Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 299 (2013) 33–51

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Under-approximation of Reachability in Multivalued Asynchronous Networks

Maxime Folschette,[a,1] Loïc Paulevé,[b] Morgan Magnin[a] and Olivier Roux[a]

[a] *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.*

[b] *ETH Zürich, Switzerland.*

BISON group, Automatic Control Laboratory, ETH Zürich Physikstrasse 3, 8092 Zurich, Switzerland.

Abstract

The Process Hitting is a recently introduced framework designed for the modelling of concurrent systems. Its originality lies in a compact representation of both components of the model and its corresponding actions: each action can modify the status of a component, and is conditioned by the status of at most one other component. This allowed to define very efficient static analysis based on local causality to compute reachability properties. However, in the case of cooperations between components (for example, when two components are supposed to interact with a third one only when they are in a given configuration), the approach leads to an over-approximated interleaving between actions, because of the pure asynchronous semantics of the model.
To address this issue, we propose an extended definition of the framework, including priority classes for actions. In this paper, we focus on a restriction of the Process Hitting with two classes of priorities and a specific behaviour of the components, that is sufficient to tackle the aforementioned problem of cooperations. We show that this class of Process Hitting models allows to represent any Asynchronous Discrete Networks, either Boolean or multivalued. Then we develop a new refinement for the under-approximation of the static analysis to give accurate results for this class of Process Hitting models. Our method thus allows to efficiently under-approximate reachability properties in Asynchronous Discrete Networks; it is in particular conclusive on reachability properties in a 94 components Boolean network, which is unprecedented.

Keywords: qualitative modelling, model abstraction, static analysis, asynchronous network

1 Introduction

Discrete modelling frameworks for biological networks is an active research field where formal methods have proved that they were very powerful. Such a work started in the seventies. It was later enriched in many directions and widely used to elucidate many biological questions. Among these questions, a major one is to

[1] Maxime.Folschette@irccyn.ec-nantes.fr

http://dx.doi.org/10.1016/j.entcs.2013.11.004

understand precisely how biological systems evolve and behave; why and how they change their usual behaviours... This leads to questions about the reachability (possible or inevitable) of some states. The ultimate goal is to discover how it could be possible to prevent biological systems from reaching some pathological states.

Of course, such formal models on which analyses are performed are abstract representations of the actual studied systems. They are associated with parameters that have to be synthesised to give the most faithful representation of the real systems with their observed behaviours. As a matter of fact, the abstractions we get are more or less rough or accurate. Usual formal frameworks for such modelling activities are state-transition systems or process algebras. We developed a quite similar framework named the Process Hitting [10], consisting in a restriction where the evolution of a component is determined by the state of at most one other component that does not evolve. In a sense, these kind of actions are of the form $X + Y \to X + Z$ where X behaves like a catalyst molecule that "hits" another molecule Y and changes it into Z, without being itself changed. Assuming catalysts are always available, this can represent any biochemical system made of monomolecular reactions, and can also represent catalytic networks such as metabolic networks. Our motivation behind this framework was to design a model and analysis techniques adapted to biological modelling. These analyses avoid to build the whole state space, which allows to tackle very large systems (that would have led to a huge number of states, hopelessly too huge to be analysed). They are based on the fact that most biological models have few levels of expression per component: in Boolean networks [8,13] there are only two levels per component and in its multivalued equivalent, Asynchronous Discrete Networks [5], components rarely have more that four levels.

Besides, one further objective of our work is now to improve the accuracy of the description of the studied systems dynamics. The idea for this is to introduce timing features into models: we are interested in taking into account some knowledge about the relative length of some phenomena as it is a way to refute some models (or parameters) that are inconsistent with the observed dynamic behaviours. In this paper, we are dealing with these timing properties through priorities, that are based on the simple founding idea that prioritised actions have to be processed before the other ones. Indeed, due to the Process Hitting framework restrictions, bimolecular reactions are not immediately available, but one can simulate them with an encoding called "cooperation". That encoding however introduces extra reactions, and this is where the priorities become useful, if not necessary. The extra reactions can be given "infinite speed" (high priority) so that they do not affect the behaviour of "normal" (low priority) reactions, including the bimolecular ones.

Until now, such a priority scheduling of the actions was not studied extensively in the different formal modelling frameworks dedicated to systems biology. Nevertheless, such an attempt has been carried out for Petri nets by F. Bause [1], and the concept of priority relations among the transitions of a network has also more recently been introduced by A. K. Wagler *et al.* [15,14] in order to allow modelling deterministic systems for biological applications. The concept of priority is much straightforward in the approach of process algebras as it was shown by R. Cleaveland

and M. Hennessy in [2,4] and their abstractions and equivalences were studied in [3]. It was later extended for applications in the field of systems biology by M. John *et al.* [7].

Contributions

Since our formalism (the Process Hitting) can be considered as a subset of Calculus of Communicating Systems, our work is related to such semantic ramifications of extending traditional process algebras with the concepts of priority that allow for some transitions to be given precedence over others. The concept is derived in two directions: dynamic versus static, the difference being naturally that the former one refers to a semantics where priority values may change during execution according to some evolution rules. In our work, actions exhibit a two-level static priority structure, some of them being designated as "prioritised" and others as "unprioritised".

In this paper, we introduce a new extension to the semantics of Process Hitting by partitioning actions into classes of priorities. One of the objectives is to reach an accurate representation of cooperating components in the model, that was not fulfilled with the initial semantics. We then develop an efficient under-approximation of the reachability of the state of components on a subclass of this new framework, thus allowing to compute efficient static analysis. This local reachability under-approximation can be also easily extended to study the reachability of a global state. Finally, as the subclass of models studied proves to be bisimilar to Asynchronous Discrete Networks, we state to have developed an efficient method to compute the reachability of a state and thus study the behaviour of such models.

