, n < m \Leftrightarrow \phi^1(n) < \phi^1(m))\\ \wedge(\forall n, m \in \mathbb{I}^{\omega^2}, n < m \Leftrightarrow \phi^2(n) < \phi^2(m))\\ \wedge(\nexists n^1 \in \mathbb{I}^{\omega^1}, n^2 \in \mathbb{I}^{\omega^2}, \phi^1(n^1) = \phi^2(n^2))\}.\end{aligned}$$

Definition 4.32 ($\widetilde{\rho} : \mathbf{OS} \times \wp(\mathbf{BS}) \mapsto \wp(\mathbf{OS})$).

$$\begin{aligned}\widetilde{\rho}(\omega, \mathbf{BS}) \triangleq \{\varpi \oplus \omega_{n..|\omega|} \mid n \in \mathbb{I}^{\omega}, \omega' \oplus \omega_n \in \rho(\omega_n, \mathbf{BS}(\omega_n)),\\ \varpi \in \text{interleave}(\omega_{1..n-1}, \omega')\}.\end{aligned}$$

Example 4.33 (Process Hitting in Figure 1). Given the objective sequence $b_0 \Rsh^* b_2 :: d_0 \Rsh^* d_2$, from Definition 4.23, we obtain

$$\begin{aligned}\rho(b_0 \Rsh^* b_2, \mathbf{BS}(b_0 \Rsh^* b_2)) &= \{\star \Rsh^* d_1 :: b_0 \Rsh^* b_2\}\\ \rho(d_0 \Rsh^* d_2, \mathbf{BS}(d_0 \Rsh^* d_2)) &= \{\star \Rsh^* b_0 :: b_0 \Rsh^* b_1 :: d_0 \Rsh^* d_2, \star \Rsh^* b_2 :: d_0 \Rsh^* d_2\}.\end{aligned}$$

Hence,

$$\begin{aligned}\widetilde{\rho}(b_0 \Rsh^* b_2 :: d_0 \Rsh^* d_2, \mathbf{BS}) = \{&b_0 \Rsh^* b_2 :: b_2 \Rsh^* b_0 :: b_0 \Rsh^* b_1 :: d_0 \Rsh^* d_2,\\ &\star \Rsh^* b_0 :: b_0 \Rsh^* b_2 :: b_2 \Rsh^* b_1 :: d_0 \Rsh^* d_2,\\ &\star \Rsh^* b_0 :: b_0 \Rsh^* b_1 :: b_1 \Rsh^* b_2 :: d_0 \Rsh^* d_2,\\ &b_0 \Rsh^* b_2 :: b_2 \Rsh^* b_2 :: d_0 \Rsh^* d_2\}.\end{aligned}$$

Lemma 4.34. $\gamma_\varsigma(\omega) = \gamma_\varsigma(\widetilde{\rho}(\omega, \mathbf{BS}))$.

Proof.

($\supseteq$) We have

$$\forall \omega' \in \widetilde{\rho}(\omega, \mathbf{BS}), \omega' \preccurlyeq_{\mathbf{OS}} \omega.$$

($\subseteq$) By Lemma 4.25 and Definition 4.31,

$$\delta \in \gamma_\varsigma(\omega) \Rightarrow \exists \omega' \in \widetilde{\rho}(\omega, \mathbf{BS}), \delta \in \gamma_\varsigma(\omega'). \qquad \square$$

5. Over- and under-approximations of process reachability

We define the *Process Reachability* problem as deciding if a given objective sequence $\omega \in \mathbf{OS}$ is concretisable for a given Process Hitting in a context ς: that is, if the set $\gamma_\varsigma(\omega)$ is non-empty.

Using the refinement operators defined in the previous section, we establish several necessary or sufficient conditions for the concretisability of an objective sequence in a given context ς. These objective sequences can be given either by a user (to check some temporal properties) or extracted from **BS** (with β – see Definition 4.22) to refine this set of bounce sequences. Indeed, if a bounce sequence is not concretisable in ς, it can be ignored in all analyses in the scope of this context.

The aim of these approximations is that they can be computed very rapidly, enabling us to overcome the state-space explosion problem inherent in the analysis of dynamics like these. While inconclusive in some cases, the application section (see Section 6) shows that our analyses are well-suited to biological regulatory networks dynamics and have a very promising scalability.

The rest of this section is structured as follows. Section 5.1 presents a first over-approximation based on an unordered analysis of objectives required to concretise the given objective sequence. Section 5.2 then refines this approximation by exploiting the sequentiality of objectives to concretise. Section 5.3 uses order constraints between process occurrences to complete these over-approximations. Finally, Section 5.4 sets up an under-approximation of the process reachability decision.

5.1. *Unordered over-approximation*

Given a context ς and an objective sequence ω, we find that ω is concretisable only if each objective $\omega_n, n \in \mathbb{I}^\omega$, is independently concretisable in the same context (see Proposition 5.1). In this way, we can approximate the concretisability of an objective sequence by recursively applying Proposition 5.1 to extract objectives from the given objective sequence, and use the refinement operator $\rho^\wedge$ to extend the objective into several objective sequences.

Proposition 5.1. $\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^\omega, \gamma_\varsigma(\omega_n) \neq \emptyset$.

Proof. Definition 4.16 and Property 4.18 give $\omega \preccurlyeq_{\mathbf{OS}} \omega_i$. □

Given an objective P, $\mathrm{minCont}_\varsigma(P)$ (see Definition 5.2) is the set of re-targeted objectives Q, with $\mathrm{target}(P) \neq \mathrm{target}(Q)$ and $\mathrm{bounce}(P) = \mathrm{bounce}(Q)$, which are always derived from a recursive application of $\rho^\wedge(P, \mathbf{BS}^\wedge(P))$ using Proposition 5.1 (see Lemma 5.3).

Definition 5.2 ($\mathrm{minCont}_\varsigma : \mathbf{Obj} \mapsto \wp(\mathbf{Obj})$).

$$\mathrm{minCont}_\varsigma(\star \Rsh^* a_j) \triangleq \{a_k \Rsh^* a_j \mid a_k \neq a_j \wedge \forall a_i \in \varsigma[a], a_k \in \mathrm{minCont}_\varsigma^{Obj}(a, a_i \Rsh^* a_j)\}$$

$$\mathrm{minCont}_\varsigma^{Obj} : \Sigma \times \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$$

$$\mathrm{minCont}_\varsigma^{Obj}(a, P) \triangleq \begin{cases} \emptyset & \text{if } \mathbf{BS}^\wedge(P) = \emptyset \\ \{p \in \mathbf{Proc} \mid \forall ps \in \mathbf{BS}^\wedge(P), & \\ \qquad \exists q \in ps, p \in \mathrm{minCont}_\varsigma^{Proc}(a, q)\} & \text{otherwise.} \end{cases}$$

$$\mathrm{minCont}_\varsigma^{Proc} : \Sigma \times \mathbf{Proc} \mapsto \wp(\mathbf{Proc})$$

$$\mathrm{minCont}_\varsigma^{Proc}(a, b_i) \triangleq \begin{cases} \{b_i\} & \text{if } a = b \\ \{p \in \mathbf{Proc} \mid \forall b_j \in \varsigma[b], & \\ \qquad p \in \mathrm{minCont}_\varsigma^{Obj}(a, b_j \Rsh^* b_i)\} & \text{otherwise.} \end{cases}$$

Lemma 5.3. $a_k \Rsh^* a_j \in \mathrm{minCont}_\varsigma(\star \Rsh^* a_j) \Longrightarrow \gamma_\varsigma(\star \Rsh^* a_j) = \gamma_\varsigma(\star \Rsh^* a_k :: a_k \Rsh^* a_j)$.

Proof. We use induction on $\mathrm{minCont}_\varsigma$, a_k occurs in all recursive refinements of $\star \Rsh^* a_j$ by $\rho^\wedge$. □

Example 5.4 (Process Hitting in Figure 1). Given $\varsigma = \langle a_1, b_1, b_2, d_0 \rangle$, we have $\mathrm{minCont}_\varsigma (b_1 \Rsh^* b_2) = \{b_0 \Rsh^* b_2\}$:

$$\begin{aligned}
&\mathbf{BS}^\wedge(b_1 \Rsh^* b_2) = \{\{a_1, d_1\}\}\\
&\mathrm{minCont}_\varsigma^{Obj}(b, b_1 \Rsh^* b_2) = \mathrm{minCont}_\varsigma^{Proc}(b, a_1) \cup \mathrm{minCont}_\varsigma^{Proc}(b, d_1)\\
&\mathrm{minCont}_\varsigma^{Proc}(b, a_1) = \mathrm{minCont}_\varsigma^{Obj}(b, a_1 \Rsh^* a_1) = \emptyset\\
&\mathrm{minCont}_\varsigma^{Proc}(b, d_1) = \mathrm{minCont}_\varsigma^{Obj}(b, d_0 \Rsh^* d_1)\\
&\mathbf{BS}^\wedge(d_0 \Rsh^* d_1) = \{\{b_0\}\}\\
&\mathrm{minCont}_\varsigma^{Obj}(b, d_0 \Rsh^* d_1) = \mathrm{minCont}_\varsigma^{Proc}(b, b_0) = \{b_0\}.
\end{aligned}$$

Thus,

$$\mathrm{minCont}_\varsigma^{Obj}(b, b_1 \Rsh^* b_2) = \{b_0\}.$$

Proposition 5.5 summarises the necessary conditions for the concretisability of an objective P in context ς: for at least one set of processes $ps \in \mathbf{BS}_\varsigma^\wedge(P)$, $\forall p \in ps$, the objective $\star \Rsh^* p$ is concretisable in ς; and $\forall Q \in \mathrm{minCont}_\varsigma(P)$, Q is concretisable in ς.