The method developed in this paper has been implemented into the existing Pint library and tested on a large-scale biological model containing 94 components. The under-approximation turned out to be conclusive in all cases and results were computed in hundredths of seconds, thus overtaking the efficiency of usual model-checkers.

Our paper is organised as follows. The Process Hitting framework is defined in section 2; we introduce static analysis of the Process Hitting in section 3; section 4 illustrates the approach on an example before the discussion and conclusion in section 5.

Notations

If A is a finite set, $|A|$ is the cardinality of A and $\wp(A)$ is the power set of A. $\mathbb{N}$ is the set of natural numbers, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$ is the set of positive natural numbers and $[\![x; y]\!] = \{x, x+1, \dots, y-1, y\}$ is the set of natural numbers from x to y included. If $x = (x_i)_{i \in [\![1;n]\!]}$ is a sequence of elements indexed by $i \in [\![1; n]\!]$, $\mathbb{I}^x = [\![1; n]\!]$ is the set of indexes of this sequence. We also denote by ε the empty sequence. If A and B are sets, $f : A \to B$ denotes an application f that maps the elements of A to elements of B. $\mathbf{lfp}\{x_0\}\,(x \mapsto x')$ is the least fixed point of the function $x \mapsto x'$ which is greater than x_0. The Cartesian product is denoted $\times$.

2 The Process Hitting Framework

We give in this section the definition and the semantics of the Process Hitting (PH) with priorities, which is an extension of the basic semantics given in [10]. Then we describe the modelling of cooperation between components and discuss how the new aforementioned semantics makes this modelling more accurate. Finally, in order to perform a static analysis adapted to this new semantics, we give several criteria to restrict the class of models that we can study, and give several theorems that follow. This class of models is equivalent to Asynchronous Discrete Networks.

2.1 Definition of the Process Hitting with k classes of priorities

A PH with k classes of priorities (Def. 2.1), also simply called "PH" in the following when it is not ambiguous, gathers a finite number of concurrent *processes* divided into a finite set of *sorts*. A process belongs to a unique sort and is noted a_i where a is the sort and i the identifier of the process within the sort a. Each process stands for a kind of "activity level" of its sort; a state of the PH thus corresponds to a set of processes containing exactly one process of each sort.

The concurrent interactions between processes are defined by a set of *actions* divided into classes of priorities. Actions describe the replacement of a process by another of the same sort conditioned by the presence of at most one other process and by the fact that no other action of higher priority can be played in the considered state of the PH. An action is denoted by $a_i \to b_j \not\uparrow b_k$ 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 $h = a_i \to b_j \not\uparrow b_k$ is read as "a_i *hits* b_j to make it bounce to b_k", and a_i, b_j, b_k are called respectively *hitter*, *target* and *bounce* of the action, and can be referred to as $\mathsf{hitter}(h), \mathsf{target}(h), \mathsf{bounce}(h)$, respectively.

Definition 2.1 [Process Hitting with k classes of priorities] If $k \in \mathbb{N}^*$, a *Process Hitting with k classes of priorities* is a triplet $\mathcal{PH} = (\Sigma; \mathcal{L}; \mathcal{H}^{\langle k \rangle})$, where $\mathcal{H}^{\langle k \rangle} = (\mathcal{H}^{(1)}; \dots; \mathcal{H}^{(k)})$ is a k-tuple and:

- $\Sigma \triangleq \{a, b, \dots\}$ is the finite set of *sorts*,
- $\mathcal{L} \triangleq \underset{a \in \Sigma}{\times} \mathcal{L}_a$ is the finite set of *states*, where $\mathcal{L}_a = \{a_0, \dots, a_{l_a}\}$ is the finite set of *processes* of sort $a \in \Sigma$ and $l_a \in \mathbb{N}^*$. Each process belongs to a unique sort: $\forall (a_i; b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$,
- $\forall n \in [\![1; k]\!], \mathcal{H}^{(n)} \triangleq \{a_i \to b_j \not\uparrow b_l \mid (a; b) \in \Sigma^2 \wedge (a_i; b_j; b_l) \in \mathcal{L}_a \times \mathcal{L}_b \times \mathcal{L}_b \wedge b_j \neq b_l \wedge a = b \Rightarrow a_i = b_j\}$ is the finite set of *actions of priority n*.

We call $\mathbf{Proc} \triangleq \bigcup_{a \in \Sigma} \mathcal{L}_a$ the set of all processes, and $\mathcal{H} \triangleq \bigcup_{n \in [\![1;k]\!]} \mathcal{H}^{(n)}$ the set of all actions.

The sort of a process a_i is referred to as $\Sigma(a_i) = a$. Given a state $s \in \mathcal{L}$, the process of sort $a \in \Sigma$ present in s is denoted by $s[a]$, that is the a-coordinate of the state s. If $a_i \in \mathcal{L}_a$, we define the notation $a_i \in s \overset{\Delta}{\Leftrightarrow} s[a] = a_i$. The override of a state s by a

process a_i is defined in Def. 2.2 as the same state in which the process of sort a has been replace by a_i, which then allows to define the dynamics of a PH in Def. 2.3.

Definition 2.2 [$\Cap : \mathcal{L} \times \mathbf{Proc} \to \mathcal{L}$] Given a state $s \in \mathcal{L}$ and a process $a_i \in \mathbf{Proc}$, $(s \Cap a_i)$ is the state defined by: $(s \Cap a_i)[a] = a_i \wedge \forall b \neq a, (s \Cap a_i)[b] = s[b]$. We also extend this definition to a set of processes ps given that all processes are from different sorts by the override of each process: $\forall as \subseteq \Sigma, \forall ps \in \bigtimes_{a \in as} \mathcal{L}_a, s \Cap ps = s \bigCap_{a_i \in ps} a_i$.