Proposition 5.5. $\gamma_\varsigma(P) \neq \emptyset \Longrightarrow \exists ps \in \mathbf{BS}_\varsigma^\wedge(P), \forall p \in ps, \gamma_\varsigma(\star \Rsh^* p) \neq \emptyset$ and $\forall Q \in \mathrm{minCont}_\varsigma(P), \gamma_\varsigma(Q) \neq \emptyset$.

Proof. The result follows from Lemma 4.30, Lemma 5.3 and Proposition 5.1. □

This leads us to the definition of the monotonic function $\mathrm{badObjs}_\varsigma$ (see Definition 5.6), which removes from a set of objectives Ps those not satisfying the necessary condition given by Proposition 5.5 in the scope of Ps. Hence, the greatest fixpoint of $\mathrm{badObjs}_\varsigma$ contains all the objectives not satisfying the necessary condition for their concretisability. Theorem 5.7 follows from this fact.

Definition 5.6 ($\mathrm{badObjs}_\varsigma : \wp(\mathbf{Obj}) \mapsto \wp(\mathbf{Obj})$).

$$\begin{aligned}
\mathrm{badObjs}_\varsigma(Ps) = \{P \in Ps \mid \mathbf{BS}_\varsigma^\wedge(P) \neq \emptyset \wedge &\\
&((\forall ps \in \mathbf{BS}_\varsigma^\wedge(P), \exists p \in ps, \star \Rsh^* p \in Ps)\\
&\vee (\exists Q \in \mathrm{minCont}_\varsigma(P), Q \in Ps))\}.
\end{aligned}$$

Theorem 5.7 (unordered over-approximation).

$$\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^\omega, \omega_n \notin \mathrm{gfp}\ \mathrm{badObjs}_\varsigma.$$

Proof. Proposition 5.5 and Definition 5.6 give $\gamma_\varsigma(\omega_n) = \emptyset$ if $\omega_n \in \mathrm{gfp}\ \mathrm{badObjs}_\varsigma$. □

5.1.1. *Implementation.* Given a context ς and an objective sequence ω, we define an abstract structure $\mathscr{A}_\varsigma^\omega$ (see Definition 5.8) that mimics the relations between objectives during the execution of Proposition 5.5 (see Lemma 5.9). $\mathscr{A}_\varsigma^\omega$ gathers together three relations $(\mathrm{Req}_\varsigma^\omega, \mathrm{Sol}_\varsigma^\omega, \mathrm{Cont}_\varsigma^\omega)$, which are the requirements, solutions and minimal continuity (or re-targeting), respectively.

Definition 5.8 ($\mathscr{A}_\varsigma^\omega$). Given a context ς and an objective sequence ω, we define the abstract structure

$$\mathscr{A}_\varsigma^\omega = (\mathrm{Req}_\varsigma^\omega, \mathrm{Sol}_\varsigma^\omega, \mathrm{Cont}_\varsigma^\omega),$$

where $\mathrm{Req}_\varsigma^\omega$, $\mathrm{Sol}_\varsigma^\omega$ and $\mathrm{Cont}_\varsigma^\omega$ are defined as follows:

$$\begin{aligned}
\mathrm{Req}_\varsigma^\omega &\triangleq \{(a_i, a_j \Rsh^* a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in \mathrm{Sol}_\varsigma^\omega, a_i \in ps \\
&\qquad\qquad \vee \exists n \in \mathbb{I}^\omega, \mathrm{bounce}(\omega) = a_i)\} \\
\mathrm{Sol}_\varsigma^\omega &\triangleq \{(P, ps) \in \mathbf{Obj} \times \wp(\mathbf{Proc}) \mid \exists (a_i, P) \in \mathrm{Req}_\varsigma^\omega \wedge ps \in \mathbf{BS}^\wedge(P)\} \\
\mathrm{Cont}_\varsigma^\omega &\triangleq \{(P, Q) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in \mathrm{Sol}_\varsigma^\omega \wedge Q \in \mathrm{minCont}_\varsigma(p)\}.
\end{aligned}$$

Lemma 5.9. Given an objective P referenced in $\mathscr{A}_\varsigma^\omega$, we have

$$\begin{aligned}
\omega' \oplus P \in \rho^\wedge(P, \mathbf{BS}^\wedge(P) \iff & (P, \{\mathrm{bounce}(\omega'_n) \mid n \in \mathbb{I}^{\omega'}\}) \in \mathrm{Sol}_\varsigma^\omega \\
& \wedge \forall n \in \mathbb{I}^{\omega'}, a_j = \mathrm{bounce}(\omega'_n) \\
& \wedge \forall a_i \in \varsigma[a], (a_j, a_i \Rsh^* a_j) \in \mathrm{Req}_\varsigma^\omega
\end{aligned}$$

and

$$Q \in \mathrm{minCont}_\varsigma(P) \iff (P, Q) \in \mathrm{Cont}_\varsigma^\omega.$$

Proof. The result follows from the construction of $\mathscr{A}_\varsigma^\omega$. □

$\mathscr{A}_\varsigma^\omega$ has a graph structure, with cycles, potentially. Note that, as $|\mathbf{Obj}| = \sum_{a \in \Sigma} |L_a|^2$, the size of $\mathrm{Req}_\varsigma^\omega$ and $\mathrm{Cont}_\varsigma^\omega$ sets are polynomial in the number of processes in the Process Hitting. The size of $\mathrm{Sol}_\varsigma^\omega$ also depends on the cardinality of $\mathbf{BS}^\wedge$, which follows the maximum number of combinations of $|L_a|$ different processes, which is much less than exponential growth.

Finally, Algorithm 5.10 describes the computation of the Theorem 5.7 decision.

Algorithm 5.10 (unordered over-approximation). Given a context ς, an objective sequence $\omega \in \mathbf{OS}$ and the abstract structure $\mathscr{A}_\varsigma^\omega$:

(1) Initialise $\Theta = \{P \in \mathbf{Obj} \mid (P, \emptyset) \in \mathrm{Sol}_\varsigma^\omega\}$.
(2) Repeat the following until Θ becomes a fixpoint:

(a) $\Upsilon = \{p \in \mathbf{Proc} \mid \exists P \in \Theta, (p, P) \in \mathrm{Req}_\varsigma^\omega\}$.

(b) $\Theta = \{P \in \mathbf{Obj} \mid \exists (P, ps) \in \mathrm{Sol}_\varsigma^\omega, ps \subseteq \Upsilon \wedge \forall (P, Q) \in \mathrm{Cont}_\varsigma^\omega, Q \in \Theta\}$.

(3) $\gamma_\varsigma(\omega) \neq \emptyset \implies \forall n \in \mathbb{I}^\omega, \exists P \in \Theta, \mathrm{target}(P) \in \varsigma \wedge \mathrm{bounce}(P) = \mathrm{bounce}(\omega_n)$.