Definition 2.3 [Dynamics of a PH ($\to_{PH}$)] An action $h = a_i \to b_j \nearrow b_k \in \mathcal{H}^{(n)}$ of priority n is *playable* in $s \in \mathcal{L}$ if and only if $s[a] = a_i$, $s[b] = b_j$ and $\forall m < n, \forall g \in \mathcal{H}^{(m)}, \mathsf{hitter}(g) \notin s \vee \mathsf{target}(g) \notin s$. In such a case, $(s \cdot h)$ stands for the state resulting from the play of the action h in s and is defined by: $(s \cdot h) = s \Cap \mathsf{bounce}(h)$. Moreover, we denote: $s \to_{PH} (s \cdot h)$.

If $s \in \mathcal{L}$, a *scenario* δ from s is a sequence of actions of $\mathcal{H}$ that can be played successively in s. The set of all scenarios from s is noted $\mathbf{Sce}(s)$.

In Def. 2.4, we define the n-reduction of a given PH as the PH with n classes of priorities in which only actions of priority lower or equal to n are considered.

Definition 2.4 [PH n-reduction] If $\mathcal{PH} = (\Sigma; \mathcal{L}; \mathcal{H}^{\langle k \rangle})$ is a Process Hitting with k classes of priorities and $n \in [\![1; k]\!]$, we denote $\mathcal{PH}_{|n}$ the n-reduction of $\mathcal{PH}$. $\mathcal{PH}_{|n} = (\Sigma; \mathcal{L}; \mathcal{H}'^{\langle n \rangle})$ is a PH with n classes of priorities with:

$$\mathcal{H}'^{\langle n \rangle} = (\mathcal{H}^{(1)}; \ldots; \mathcal{H}^{(n)})$$

Furthermore, we denote: $\mathbf{Sce}_{|n}(s)$ the set of scenarios from s in $\mathcal{PH}_{|n}$.

Example 2.5 Fig. 1 gives an example of PH with 2 classes of priorities where:

$$\Sigma = \{a, b, c, ab\} \ ,$$
$$\mathcal{L}_a = \{a_0, a_1\} \ , \qquad \mathcal{L}_b = \{b_0, b_1\} \ ,$$
$$\mathcal{L}_c = \{c_0, c_1\} \ , \qquad \mathcal{L}_{ab} = \{ab_{00}, ab_{01}, ab_{10}, ab_{11}\} \ .$$

There also is especially: $\{ab_{11} \to c_0 \nearrow c_1, a_1 \to a_1 \nearrow a_0, a_0 \to b_0 \nearrow b_1\} \subseteq \mathcal{H}^{(2)}$.

2.2 Modelling cooperation

Cooperation between processes to make another process bounce can be expressed in PH by building a *cooperative sort*, as described in [10]. Fig. 1 shows an example of cooperation between processes a_1 and b_1 to make c_0 bounce to c_1: a cooperative sort ab is defined with 4 processes (one for each sub-state of the presence of processes a_1 and b_1). For the sake of clarity, the processes of ab are indexed using the sub-state they represent. Hence, ab_{10} represents the sub-state $\langle a_1, b_0 \rangle$, and so on. Each process of sort a and b hit ab to make it bounce to the process reflecting the status of the sorts a and b (e.g., $a_1 \to ab_{00} \nearrow ab_{10}$ and $a_1 \to ab_{01} \nearrow ab_{11}$). Then, to represent

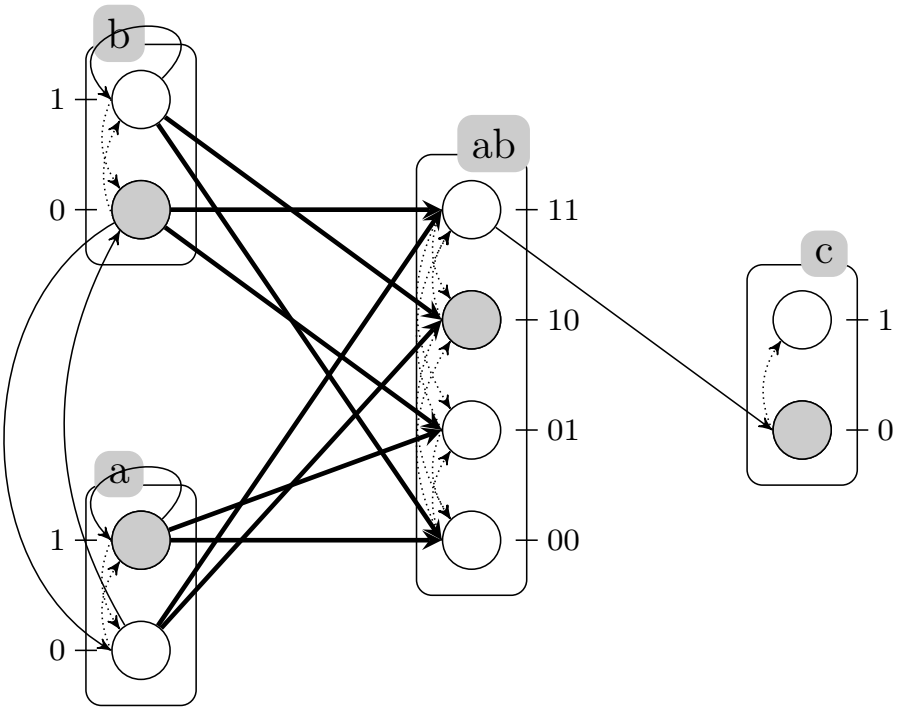

Figure 1. An example of PH with 2 classes of priorities. Sorts are represented as labelled boxes and processes as circles with their identifier on the side. Actions of $\mathcal{H}^{(1)}$ are represented by thick arrows and actions of $\mathcal{H}^{(2)}$ are represented by single arrows; the hit part of each action in drawn in plain line and the bounce part is in dotted line. Greyed processes stand for a possible state $s = \langle a_1, b_0, c_0, ab_{10} \rangle$.

the cooperation between a_1 and b_1, the process ab_{11} hits c_0 to make it bounce to c_1 instead of independent hits from a_1 and b_1.