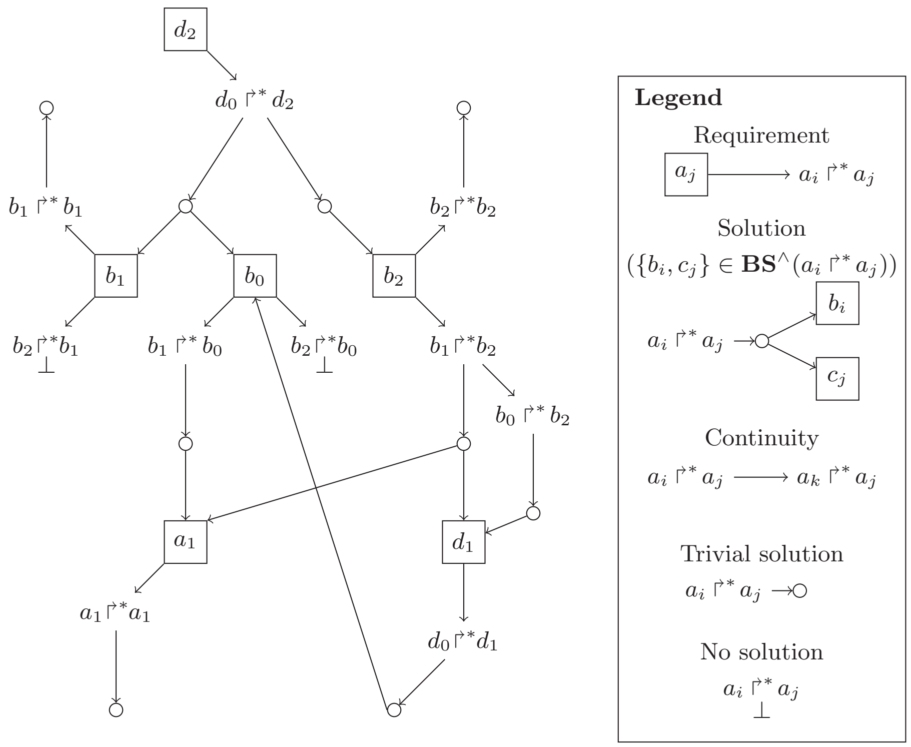

Figure 4. Graphical representation of the abstract structure $\mathscr{A}^\omega_\varsigma$ extracted from the Process Hitting in Figure 1, with $\omega = d_0 \Rsh^* d_2$ and $\varsigma = \langle a_1, b_1, b_2, d_0 \rangle$.

5.1.2. *Complexity.* The computation of $\mathscr{A}^\omega_\varsigma$ is done by iteratively adding the required processes and objectives. The steps of Algorithm 5.10 are carried out polynomially in the size of $\mathscr{A}^\omega_\varsigma$. Putting aside the $\mathbf{BS}^\wedge$ computation complexity, the proposed over-approximation can then be achieved by a number of operations polynomial in the size of the abstract structure.

5.1.3. *Examples.* Figures 4 and 5 graphically represent the abstract structures extracted from the Process Hitting example in Figure 1 for different objective sequences and contexts. Such structures are built recursively: starting from a process node (squares), it is related to each group of objectives having their target in the given context and their bounce equal to this process; then each objective node is related to one solution node (circles) per abstract bounce sequences; finally, solutions nodes are related to the process nodes composing them. For instance, in Figure 4 with the objective sequence $\omega = d_0 \Rsh^* d_2$ and the context $\varsigma = \langle a_1, b_1, b_2, d_0 \rangle$, the structure is built from the process node d_2 (the bounce of the objective in ω), which is related to the objective node $d_0 \Rsh^* d_2$, which is related to two solutions ($|\mathbf{BS}^\wedge(d_0 \Rsh^* d_2)| = 2$): one of which is related to the process nodes b_0 and b_1 ($\{b_0, b_1\} \in \mathbf{BS}^\wedge(d_0 \Rsh^* d_2)$) and the other to the process node b_2 ($\{b_2\} \in \mathbf{BS}^\wedge(d_0 \Rsh^* d_2)$).

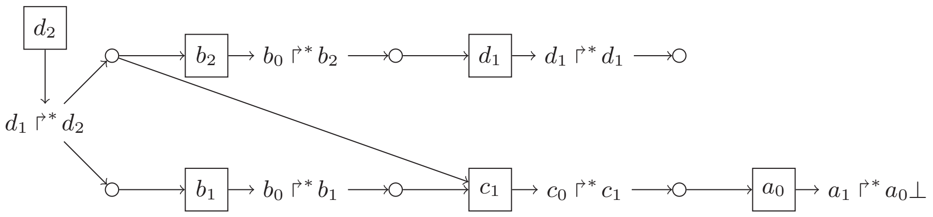

Figure 5. Abstract structure $\mathscr{A}^{\omega}_{\varsigma}$ for the Process Hitting example in Figure 1 with $\omega = d_1 \Rsh^* d_2$ and $\varsigma = \langle a_1, b_0, c_0, d_1 \rangle$. By Theorem 5.7, the objective $d_1 \Rsh^* d_2$ is *not concretisable.*

In the case addressed in Figure 4, Theorem 5.7 is satisfied. Figure 5 applies Theorem 5.7 to the Process Hitting example in Figure 1 for the decision of the objective $d_1 \Rsh^* d_2$ concretisability. In this case, the necessary condition is not satisfied.

5.2. *Ordered over-approximation*

This section exploits the ordering of objectives in an objective sequence to increase the conclusiveness of the over-approximation for its concretisability.

Given a non-trivial objective P in the given context ς, we use ends(P) to denote the set of processes a scenario concretising P may lead to (see Definition 5.11 and Proposition 5.12). This set is computed from $\mathbf{BS}(P)$ by taking the hitter and bounce of the last action in each bounce sequence.

Definition 5.11 (ends$_\varsigma$: **Obj** $\mapsto \wp(\wp(\textbf{Proc}))$).

$$\text{ends}_\varsigma(\star \Rsh^* a_i) \triangleq \{\text{end}(h) \mid \exists a_j \in \varsigma[a], \exists \zeta \in \mathbf{BS}(a_j \Rsh^* a_i), \zeta \neq \varepsilon \wedge h = \zeta_{|\zeta|}\}.$$

Proposition 5.12. $\gamma_\varsigma(\star \Rsh^* a_i) \neq \emptyset \wedge a_i \notin \varsigma[a] \implies \exists \delta \in \gamma_\varsigma(\star \Rsh^* a_i), \exists eps \in \text{ends}_\varsigma(\star \Rsh^* a_i)$ such that $eps \subseteq \text{end}(\delta)$.

Proof. Since $a_i \notin \varsigma[a]$, there exists $n \in \mathbb{I}^\delta$ such that bounce$(\delta_n) = a_i$. Hence, we have $\delta_{1..n} \in \gamma_\varsigma(\star \Rsh^* a_i)$ and end$(\delta_n) \subseteq$ end$(\delta_{1..n})$. So by Definition 5.11, we then get end$(\delta_n) \in$ ends$_\varsigma(\star \Rsh^* a_i)$. □

Given an abstract structure $\mathscr{A}^{\omega}_{\varsigma} \in \mathbb{A}$, we define procs$(\mathscr{A}^{\omega}_{\varsigma})$ as the set of processes referenced in $\mathscr{A}^{\omega}_{\varsigma}$ (see Definition 5.13) and maxprocs$_\varsigma(\omega)$ (see Definition 5.14) as the set of processes present in the abstract structure having its context saturated (that is, such that $\varsigma \Cap \text{procs}(\mathscr{A}^{\omega}_{\varsigma}) = \varsigma$). Note that there is a particular optimisation when $\omega = P$, where the process bounce(P) can be ignored by the context.

Definition 5.13 (procs : $\mathbb{A} \mapsto \wp(\textbf{Proc})$).

$$\begin{aligned}\text{procs}((\text{Sol}^{\omega}_{\varsigma}, \text{Req}^{\omega}_{\varsigma}, \text{Cont}^{\omega}_{\varsigma})) \triangleq \{p \in \textbf{Proc} \mid \exists (P, ps) \in \text{Sol}^{\omega}_{\varsigma}, & \; p \in ps \\ & \vee p = \text{target}(P) \\ & \vee (P \neq \omega \Rightarrow p = \text{bounce}(P))\}.\end{aligned}$$

Definition 5.14 (maxprocs$_\varsigma$: **OS** $\mapsto \wp$(**Proc**)).

$$\text{maxprocs}_\varsigma(\omega) \triangleq \text{procs}(\lceil \mathscr{A}_\varsigma^\omega \rceil) \qquad \text{where } \lceil \mathscr{A}_\varsigma^\omega \rceil \triangleq \text{lfp}\{\mathscr{A}_\varsigma^\omega\} \left(\mathscr{A}_\varsigma^\omega \mapsto \mathscr{A}^\omega_{\varsigma \Cap \text{procs}(\mathscr{A}_\varsigma^\omega)} \right).$$

By intersecting maxprocs$_\varsigma(\omega)$ and ends$_\varsigma(\omega_1)$, we obtain a context in which $\omega_{2..|\omega|}$ concretisability should satisfy Theorem 5.7 if ω is concretisable in ς. This is stated by Theorem 5.15, which gives a straightforward refinement of Theorem 5.7 by taking the sequentiality of objectives in ω into account.

Theorem 5.15 (ordered over-approximation). Given a context ς and $\omega = P :: \omega' \in$ **OS** such that bounce$(P) \notin \varsigma$, we have

$$\gamma_\varsigma(P :: \omega') \neq \varnothing \Longrightarrow \exists eps \in \text{ends}_\varsigma(P), \forall n \in \mathbb{I}^{\omega'}, \omega'_n \notin \text{gfp badObjs}_{max\varsigma},$$

with $max\varsigma = \varsigma \Cap \text{maxprocs}_\varsigma(\omega) \Cap eps$.