We note that cooperative sorts are standard PH sorts and do not involve any special treatment regarding the semantics of related actions. Furthermore, it is possible to "factorise" cooperative sorts in order to decrease the number of processes created within each cooperative sort. For example, if three processes x_1, y_1 and z_1 cooperate, it is preferable to create a cooperative sort xy with 4 processes to state the presence of x_1 and y_1 and a second cooperative sort xyz with 4 processes to state the presence of xy_{11} and z_1, rather than a unique cooperative sort with 8 processes stating the presence of x_1, y_1 and z_1. This "factorisation" allows to prevent the combinatorial explosion of the number of processes in cooperative sorts, especially for cooperations between more than three processes. It may have computational consequences as the static analysis method developed in Sect. 3 does not suffer from the number of sorts but from the number of processes in each sort.

The construction of cooperation in PH allows to encode any Boolean function between cooperating processes [10]. Due to the introduction of priorities into the PH framework, it is possible to build cooperations with no temporal shift by defining actions updating the cooperative sorts with the highest class of priority. This allows to gain the same expressivity in PH than in Boolean networks, as stated in Subsect. 2.2. The aim of this paper is to allow the static analysis of the dynamics to be

handled on PH models comprising such higher priority actions updating cooperative sorts.

2.3 Restrictions

In the scope of this paper, we focus on a specific class of PH models. We define here the restrictions that lead to this class of models and show that they are equivalent to discrete networks.

Criterion 2.6 allows to distinguish two kinds of actions: *unprioritised actions* modelling the non-determinacy of biological evolutions and *prioritised actions* used to model non-biological behaviours in the model, namely the update of cooperative sorts. Criterion 2.7 states that the dynamics of the studied model $\mathcal{PH}$ contains no infinite sequence of prioritised actions. As these actions can be considered as non-biological and therefore instantaneous, we thus prevent the existence of any Zeno-like behaviour which would allow the play of an infinite sequence of prioritised actions in "zero time".

Criterion 2.6 (2 classes of priorities) *In this paper, we only consider Process Hitting with 2 classes of priorities:* $\mathcal{PH} = (\Sigma; \mathcal{L}; \mathcal{H}^{\langle 2 \rangle})$.

Criterion 2.7 (Bounded termination) *The dynamics of* $\mathcal{PH}_{|1}$ *contains no cycles:* $\exists N \in \mathbb{N}, \forall s \in \mathcal{L}, \forall \delta \in \mathbf{Sce}_{|1}(s), |\delta| \leq N$.

In Def. 2.8 we define a well-formed component as a sort that is hit only by unprioritised actions, or that no action hits.

Definition 2.8 [Well-formed component (Γ)] A sort $a \in \Sigma$ is a *well-formed component* if and only if:

$$\forall h \in \mathcal{H}, \Sigma(\mathsf{target}(h)) = a \Rightarrow \mathsf{prio}(h) = 2 \ .$$

We call Γ the set of well-formed components.

Def. 2.9 defines chains of prioritised actions, and Criterion 2.10 prevents the presence of cycles in these chains.

Definition 2.9 The set of chains of actions $\mathcal{H}(a, b)$ between two sorts $a, b \in \Sigma$ is defined as below:

$$\begin{aligned} \mathcal{H}(a,b) = \{(h_i)_{i \in [\![1;s+1]\!]} \in (\mathcal{H}^{(1)})^{s+1} \mid s \in \mathbb{N} \\ \wedge\, \Sigma(\mathsf{hitter}(h_1)) = a \wedge \Sigma(\mathsf{target}(h_{s+1})) = b \\ \wedge\, \forall i \in [\![1;s]\!], \Sigma(\mathsf{target}(h_i)) = \Sigma(\mathsf{hitter}(h_{i+1}))\} \end{aligned}$$

where $(\mathcal{H}^{(1)})^{s+1}$ is the Cartesian product of $s+1$ times the set $\mathcal{H}^{(1)}$.

Criterion 2.10 (Cycle-freeness in prioritised actions) *There is no cycles in chains of actions:* $\forall a \in \Sigma, \mathcal{H}(a, a) = \emptyset$.

In Def. 2.11 we define the notion of neighbouring sorts and actions of a given sort. The set of neighbouring sorts $V_\Sigma(a)$ of a sort a is the set of components that can interact with it through prioritised actions. The set of neighbouring actions $V_\mathcal{H}(a)$ of a is the set of prioritised actions influencing a. These definitions are permitted by the previous restrictions (Criterion 2.6 and 2.10).

Definition 2.11 $[V_\Sigma : \Sigma \to \wp(\Sigma);\ V_\mathcal{H} : \Sigma \to \wp(\mathcal{H})]$ For all sort $a \in \Sigma$,

$$V_\Sigma(a) = \{b \in \Gamma \mid \mathcal{H}(b,a) \neq \emptyset\}$$
$$V_\mathcal{H}(a) = \{h \in \mathcal{H}^{(1)} \mid \exists b \in V_\Sigma(a), \exists hs \in \mathcal{H}(b,a), h \in hs\}$$

Finally, we introduce the notion of local steady-state of a sort from a given state. This local steady-state is the set of processes towards which the sort tends to evolve to, and stay in, when playing only prioritised actions. We denote $\mathsf{lst}_s(a)$ this set of processes for a sort a in state s (Def. 2.12), and we derive from this the notion of well-formed cooperative sort (Def. 2.13). A well-formed cooperative sort models a cooperation between sorts as presented in Subsect. 2.2; therefore, it must be only hit by prioritised actions, in a way that its local steady-states efficiently represent all configurations of its neighbouring sorts.

Definition 2.12 $[\mathsf{lst} : \Sigma \to \wp(\mathbf{Proc})]$ For all $a \in \Sigma$,

$$\mathsf{lst}_s(a) = \{a_i \in \mathcal{L}_a \mid \exists \delta \in \mathbf{Sce}_{|1}(s), (s \cdot \delta)[a] = a_i \\ \wedge \forall b_i \to c_j \nearrow c_k \in V_\mathcal{H}(a), (s \cdot \delta)[b] \neq b_i \vee (s \cdot \delta)[c] \neq c_j\}$$

Of course, if $a \in \Gamma$, then $\mathsf{lst}_s(a) = \{s[a]\}$.

Definition 2.13 [Well-formed cooperative sorts (Δ)] A sort $a \in \Sigma$ is a *well-formed cooperative* sort if and only if:

(i) $\exists b \in \Sigma, \mathcal{H}(b,a) \neq \emptyset$

(ii) $\forall s \in \mathcal{L}, \exists a_i \in \mathcal{L}_a, \mathsf{lst}_s(a) = \{a_i\}$

(iii) $\forall a_i \in \mathcal{L}_a, \exists s \in \mathcal{L}, \mathsf{lst}_s(a) = \{a_i\}$

We call Δ the set of well-formed cooperative sorts.

Because of Def. 2.13(ii), we denote in the following: $\mathsf{lst}_s(a) = a_i$. Furthermore, because of Def. 2.13(iii), we denote $\mathsf{procState}(a_i)$ the set of sub-states represented by the process a_i of any cooperative sort a (Def. 2.14).

Definition 2.14 $[\mathsf{procState} : \mathbf{Proc} \to \wp(\mathbf{Proc})]$ If $a \in \Delta$ and $a_i \in \mathcal{L}_a$, we define:

$$\mathsf{procState}(a_i) = \{ps \in \bigtimes_{b \in V_\Sigma(a)} \mathcal{L}_b \mid \forall s \in \mathcal{L}, \mathsf{lst}_{s \Cap ps}(a) = a_i\}$$

In the following we simply write "component" (resp. "cooperative sort") instead of "well-formed component" (resp. "well-formed cooperative sort"). Finally, Criterion 2.15 states that the set of sorts of the considered PH must be divided between components and cooperative sorts.

Criterion 2.15 (Components & cooperative sorts partition)

$$\Sigma = \Gamma \cup \Delta \wedge \Gamma \cap \Delta = \emptyset$$

Example 2.16 The PH in Fig. 1 contains three components a, b and c and a cooperative sort ab that models cooperation between sorts a and b.

The criteria given in this subsection allow to define a class of PH models that is (weakly) bisimilar to Asynchronous Boolean Networks or, more generally, to the multivalued version of this framework, called Asynchronous Discrete Networks (ADN). A translation of ADN into PH is given in Appendix B, alongside with a demonstration of the weak bisimulation.

2.4 Consequences of the restrictions

In this subsection, we give several general theorems that can be derived from the restrictions of Subsect. 2.2, and which will help building the static analysis of Sect. 3.

We first denote by $\mathsf{update}(s)$ the state equivalent to s but in which all cooperative sorts are updated (Def. 2.17). This state is unique due to the properties of lst given in the previous subsection. Then, Theorem 2.18 states that from any state, there exists a scenario updating the cooperative sorts of this state.

Definition 2.17 $[\mathsf{update} : \mathcal{L} \to \mathcal{L}]$ For all $s \in \mathcal{L}$, we define:

$$\mathsf{update}(s) = s \Cap \{\mathsf{lst}_s(a) \mid a \in \Delta\} \ .$$

Theorem 2.18 $\forall s \in \mathcal{L}, \exists \delta \in \mathbf{Sce}_{|1}(s), s \cdot \delta = \mathsf{update}(s)$

Proof Let a be a cooperative sort so that $s[a] \neq \mathsf{lst}_s(a)$. Given the definition of $\mathsf{lst}_s(a)$, there exists a scenario δ updating a in s so that $\forall \delta' \in \mathbf{Sce}_{|1}(s \cdot \delta)$, $(s \cdot \delta \cdot \delta')[a] = \mathsf{lst}_s(a)$. As there is no cycle of actions between the cooperative sorts (Criterion 2.10) and given that an updated cooperative sort cannot evolve, at most $|\Delta|$ updates have to be performed. □

Theorem 2.19 states that for a given state s, and for any action $h = a_i \to b_j \Rsh b_k$ where a and b are components, if $s[a] = a_i$ and $s[b] = b_j$, then h can always be played after a series of hits (and these hits do not prevent it to be fired). Theorem 2.20 states the same if a is a cooperative sort, under the condition that a is updated in s.

Theorem 2.19 $\forall s \in \mathcal{L}, \forall a, b \in \Gamma, \forall h = a_i \to b_j \Rsh b_k \in \mathcal{H}$,
$(s[a] = a_i \wedge s[b] = b_j) \Rightarrow (\exists \delta \in \mathbf{Sce}_{|1}(s), (s \cdot \delta) \to_{PH} (s \cdot \delta \cdot h))$

Proof From Theorem 2.18, there exists a scenario δ with: $(s \cdot \delta) = \mathsf{update}(s)$. As $a, b \in \Gamma$, $a_i \in (s \cdot \delta)$ and $b_j \in (s \cdot \delta)$. Finally, by definition of $\mathsf{update}(s)$, no prioritised action can be played in $(s \cdot \delta)$, thus h can be played in $(s \cdot \delta)$. □

Theorem 2.20 $\forall s \in \mathcal{L}, \forall h = a_i \to b_j \Rsh b_k \in \mathcal{H}, a \in \Delta, b \in \Gamma$
$(s[a] = a_i \wedge s[b] = b_j \wedge \mathsf{lst}_s(a) = a_i) \Rightarrow (\exists \delta \in \mathbf{Sce}_{|1}(s), (s \cdot \delta) \to_{PH} (s \cdot \delta \cdot h))$

Proof Similar to the proof of Theorem 2.19; as $a_i \in \mathsf{lst}_s(a)$, $a_i \in (s \cdot \delta)$. □

3 Static Analysis

The aim of this section is to define the problem of reachability in a PH, and propose an under-approximation allowing to efficiently solve it. The static analysis presented here is inspired from [11].