Proof. We have $\delta \in \gamma_\varsigma(P :: \omega') \Rightarrow \exists n \in \mathbb{I}^\delta$ such that

$$\begin{aligned} \text{bounce}(\delta_n) &= \text{bounce}(P) \\ eps &\subseteq \text{end}(\delta_{1..n}) \\ \delta_{n+1..|\delta|} &\in \gamma_{\varsigma'}(\omega') \end{aligned}$$

with $\varsigma' = \varsigma \Cap \text{support}(\delta)$.

Also $\forall n \in \mathbb{I}^{\omega'}$, if $\Sigma(\omega'_n) \in \Sigma(eps)$, since $\gamma_{\varsigma'}(\omega'_n) \neq \varnothing$ and by induction, we have

$$\omega'_n \notin \text{gfp badObjs}_{max\varsigma}.$$

If $\Sigma(\omega'_n) \notin \Sigma(eps)$, we have bounce$(\omega'_n) \in max\varsigma$, hence

$$\omega'_n \notin \text{gfp badObjs}_{max\varsigma}$$

(trivial objective). □

By defining maxCtx(ς, ω, n) as the maximum context after the resolution of $\omega_{1..n}$ (see Definition 5.16), the above theorem can be extended straightforwardly to Corollary 5.17.

Definition 5.16 (maxCtx : **Ctx** $\times$ **OS** $\times \mathbb{N} \mapsto$ **Ctx**). (we assume $n \in \{0\} \cup \mathbb{I}^n$):

$$\text{maxCtx}(\varsigma, \omega, n) \triangleq \begin{cases} \varsigma & \text{if } n = 0 \\ \varsigma \Cap \text{maxprocs}_\varsigma(\omega) & \text{if bounce}(\omega_n) \in \text{maxCtx}(\varsigma, \omega, n-1) \\ \varsigma \Cap \text{maxprocs}_\varsigma(\omega) \Cap eps & \text{otherwise, where } eps \in \text{ends}_\varsigma(\omega_n). \end{cases}$$

Corollary 5.17. We have

$$\gamma_\varsigma(\omega) \neq \varnothing \Longrightarrow \forall n \in \mathbb{I}^\omega, \gamma_{max\varsigma}(\omega_n) \neq \varnothing,$$

with

$$max\varsigma = \text{maxCtx}(\varsigma, \omega, n-1).$$

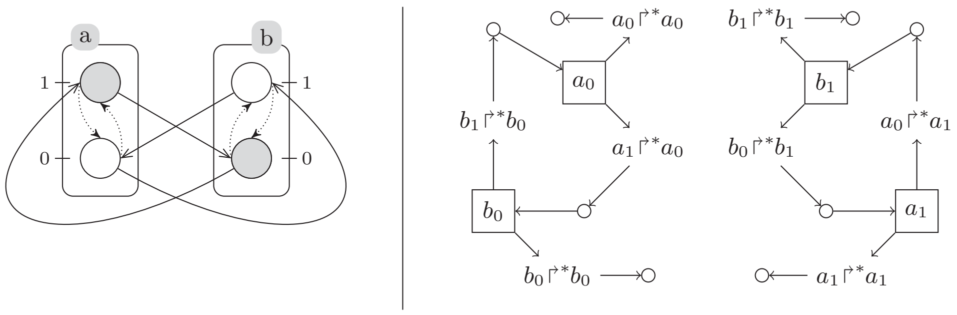

Figure 6. The diagram on the right-hand side is the saturated abstract structure $\lceil \mathscr{A}_\varsigma^\omega \rceil$ of the Process Hitting on the left-hand side with $\varsigma = \langle a_1, b_0 \rangle$ and $\omega = a_1 \Rsh^* a_0 :: b_0 \Rsh^* b_1$. Theorem 5.15 concludes that this objective sequence is not concretisable.

5.2.1. *Implementation.* The computation of $\text{maxprocs}_\varsigma(\omega)$ requires at most $|\mathbf{Proc}|$ iterations, giving a number of steps polynomial in the number of processes. The computation of $\text{ends}_\varsigma(\star \Rsh^* a_i)$ can either be derived from prior $\mathbf{BS}(\star \Rsh^* a_i)$ computations or, if the **BS** are too costly to compute, it can be approximated by either $\{\{\text{bounce}(P)\}\}$, or by

$$\{\text{end}(h) \mid h \in \mathscr{H} \wedge \text{bounce}(h) = a_i \wedge \exists a_j \in \varsigma[a], \exists ps \in \mathbf{BS}^\wedge(a_j \Rsh^* a_i), \text{target}(h) \in ps\}.$$

5.2.2. *Example.* Given the Process Hitting defined in Figure 6, with $\varsigma = \langle a_1, b_0 \rangle$, Theorem 5.7 is inconclusive on the concretisability of $\omega = a_1 \Rsh^* a_0 :: b_0 \Rsh^* b_1$. By applying Theorem 5.15, it appears that $b_0 \Rsh^* b_1$ is not concretisable in $max\varsigma = \text{maxCtx}(\varsigma, \omega, 1) = \langle a_0, b_0 \rangle$ (in such a case, the $\mathscr{A}_{max\varsigma}^{b_0 \Rsh^* b_1}$ forms a unique cycle between b_1 and a_1).

5.3. *Over-approximation using process occurrence order constraints*

The fact that an objective $a_i \Rsh^* a_j$ has no solution ($\mathbf{BS}^\wedge(a_i \Rsh^* a_j) = \varnothing$) tells us that the process a_j never occurs after a_i. This order constraint between process occurrences is denoted by $a_j \lhd a_i$ (see Definition 5.18 and Property 5.19).

Definition 5.18 ($\lhd$). The binary relation $\lhd \subset \mathbf{Proc} \times \mathbf{Proc}$ is a partial pre-order such that $a_j \lhd a_i$ (that is, $(a_j, a_i) \in \lhd$) if and only if there exists no scenario where a_j occurs after a_i:

$$a_j \lhd a_i \stackrel{\Delta}{=} \nexists \delta \in \mathbf{Sce} \text{ such that } \exists n, m \in \mathbb{I}^\delta, n \leqslant m, \text{target}(\delta_n) = a_i \wedge \text{bounce}(\delta_m) = a_j.$$

Property 5.19 (order constraint uncovering). $\mathbf{BS}(a_i \Rsh^* a_j) = \varnothing \Longrightarrow a_j \lhd a_i$.

Now that we know some order constraints on process occurrences, we want to check if some sequence of objectives does not contradict such constraints. This can be achieved by computing the processes that always occur when resolving an objective: given an objective sequence ω, if a process a_i is required by ω_n and a process a_j by ω_m, $n < m$, then the constraint $a_j \lhd a_i$ should not exist. This is illustrated by Figure 7.

In a similar way to minCont_ς (see Definition 5.2), $\text{minProc}_\varsigma(P)$ refers to the set of processes of *any* sort that occur in all refinements of P in the context ς (see Definition 5.20

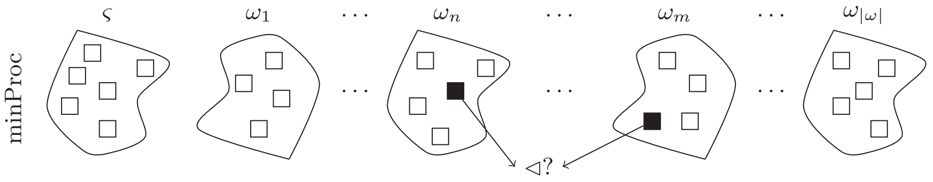

Figure 7. Illustration of the method developed in Section 5.3 to over-approximate the concretisability of an objective sequence ω in the context ς: for each objective, the minimal set of processes occurring (represented by squares) are computed using minProc (see Definition 5.22); the sequence is not concretisable when two processes (in black) occurring in distinct objectives resolution contradict the process occurrences order $\lhd$.

and Lemma 5.21). Using the previously defined maxCtx (see Definition 5.16), we define minProc(ς, ω, n) (see Definition 5.22), which is the set of processes occurring in the resolution of ω_n after having resolved $\omega_{1..n-1}$. From this definition, Theorem 5.23 states the over-approximation illustrated in Figure 7.

Definition 5.20 (minProc$_\varsigma$: **Obj** $\mapsto \wp$(**Proc**)). Given a context ς,

$$\begin{aligned}
\text{minProc}_\varsigma(\star \Rsh^* a_i) &\stackrel{\Delta}{=} \{p \in \textbf{Proc} \mid \forall a_j \in \varsigma[a],\\
&\qquad \textbf{BS}(a_j \Rsh^* a_i) \neq \emptyset \Rightarrow p \in \text{minProc}^{Obj}_\varsigma(a_j \Rsh^* a_i)\}\\
\text{minProc}^{Obj}_\varsigma &: \textbf{Obj} \mapsto \wp(\textbf{Proc})\\
\text{minProc}^{Obj}_\varsigma(a_j \Rsh^* a_i) &\stackrel{\Delta}{=} \{a_i\} \cup \{p \in \textbf{Proc}\\
&\quad \mid \forall ps \in \textbf{BS}^\wedge(a_j \Rsh^* a_i), \exists q \in ps, p \in \text{minProc}_\varsigma(\star \Rsh^* q)\\
&\quad \vee \exists a_k \Rsh^* a_i \in \text{minCont}^{Obj}_\varsigma(a, a_j \Rsh^* a_i),\\
&\qquad p \in \text{minProc}^{Obj}_\varsigma(a_j \Rsh^* a_k) \cup \text{minProc}^{Obj}_\varsigma(a_k \Rsh^* a_i)))\}.
\end{aligned}$$

Lemma 5.21. $\forall \delta \in \gamma_\varsigma(P), \forall p \in \text{minProc}_\varsigma(P), p \in \delta$.

Proof. We show by induction on minProc$_\varsigma$, that p occurs in all recursive refinements of P by $\rho^\wedge$. □

Definition 5.22 (minProc : **Ctx** $\times$ **OS** $\times \mathbb{N} \mapsto \wp$(**Proc**)). Assuming $n \in \{0\} \cup \mathbb{I}^\omega$, we define

$$\text{minProc}(\varsigma, \omega, n) \stackrel{\Delta}{=} \begin{cases} \{a_i \in \varsigma\} & \text{if } n = 0\\ \text{minProc}_{max\varsigma}(\omega_n) & \text{otherwise}\\ & \text{with } max\varsigma = \text{maxCtx}(\varsigma, w, n-1). \end{cases}$$

Theorem 5.23 (ordered over-approximation refined by $\lhd$). We have

$$\gamma_\varsigma(\omega) \neq \emptyset \implies$$
$$\nexists n, m \in \{0\} \cup \mathbb{I}^\omega, n < m, \exists p \in \text{minProc}(\varsigma, \omega, n), \exists q \in \text{minProc}(\varsigma, \omega, m), q \lhd p.$$

Proof. The result follows from Lemma 5.21, Corollary 5.17 and Definition 5.22. □

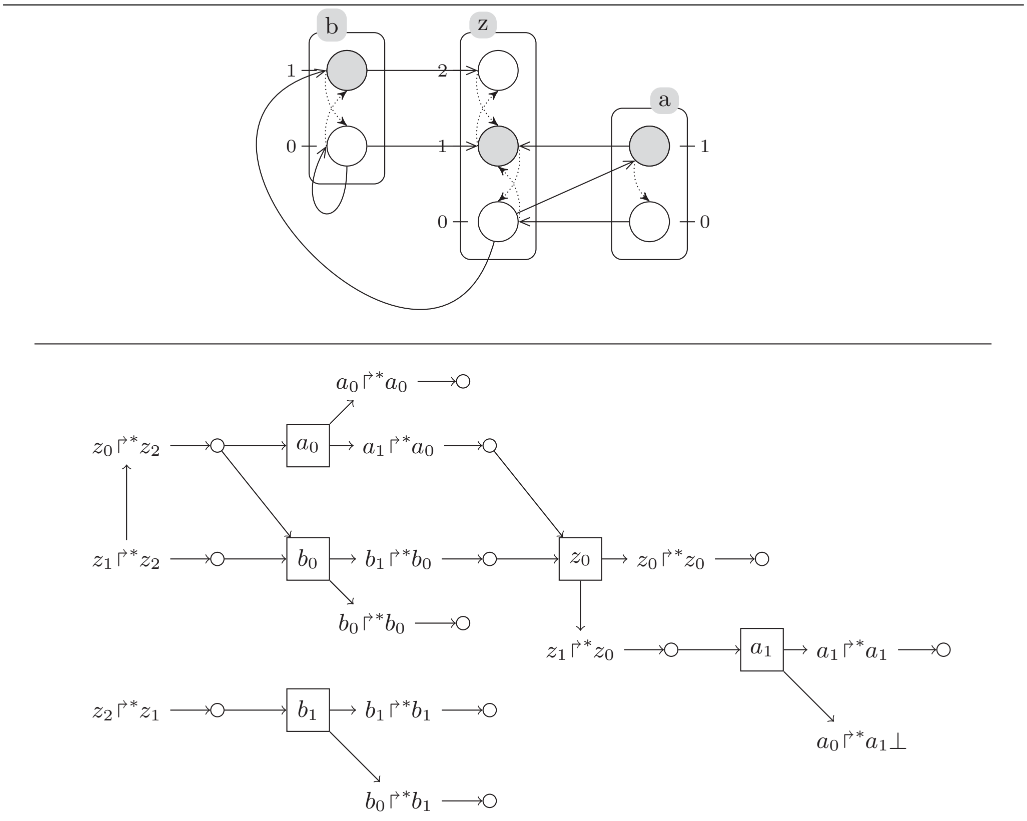

Figure 8. The bottom diagram shows the saturated abstract structure $\lceil \mathscr{A}_\varsigma^\omega \rceil$ of the Process Hitting in the top diagram with $\varsigma = \langle a_1, b_1, z_1 \rangle$ and $\omega = z_1 \Rsh^* z_2 :: z_2 \Rsh^* z_1 :: z_1 \Rsh^* z_2$. Theorem 5.23 tells us that this objective sequence is not concretisable.

5.3.1. *Implementation.* The implementation is very similar to that presented in Section 5.2. The uncovering of $\lhd$ is done linearly in the size of the saturated abstract structure $\lceil \mathscr{A}_\varsigma^\omega \rceil$.

5.3.2. *Example.* Let us define the Process Hitting as in Figure 8, and its saturated abstract structure $\lceil \mathscr{A}_\varsigma^\omega \rceil$, with $\varsigma = \langle a_1, b_1, z_1 \rangle$ and $\omega = z_1 \Rsh^* z_2 :: z_2 \Rsh^* z_1 :: z_1 \Rsh^* z_2$, for which the concretisability has to be decided. The evaluation of minCont(ς, ω, n) and maxCtx(ς, ω, n) give the following:

	$n = 0 - \varsigma$	$n = 1 - z_1 \Rsh^* z_2$	$n = 2 - z_2 \Rsh^* z_1$	$n = 3 - z_1 \Rsh^* z_2$
minProc(ς, ω, n)	a_1, b_1, z_1	$\mathbf{a_0}, a_1, b_0, z_0, z_2$	b_1, z_1	$a_0, \mathbf{a_1}, b_0, z_0, z_2$
maxCtx(ς, ω, n)	a_1, b_1, z_1	a_0, a_1, b_0, z_2	a_0, a_1, b_1, z_1	-

As $a_1 \lhd a_0$, we have ω is not concretisable in ς.

5.4. *Under-approximation*

The under-approximation procedure presented in this section takes advantage of a variant of the abstract structure used in the above over-approximations. If certain conditions on this abstract structure hold, we can show that a scenario concretising the given objective sequence exists. The proposed construction of the scenario is made by a so-called *top-down* resolution: given an objective sequence ω, we first build the scenario concretising ω_1 by preempting the resolution of the objective sequence $\omega_{2..|\omega|}$ (and hence, ignoring any objective interleaving that may be required to concretise ω). As stated by the refinement operators in Section 4.4, the concretisation of ω_1 involves the concretisation of a refinement of ω_1, resulting in a recursive procedure of scenario construction.

We first give an alternative definition of the set of scenarios concretising an objective sequence ω in a context ς by saying $\ell_\varsigma(\omega)$ is empty unless, for each state $s \in L$ included in ς, there exists a scenario $\delta \in \gamma_\varsigma(\omega)$ such that δ is playable in s, in which case, $\ell_\varsigma(\omega) = \gamma_\varsigma(\omega)$ (see Definition 5.24). Property 5.25 is directly derived from this definition and the extension of ℓ_ς to a set of objective sequences is given in Definition 5.26.

Definition 5.24 (**ℓ_ς : OS $\mapsto \wp$(Sce)).**

$$\ell_\varsigma(\omega) \triangleq \begin{cases} \gamma_\varsigma(\omega) & \text{if } \forall s \in L, s \subseteq \varsigma, \exists \delta \in \gamma_\varsigma(\omega), \text{support}(\delta) \subseteq s \\ \varnothing & \text{otherwise.} \end{cases}$$

Property 5.25. $\varsigma' \subseteq \varsigma \wedge \ell_\varsigma(\omega) \neq \varnothing \Longrightarrow \ell_{\varsigma'}(\omega) \neq \varnothing$.

Definition 5.26 (**ℓ_ς : $\wp$(OS) $\mapsto \wp$(Sce)).** $\ell_\varsigma(\Omega) \triangleq \{\delta \in \ell_\varsigma(\omega) \mid \omega \in \Omega\}$.