3.1 Preliminary definitions

The reachability of a process a_j of a given sort a from another process a_i is called an objective and is denoted $a_i \Rsh^* a_j$ (Def. 3.1).

Definition 3.1 [Objective (**Obj**)] If $a \in \Gamma$, the reachability of a process a_j from a process a_i is called an *objective*, noted $a_i \Rsh^* a_j$. The set of all objectives is called $\mathbf{Obj} \triangleq \{a_i \Rsh^* a_j \mid a \in \Gamma \wedge (a_i, a_j) \in \mathcal{L}_a^2\}$. For an objective $P = a_i \Rsh^* a_j \in \mathbf{Obj}$, we define: $\Sigma(P) = a$, $\mathsf{target}(P) = a_i$, $\mathsf{bounce}(P) = a_j$, and P is said *trivial* if $a_i = a_j$.

We define an *objective sequence* as a sequence of objectives in which each objective target must be equal to the previous objective bounce of the same sort, if it exists. The set of all objective sequences is denoted by **OS**.

A context (Def. 3.2) extends the notion of state to a set of possible initial states. We also extend the override operator to contexts (Def. 3.3).

Definition 3.2 [Context (**Ctx**)] A *context* ς associates to each sort in Σ a non-empty subset of its processes: $\forall a \in \Sigma, \varsigma[a] \subseteq \mathcal{L}_a \wedge \varsigma[a] \neq \emptyset$. **Ctx** is the set of all contexts.

Definition 3.3 [$\Cap : \mathbf{Ctx} \times \wp(\mathbf{Proc}) \to \mathbf{Ctx}$] For any $\varsigma \in \mathbf{Ctx}$ and set of processes $ps \in \wp(\mathbf{Proc})$, the override of ς by ps is noted $\varsigma \Cap ps$ and is defined by:

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

For a given context ς, we note $a_i \in \varsigma$ if and only if $a_i \in \varsigma[a]$, and for all $ps \in \wp(\mathbf{Proc})$ or $ps \in \mathcal{L}$, $ps \subseteq \varsigma \overset{\Delta}{\Leftrightarrow} \forall a_i \in ps, a_i \in \varsigma$. A sequence of actions δ is *playable* in a context ς if and only if $\exists s \subseteq \varsigma, \delta \in \mathbf{Sce}(s)$. We denote then: $\delta \in \mathbf{Sce}(\varsigma)$, and the play of δ in ς is $\varsigma \cdot \delta = \varsigma \Cap \mathsf{end}(\delta)$, where $\mathsf{end}(\delta)$ is the set containing the last process in the sequence δ (hitter or bounce) of every sort mentioned in δ.

Finally, a bounce sequence on a sort a (Def. 3.4) is a sequence of actions hitting a in which the bounce process of each action is the hitter process of the following action. Bounce sequences are used to find local solutions to a given objective. A bounce sequence on a can be abstracted into sets of all its hitters that are not in sort a (Def. 3.5). This abstraction allows to propagate an objective on the sort a into objectives on other sots. In the following, we denote: $\mathbf{Sol} = \wp(\mathbf{Proc})$.

Definition 3.4 [Bounce sequence (**BS**)] A *bounce sequence* ζ is a sequence of actions so that $\forall n \in \mathbb{I}^\zeta, n < |\zeta|, \mathsf{bounce}(\zeta_n) = \mathsf{target}(\zeta_{n+1})$. **BS** denotes the set of

all bounce sequences, and $\mathbf{BS}(P)$ denotes the set of bounce sequences *solving* an objective P:

$$\mathbf{BS}(a_i \Rsh^* a_j) = \{\zeta \in \mathbf{BS} \mid \mathsf{target}(\zeta_1) = a_i \wedge \mathsf{bounce}(\zeta_{|\zeta|}) = a_j\} \ .$$

$\mathbf{BS}(a_i \Rsh^* a_j) = \emptyset$ if there is no way to reach a_j from a_i and $\varepsilon \in \mathbf{BS}(a_i \Rsh^* a_i)$.

Definition 3.5 $[\mathbf{BS}^\wedge : \mathbf{Obj} \to \wp(\mathbf{Sol})]$ The *abstractions of bounce sequences* of an objective P, denoted by the set $\mathbf{BS}^\wedge(P)$, are the sets of hitters of bounce sequences solving P:

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

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

3.2 Under-approximation

We denote $\gamma_\varsigma(\omega)$ the set of scenarios concretising an objective sequence ω in the context ς. In Def. 3.6, we define $\ell_\varsigma(\omega)$ as equal to $\gamma_\varsigma(\omega)$ if and only if $\gamma_\varsigma(\omega)$ contains scenarios starting from all states $s \subseteq \varsigma$. Theorem 3.7 is used to over-approximate the initial context ς.

Definition 3.6 $[\ell_\varsigma : \mathbf{OS} \to \wp(\mathbf{Sce})]$

$$\ell_\varsigma(\omega) = \begin{cases} \gamma_\varsigma(\omega) & \text{if } \forall s \in \mathcal{L}, s \subseteq \varsigma, \exists \delta \in \gamma_\varsigma(\omega), \delta \in \mathbf{Sce}(s) \\ \emptyset & \text{else.} \end{cases}$$

Theorem 3.7 $\varsigma \subseteq \varsigma' \wedge \ell_{\varsigma'}(\omega) \neq \emptyset \Longrightarrow \ell_\varsigma(\omega) \neq \emptyset$.

For any objective P and context ς, Def. 3.8 gives the set of processes of sort $\Sigma(P)$ that are required to solve P in ς, given by $\mathsf{maxCont}_\varsigma(\Sigma(P), P)$.

Definition 3.8 $[\mathsf{maxCont}_\varsigma : \Sigma \times \mathbf{Obj} \to \wp(\mathbf{Proc})]$

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

The graph of local causality $\lceil \mathcal{B}^\omega_\varsigma \rceil = (V, E)$ defined in Def. 3.9 is a graph where $V \subseteq \mathbf{Proc} \cup \mathbf{Obj} \cup \mathbf{Sol}$ and $E \subseteq V \times V$. A node in **Proc** is a required process, a node in **Obj** is an objective to reach a given process and a node in **Sol** is a set of processes required for the solving. An objective $P \in \mathbf{Obj}$ is solvable if the abstractions of bounce sequences $\mathbf{BS}^\wedge(P) \in \mathbf{Sol}$ (Def. 3.5) can be reached (Eq. (4)), thus leading to several required processes (Eq. (5)). If $a \in \Gamma$, the reachability of one of its process a_i is approximated by the ability to solve all objectives $a_j \Rsh^* a_i \in \mathbf{Obj}$ for all a_j in the initial context (Eq. (6)); if $a \in \Delta$, the reachability of a_i is simply solved by the set of processes $\mathsf{procState}(a_i)$ (Def. 2.14) that it represents (Eq. (7)). The solving of an objective P may require a process of $\Sigma(P)$, i.e. $\mathsf{maxCont}(\Sigma(P), P) \neq \emptyset$ (Def. 3.8); in this case, P is re-targeted (Eq. (8)). Eq. (1), (2) and (3) ensure that all required

nodes are in V_ς^ω. Finally, as the active process of every sort may evolve, $\lceil \mathcal{B}_\varsigma^\omega \rceil$ is obtained by iteratively saturating with every process it contains, i.e. by overriding its initial context ς by $\mathsf{procs}(V, E)$, defined by:

$$\mathsf{procs}(V, E) = (V \cap \mathbf{Proc}) \cup \{\mathsf{target}(P), \mathsf{bounce}(P) \mid P \in V \cap \mathbf{Obj}\}$$

Definition 3.9 The graph of local causality $\lceil \mathcal{B}_\varsigma^\omega \rceil = (\lceil V_\varsigma^\omega \rceil, \lceil E_\varsigma^\omega \rceil)$ is defined as: $\lceil \mathcal{B}_\varsigma^\omega \rceil = \mathbf{lfp}\{\mathcal{B}_\varsigma^\omega\}\left(\mathcal{B}_\varsigma^\omega \mapsto \mathcal{B}^\omega_{\varsigma \Cup \mathsf{procs}(\mathcal{B}_\varsigma^\omega)}\right)$, where $\mathcal{B}_\varsigma^\omega = (V_\varsigma^\omega, E_\varsigma^\omega)$ is the smallest graph with $V_\varsigma^\omega \subseteq \mathbf{Proc} \cup \mathbf{Obj} \cup \mathbf{Sol}$ and $E_\varsigma^\omega \subseteq V_\varsigma^\omega \times V_\varsigma^\omega$ so that:

$$\omega \subseteq V_\varsigma^\omega \tag{1}$$
$$P \in V \cap \mathbf{Proc} \Rightarrow \mathsf{bounce}(P) \in V_\varsigma^\omega \tag{2}$$
$$(x, y) \in E \Rightarrow y \in V_\varsigma^\omega \tag{3}$$
$$P \in V \cap \mathbf{Obj} \wedge ps \in \mathbf{BS}(P) \Rightarrow (P, ps) \in E_\varsigma^\omega \tag{4}$$
$$ps \in V \cap \mathbf{Sol} \wedge a_i \in ps \Rightarrow (ps, a_i) \in E_\varsigma^\omega \tag{5}$$
$$a \in \Gamma \wedge a_i \in V \cap \mathbf{Proc} \wedge a_j \in \varsigma \Rightarrow (a_i, a_j \Rsh^* a_i) \in E_\varsigma^\omega \tag{6}$$
$$a \in \Delta \wedge a_i \in V \cap \mathbf{Proc} \wedge ps \in \mathsf{procState}(a_i) \Rightarrow (a_i, ps) \in E_\varsigma^\omega \tag{7}$$
$$P \in V \cap \mathbf{Obj} \wedge q \in \mathsf{maxCont}_\varsigma(\Sigma(P), P) \Rightarrow (P, q \Rsh^* \mathsf{bounce}(P)) \in E_\varsigma^\omega \tag{8}$$

In the graph of local causality, an edge $(p, ps) \in \mathbf{Proc} \times \mathbf{Sol}$ is said coherent (Def. 3.10) if none of the processes in ps conflict with the children processes of ps. Then, Theorem 3.11 gives a sufficient condition for the concretization of a sequence of objectives in a given context, which is derived immediately from the graph of local causality. A proof of this theorem is given in Annex A.

Definition 3.10 [Coherent edge] An edge $(x, y) \in E_\varsigma^\omega$ is said coherent if and only if: $(x, y) \in \lceil E_\varsigma^\omega \rceil \cap (\mathbf{Proc} \times \mathbf{Sol}) \Rightarrow y$ has no children process $a_j \in \lceil V_\varsigma^\omega \rceil \cap \mathbf{Proc}$ so that $\exists a_i \in y,\ a_i \neq a_j$.

Theorem 3.11 (Under-Approximation) *If the graph $\lceil \mathcal{B}_\varsigma^\omega \rceil$ contains no cycle, all objectives have at least one solution and all edges are coherent, then $\ell_\varsigma(\omega) \neq \emptyset$.*

Computing the graph of local causality is polynomial in the number of sorts in $\mathcal{PH}$ and exponential in the number of processes in one sort. Checking the properties allowing to apply Theorem 3.11 is polynomial in the size of the graph. Furthermore, it is possible to compute only a subset of $V \cap \mathbf{Sol}$; in this case, the overall method turns out to be exponential in the number of solutions to each objective. Our method can thus be considered as efficient compared to regular model-checking which is usually PSPACE-complete [6].