Given an objective P, $\text{maxCont}_\varsigma(\Sigma(P), P)$ (see Definition 5.27) is the set of processes of sort $\Sigma(P)$ that may be encountered during the resolution of P. We then define the saturated abstract structure $\lceil\mathscr{B}_\varsigma^\omega\rceil = (\lceil\text{Req}_\varsigma^\omega\rceil, \lceil\text{Sol}_\varsigma^\omega\rceil, \lceil\text{Cont}_\varsigma^\omega\rceil)$ (see Definition 5.28) similarly to $\lceil\mathscr{A}_\varsigma^\omega\rceil$ (see Definition 5.8), except that $\lceil\text{Sol}_\varsigma^\omega\rceil$ can arbitrarily select bounce sequences to resolve an objective, and that $\lceil\text{Cont}_\varsigma^\omega\rceil$ reflects maxCont_ς instead of minCont_ς. It appears that if $\lceil\mathscr{B}_\varsigma^\omega\rceil$ contains no cycle, and if all referenced objectives have at least one solution, then a top-down resolution of any referenced objective succeeds in every state of ς. This is stated by Theorem 5.29, which provides sufficient conditions for the concretisation of an objective sequence in a given context.

Definition 5.27 (maxCont_ς **: $\Sigma \times$ Obj $\mapsto \wp$(Proc)).**

$$\text{maxCont}_\varsigma(a, P) \triangleq \{p \in \textbf{Proc} \mid \exists ps \in \textbf{BS}^\wedge(P), \exists b_i \in ps, b = a \wedge p = b_i \\ \vee b \neq a \wedge p \in \text{maxCont}_\varsigma(a, b_j \Rsh^* b_i) \wedge b_j \in \varsigma[b])\}.$$

Definition 5.28 (**$\lceil\mathscr{B}_\varsigma^\omega\rceil$).** The abstract structure

$$\lceil\mathscr{B}_\varsigma^\omega\rceil = (\lceil\text{Req}_\varsigma^\omega\rceil, \lceil\text{Sol}_\varsigma^\omega\rceil, \lceil\text{Cont}_\varsigma^\omega\rceil)$$

is defined as

$$\lceil\mathscr{B}_\varsigma^\omega\rceil \triangleq \text{lfp}\{\mathscr{B}_\varsigma^\omega\} \left(\mathscr{B}_\varsigma^\omega \mapsto \mathscr{B}^\omega_{\varsigma \Cap \text{procs}(\mathscr{B}_\varsigma^\omega)}\right),$$

with

$$\mathscr{B}^\omega_\varsigma = (\overline{\mathrm{Req}^\omega_\varsigma}, \overline{\mathrm{Sol}^\omega_\varsigma}, \overline{\mathrm{Cont}^\omega_\varsigma})$$

and

$$\begin{aligned}
\overline{\mathrm{Req}^\omega_\varsigma} &\stackrel{\Delta}{=} \{(a_i, a_j \Rsh^* a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in \overline{\mathrm{Sol}^\omega_\varsigma}, a_i \in ps \\
&\qquad\qquad \vee \exists n \in \mathbb{I}^\omega, \mathrm{bounce}(\omega) = a_i\} \\
\overline{\mathrm{Sol}^\omega_\varsigma} &\stackrel{\Delta}{\subseteq} \{(P, ps) \in \mathbf{Obj} \times \wp(\mathbf{Proc}) \mid \exists (a_i, P) \in \overline{\mathrm{Req}^\omega_\varsigma} \wedge ps \in \mathbf{BS}^\wedge(P)\} \\
\overline{\mathrm{Cont}^\omega_\varsigma} &\stackrel{\Delta}{=} \{(P, q \Rsh^* \mathrm{bounce}(P)) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in \overline{\mathrm{Sol}^\omega_\varsigma} \\
&\qquad\qquad \wedge q \in \mathrm{maxCont}_\varsigma(\Sigma(P), P)\}.
\end{aligned}$$

Theorem 5.29 (under-approximation). If the graph $\lceil \mathscr{B}^\omega_\varsigma \rceil$ has no cycles and all referenced objectives have at least one solution, then $\ell_\varsigma(\omega) \neq \emptyset$.

Proof. We use $max\varsigma = \varsigma \Cap \mathrm{procs}(\lceil \mathscr{B}^\omega_\varsigma \rceil)$ to denote the context handled by $\lceil \mathscr{B}^\omega_\varsigma \rceil$. By induction on the acyclic graph $\lceil \mathscr{B}^\omega_\varsigma \rceil$, we prove that $\forall s \in L, s \sqsubseteq max\varsigma$, for each objective P referenced in $\lceil \mathscr{B}^\omega_\varsigma \rceil$ such that $\mathrm{target}(P) \in s$, $\exists \delta \in \ell_s(P)$ and $\mathrm{end}(\delta) \sqsubseteq max\varsigma$.

— $(P, \emptyset) \in \lceil \mathrm{Sol}^\omega_\varsigma \rceil \Rightarrow$ either $\mathrm{target}(P) = \mathrm{bounce}(P)$ (thus $\delta = \varepsilon$), or $\forall \zeta \in \mathbf{BS}(P), \zeta \in \mathbf{Sce} \wedge \Sigma(\zeta) = \{\Sigma(P)\}$, thus $\delta = \zeta$.

— We assume that all objectives that are children of P are concretisable (no cycles). If $\exists Q \in \lceil \mathrm{Cont}^\omega_\varsigma \rceil$, then, by hypothesis, $\ell_s(\mathrm{target}(P) \Rsh^* \mathrm{target}(Q) :: Q) \neq \emptyset$, thus $\ell_s(P) \neq \emptyset$. Otherwise, by Definition 5.27, the concretisations of children of P do not require any process of sort $\Sigma(P)$. Also, there exists $\zeta \in \mathbf{BS}(P)$ such that $(P, \zeta^\wedge) \in \lceil \mathrm{Sol}^\omega_\varsigma \rceil$. By hypothesis,

$$\forall n \in \mathbb{I}^\zeta, \exists \delta^n \in \ell_{s^{n-1}}(\star \Rsh^* \mathrm{hitter}(\zeta_n))$$

with either $s^{n-1} = s$ if $n = 1$ or

$$s^{n-1} = s \cdot \delta^1 \cdot \ldots \cdot \delta^{n-1}.$$

Moreover, $\Sigma(P) \notin \Sigma(\delta^n)$ (by Definition 5.27). Therefore,

$$\delta = \delta^1 :: \zeta^1 :: \ldots :: \delta^n :: \zeta^n \in \ell_s(P)$$

and $\mathrm{end}(\delta) \sqsubseteq max\varsigma$.

Finally, as $\ell_{max\varsigma}(\omega) \neq \emptyset$, we have $\ell_\varsigma(\omega) \neq \emptyset$ (see Property 5.25). □

From the proof of the above theorem, we define $\mathrm{endProc}(\lceil \mathscr{B}^\omega_\varsigma \rceil, P)$ (see Definition 5.30) as the maximum set of processes a scenario built by Theorem 5.29 may end with (see Corollary 5.31). This allows a straightforward extension of Theorem 5.29 to take the sequentiality of objectives into account (see Corollary 5.32).

Definition 5.30 (endProc : $\mathbb{B} \times$ **Obj** $\mapsto \wp($**Proc**$)$).

$$\begin{aligned}\mathrm{endProc}(\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil, P) \triangleq \{ p \in \mathbf{Proc} \mid\ & \Sigma(p) = \Sigma(P) \wedge p = \mathrm{bounce}(P) \\ & \wedge \Sigma(p) \neq \Sigma(P) \\ & \wedge (\exists (P, Q) \in \lceil \mathrm{Cont}^{\omega}_{\varsigma} \rceil, p \in \mathrm{endProc}(\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil, \\ & \qquad \mathrm{target}(P) \Rsh^{*} \mathrm{target}(Q)) \\ & \qquad \vee\, p \in \mathrm{endProc}(\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil, Q) \\ & \vee\, \exists (P, ps) \in \lceil \mathrm{Sol}^{\omega}_{\varsigma} \rceil, \exists b_i \in ps, \exists b_j \in \varsigma[b], \\ & \qquad p \in \mathrm{endProc}(\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil, b_j \Rsh^{*} b_i)) \}.\end{aligned}$$

Corollary 5.31. Given $P \in$ **Obj**, if Theorem 5.29 is satisfied with $\ell_{\varsigma}(P) \neq \emptyset$, then $\forall s \in L, s \subset \varsigma, \exists \delta \in \ell_{\varsigma}(P)$ such that $\mathrm{end}(\delta) \subseteq \mathrm{endProc}(\mathscr{B}^{P}_{\varsigma}, P)$.

Corollary 5.32. Given $P \in$ **Obj**, and $\omega \in$ **OS**, if Theorem 5.29 is satisfied with $\ell_{\varsigma}(P) \neq \emptyset$, and if Theorem 5.29 is satisfied with $\ell_{\varsigma'}(\omega) \neq \emptyset$, where $\varsigma' = \varsigma \Cap \mathrm{endProc}(\mathscr{B}^{P}_{\varsigma}, P)$, then Theorem 5.29 is satisfied with $\ell_{\varsigma}(P \oplus \omega) \neq \emptyset$.

5.4.1. *Implementation.* The computation of the saturated abstract structure $\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil$ works by progressive addition of relations between objectives and processes, which are found by several traversals of the abstract structure. This gives a maximal complexity that is polynomial in the size of the abstract structure. Checking the conditions for Theorem 5.29 is linear in the size of the obtained abstract structure. It is worth noting that arbitrarily selecting solutions for objectives in $\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil$ prevents spurious saturations and may increase the satisfaction of the above theorem, but potentially increases the complexity of checking (as several combinations of solutions can be tested).

5.4.2. *Examples.* Figure 9 shows two examples of the application of Theorem 5.29 on the Process Hitting example of Figure 1.

5.4.3. *Discussion.* Corollary 5.32 suggests that testing a refined objective sequence may be more conclusive than testing the original objective sequence. Future work may use a graph analysis of $\lceil \mathscr{B}^{\omega}_{\varsigma} \rceil$ to determine which refined objective sequences are good candidates for satisfying Theorem 5.29, thereby increasing the conclusiveness of the method.

6. Application to Biological Regulatory Networks

In order to give a practical direct application of the contributions of this paper, we briefly describe the results obtained on the formal analysis of large Biological Regulatory Networks (BRNs). The details of how we have used Process Hitting to model BRNs are given in Paulevé *et al.* (2011b) and we will only give an informal overview of it here – see Section 6.1.

The analysis (see Section 6.2) is performed on a real biological model extracted from the literature, and shows an impressive enhancement in terms of the tractability of the approach compared with other standard model-checking techniques.

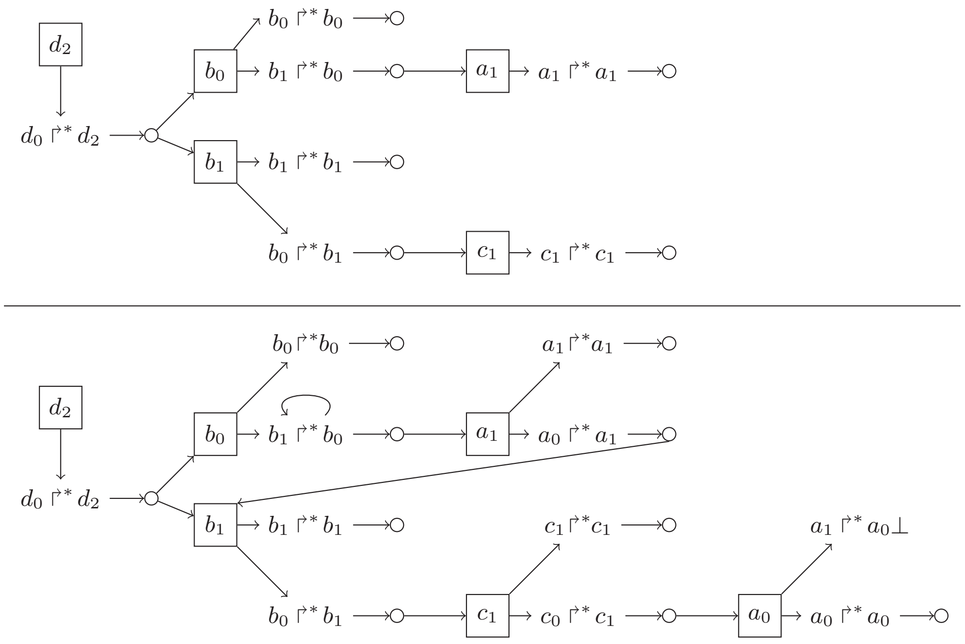
d_2
$d_0 \upharpoonright^* d_2$
b_0
b_1
$b_0 \upharpoonright^* b_0$
$b_1 \upharpoonright^* b_0$
a_1
$a_1 \upharpoonright^* a_1$
$b_1 \upharpoonright^* b_1$
$b_0 \upharpoonright^* b_1$
c_1
$c_1 \upharpoonright^* c_1$
d_2
$d_0 \upharpoonright^* d_2$
$b_0 \upharpoonright^* b_0$
$a_1 \upharpoonright^* a_1$
b_0
$b_1 \upharpoonright^* b_0$
a_1
$a_0 \upharpoonright^* a_1$
b_1
$b_1 \upharpoonright^* b_1$
$c_1 \upharpoonright^* c_1$
$a_1 \upharpoonright^* a_0 \perp$
$b_0 \upharpoonright^* b_1$
c_1
$c_0 \upharpoonright^* c_1$
a_0
$a_0 \upharpoonright^* a_0$

Figure 9. The top diagram shows the saturated abstract structure $\lceil \mathscr{B}_{\varsigma}^{\omega} \rceil$ from the Process Hitting in Figure 1 with $\omega = d_0 \upharpoonright^* d_2$ and $\varsigma = \langle a_1, b_1, c_1, d_0 \rangle$, and the bottom diagram shows it with $\varsigma = \langle a_0, b_1, c_0, d_0 \rangle$. By Theorem 5.29, ω is concretisable in $\langle a_1, b_1, c_1, d_0 \rangle$. At this stage, our procedure is inconclusive for the concretisability of ω in $\langle a_0, b_1, c_0, d_0 \rangle$.

6.1. *From Biological Regulatory Networks to Process Hittings*

We first sketch the modelling of a discrete BRN in the Process Hitting framework. Basically, for each component there is a corresponding sort, and for each state of the components there is a corresponding process. If a component a at state i activates a component b at state j, an action $a_i \to b_j \upharpoonright b_k$ is added, where b_k is the state of b after activation. The inhibition is modelled similarly. The realisation of boolean functions between nodes is modelled using a dedicated sort, and is illustrated in Figure 10. The full formalisation of this translation can be found in Paulevé *et al.* (2011b).

6.2. *T-cell receptor signalling pathway (94 components)*

The biological system presented in Saez-Rodriguez *et al.* (2007) models the T-cell receptor (TCR) signalling pathway, the behaviours of which reveal an activation of transcription factors controlling the cell's fate: for example, deciding whether it proliferates or not. This model is an extension of an earlier BRN model relating 40 components (Klamt *et al.* 2006).

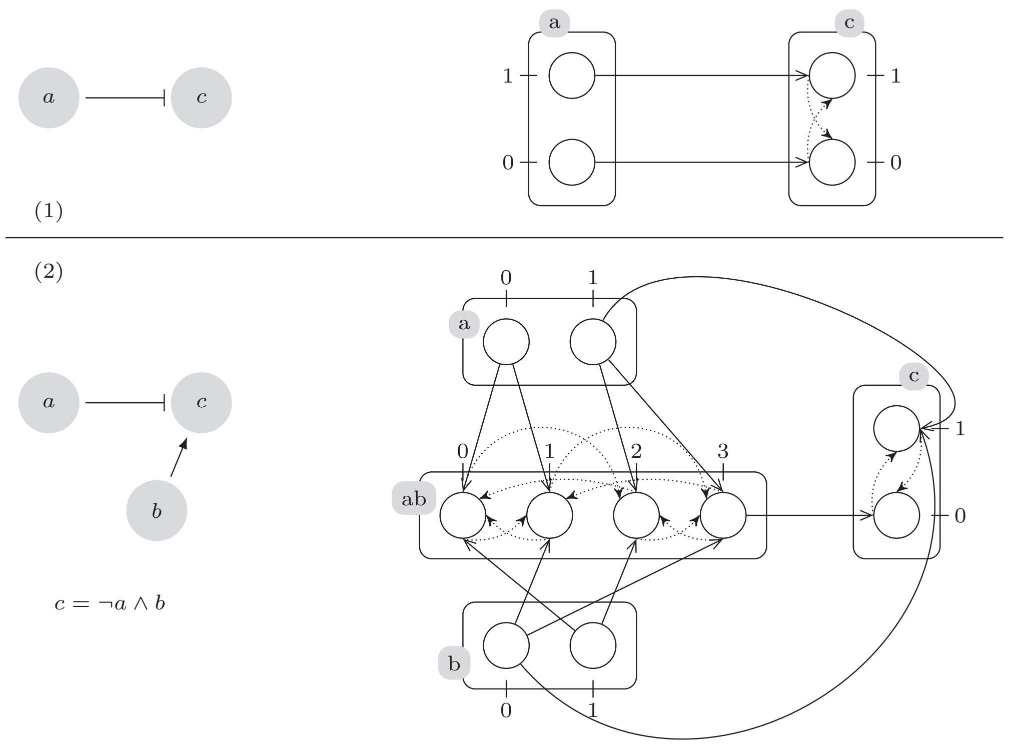

Figure 10. The diagrams on the right-hand side show examples of Process Hittings from BRNs with the interaction graphs shown on the left-hand side. The upper diagrams show simple inhibition of c by a, and the lower diagrams show a boolean function between a (inhibitor) and b (activator) on c, where ab reflects the state of the sorts a and b. In this case, ab_3 reflects the state $\langle a_0, b_1\rangle$ (and, ab_0 the state $\langle a_1, b_0\rangle$, ab_1 the state $\langle a_1, b_1\rangle$, and ab_2 the state $\langle a_0, b_0\rangle$).

The Process Hitting model[†] of this system is composed of 1124 actions between 448 processes splitted in 133 sorts (the largest sort has 16 processes). The total number of states of this model is 2^{194} ($\approx 2 \cdot 10^{58}$).

We have carried out experiments on independent reachability decisions using all possible input combinations (components CD45, CD8, TCRlig) with each output (components SRE, AP1, CRE, NFkB, NFAT, Cyc1, p21c p27k, FKHR, BclXL). *All result in conclusive decisions*. Note that we have only exploited the approximations defined by Theorem 5.7 (see Section 5.1) and Theorem 5.29 (see Section 5.4), resulting in quite simple dynamics for the independant reachability of components.

Computation times are around a hundredth of a second on a 3GHz processor with 2GB of RAM. To give a comparison, we did the same experiments with a standard symbolic model-checking method using state-space compression based on Hierarchical Set Decision Diagrams (SDDs) (Hamez *et al.* 2009), using the libDDD framework[‡], which is known

[†] The model and implementation are available at http://process.hitting.free.fr.
[‡] See http://ddd.lip6.fr for background and details of libDDD.

for its good performance. For the majority of reachability decisions, the program runs out of memory; for the others, computation times range from several seconds to hours. This shows the remarkable efficiency of our method, based on abstract interpretation.

7. Discussion

Process Hitting is a recently proposed framework suitable for modelling the dynamics of BRNs with discrete values. In Process Hitting, components are represented as sorts, and their levels as processes: at any time, one and only one process of each sort is present. The successive states of a component within the system are enclosed in the so-called sort. The replacement of one process by another of the same sort (that is, a level change of a component), is conditional on the presence of at most one other process, of any sort.

Thanks to the particular structure of Process Hitting models, a powerful static analysis by abstract interpretation has been developed to decide the successive reachability of processes, and hence of component levels in the context of BRN modelling. The computation is done by over- and under-approximation of the decision, and may turn out to be inconclusive. It exploits two complementary abstractions of scenarios in Process Hittings (by objective and bounce sequences). Several refinements of an objective sequence are then defined to provide the details of the steps required for its concretisability. These refinements are exploited to derive necessary or sufficient conditions for the process reachability satisfaction. Further work may improve the conclusiveness of our method, as discussed in Section 5.4. Also, the link between the conclusiveness of the developed method and the structure of the interaction graph of the BRN could be studied.

The implementations of the approximations presented use an abstract structure that is guaranteed to have a size nearly polynomial in the total number of processes. On the one hand, the computation of the refinements can be exponential in the number of processes within a single sort; but, on the other hand, the computations of the decisions are polynomial in the size of the abstract structure. Hence, we expect efficient analyses when the number of processes within a sort is limited, but a large number of sorts need to be handled.

This new and original approach was applied to the analysis of a large BRN relating 94 components. Response times are very fast (around a hundredth of a second on a desktop computer), showing a promising scalability of our method. We have compared the results with those produced by standard symbolic model checking techniques, which regularly fail to analyse the model because of the state-space explosion. To our knowledge, ours is the first successful application of model checking to BRNs of such a size.

7.1. *Related work*

There has been recent work on the fast computation of the set of reachable components within Kappa models, which use a rule-based language (Danos *et al.* 2008). In the general case, this set is over-approximated, and such an analysis does not enable us to decide the successive reachability of processes, as we have done in this paper. The work presented in Alimonti *et al.* (2010) on so-called T-Paths within Petri Nets establishes structural properties within Petri Nets to derive either necessary or sufficient conditions for place marking reachability. This follows a similar approach to ours. Finally, Pilegaard *et al.* (2005) applies static analysis techniques of bioambient models to study the behaviour of biological systems.

Future work will investigate other applications of the refinements of the abstraction interpretations of Process Hitting presented here. In particular, since they extract the causality between process changes, they identify processes required for a certain process reachability. This kind of analysis may lead to a control of the studied system by acting upon these *key* processes: for example, using knockdown techniques on a key gene to prevent a cascade of gene activation (gene therapy).

Another research direction is the incorporation of quantitative aspects within the decision of process reachability presented here: for example, to calculate the probability of reaching a given process within a given time interval.

References

Alimonti, P., Feuerstein, E., Laura, L. and Nanni, U. (2011) Linear time analysis of properties of conflict-free and general petri nets. *Theoretical Computer Science* **412** (4–5) 320–338.

Aracena, J. (2008) Maximum number of fixed points in regulatory boolean networks. *Bulletin of Mathematical Biology* **70** (5) 1398–1409.

Bernot, G., Cassez, F., Comet, J.-P., Delaplace, F., Müller, C. and Roux, O. (2007) Semantics of biological regulatory networks. *Electronic Notes in Theoretical Computer Science* **180** (3) 3–14.

Bernot, G., Comet, J.-P. and Khalis, Z. (2008) Gene regulatory networks with multiplexes. *European Simulation and Modelling Conference Proceedings* 423–432.

Brand, D. and Zafiropulo, P. (1983) On communicating finite-state machines. *Journal of the ACM* **30** 323–342.

Clarke, E. M. and Emerson, E. A. (1981) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logics of Programs. *Springer-Verlag Lecture Notes in Computer Science* **131** 52–71.

Cousot, P. and Cousot, R. (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages – POPL '77* 238–252.

Cousot, P. and Cousot, R. (1992) Abstract interpretation frameworks. *Journal of Logic and Computation* **2** (4) 511–547.

Danos, V., Feret, J., Fontana, W. and Krivine, J. (2008) Abstract interpretation of cellular signalling networks. In: Logozzo, F., Peled, D. and Zuck, L. (eds.) Verification, Model Checking, and Abstract Interpretation. *Springer-Verlag Lecture Notes in Computer Science* **4905** 83–97.

Hamez, A., Thierry-Mieg, Y. and Kordon, F. (2009) Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. *Fundamenta Informaticae* **94** (3-4) 413–437.

Klamt, S., Saez-Rodriguez, J., Lindquist, J., Simeoni, L. and Gilles, E. (2006) A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics* **7** (1) 56.

Paulevé, L., Magnin, M. and Roux, O. (2011a) Abstract Interpretation of Dynamics of Biological Regulatory Networks. In: Proceedings of The First International Workshop on Static Analysis and Systems Biology (SASB 2010). *Electronic Notes in Theoretical Computer Science* **272** 43–56.

Paulevé, L., Magnin, M. and Roux, O. (2011b) Refining dynamics of gene regulatory networks in a stochastic π-calculus framework. In: Priami, C., Back, R.-J., Petre, I. and de Vink, E. (eds.) Transactions on Computational Systems Biology XIII. *Springer-Verlag Lecture Notes in Computer Science* **6575** 171–191.

Paulevé, L. and Richard, A. (2010) Topological Fixed Points in Boolean Networks. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* **348** (15–16) 825–828.

Pilegaard, H., Nielson, F. and Nielson, H. R. (2005) Static analysis of a model of the LDL degradation pathway. In: Plotkin, G. (ed.) Third International Workshop on Computational Methods in Systems Biology (CMSB'05), University of Edinburgh.

Remy, É., Ruet, P. and Thieffry, D. (2008) Graphic requirements for multistability and attractive cycles in a boolean dynamical framework. *Advances in Applied Mathematics* **41** (3) 335–350.

Richard, A. (2010) Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics* **44** (4) 378–392.

Richard, A. and Comet, J.-P. (2007) Necessary conditions for multistationarity in discrete dynamical systems. *Discrete Applied Mathematics* **155** (18) 2403–2413.

Richard, A., Comet, J.-P. and Bernot, G. (2006) Formal Methods for Modeling Biological Regulatory Networks. In: Gabbar, H. A. (ed.) *Modern Formal Methods and Applications*, Springer-Verlag 83–122.

Saez-Rodriguez, J. *et al.* (2007) A logical model provides insights into t cell receptor signaling. *PLoS Comput Biol* **3** (8) e163.

Thomas, R. (1973) Boolean formalization of genetic control circuits. *Journal of Theoretical Biology* **42** (3) 563–585.