Example 3.12 Let $\mathcal{PH}' = (\Sigma, \mathcal{L}, \mathcal{H}'^{\langle 1 \rangle})$ be the "flattened" version of the PH in Fig. 1, that is: $\mathcal{H}'^{\langle 1 \rangle} = \mathcal{H}^{(1)} \cup \mathcal{H}^{(2)}$, which is equivalent to a PH in the semantics without priorities. Due to spurious behaviours inherent to the cooperative sorts in this semantics, the original under-approximation developed in [11] concludes that c_1 is reachable in $\mathcal{PH}'$ from $\varsigma = \langle a_1, b_0, c_0, ab_{10} \rangle$.

Such unwanted behaviours are palliated by the semantics of PH with priorities proposed in this paper. Indeed, the under-approximation given in Theorem 3.11 does not conclude regarding the reachability of c_1, as one of the edges of the resulting graph of local causality is not coherent (Def. 3.10), as shown in in Fig. 2. (However, from the inconclusiveness of Theorem 3.11, one cannot conclude about the unreachability of c_1. Such analysis should be driven for instance with over-approximation methods developed in [11].)

However, if $a_0 \to b_0 \nearrow b_1$ and $b_0 \to a_0 \nearrow a_1$ are replaced by $a_0 \to a_0 \nearrow a_1$ and $b_0 \to b_0 \nearrow b_1$, then Theorem 3.11 concludes that c_1 is reachable from ς.

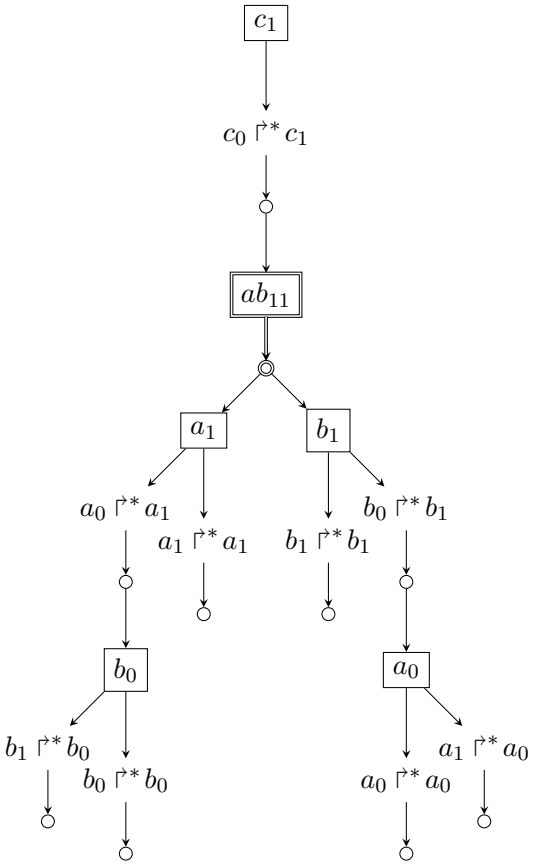

Figure 2. The graph of local causality of the PH model in Fig. 1. Rectangular nodes containing a single process are elements in **Proc**, nodes containing a couple of processes are elements in **Obj** and circle nodes are elements in **Sol**. Theorem 3.11 is inconclusive on this example as edge $(ab_{11}, \{a_1, b_1\}) \in \mathbf{Proc} \times \mathbf{Sol}$ (here represented with a double line) is not coherent (Def. 3.10). Indeed, $a_0 \in \mathbf{Proc}$ is a child of $\{a_1, b_1\}$, but $a_0 \neq a_1$ (and the same also stands for b_0).

3.3 Reachability of a state

The semantics of PH with 2 classes of priorities studied in this paper allows to model cooperative sorts accurately representing a coherent configuration of a set of sorts. Therefore, we can derive a new method to conclude about the reachability of a state (considering only components). Indeed, let $\mathcal{PH} = (\Sigma, \mathcal{L}, (\mathcal{H}^{(1)}, \mathcal{H}^{(2)}))$ be

a PH and suppose that we want to study the reachability of a state $s \in \mathcal{L}$. Let $\mathcal{PH}' = (\Sigma', \mathcal{L}', (\mathcal{H}'^{(1)}, \mathcal{H}'^{(2)}))$ with: $\Sigma' = \Sigma \cup \{\tau, \sigma\}$ and $\mathcal{L}' = \mathcal{L} \times \mathcal{L}_\tau \times \mathcal{L}_\sigma$, where τ is a cooperative sort on all components Γ of $\mathcal{PH}$ (thus $\mathcal{L}_\tau = \underset{a \in \Gamma}{\times} \mathcal{L}_a$) and σ is a component with $\mathcal{L}_\sigma = \{\sigma_0, \sigma_1\}$; furthermore, $\mathcal{H}'^{(1)}$ is the set $\mathcal{H}^{(1)}$ completed with all actions updating the cooperative sort τ, and $\mathcal{H}'^{(2)} = \mathcal{H}^{(2)} \cup \{\mathsf{lst}_s(\tau) \to \sigma_0 \Rsh \sigma_1\}$.

Given an initial context ς, the reachability of s in $\mathcal{PH}$ is equivalent to the concretization of $\sigma_0 \Rsh^* \sigma_1$ in $\mathcal{PH}'$ from the initial context $\varsigma \cup \{\sigma_0\}$ (the initial state of τ does not matter), which can be efficiently under-approximated using Theorem 3.11. Indeed, the additional action $\mathsf{lst}_s(\tau) \to \sigma_0 \Rsh \sigma_1$ in $\mathcal{H}'^{(2)}$ allows to conclude on the reachability of process $\mathsf{lst}_s(\tau)$, that is, on the reachability of the state s (considering only the components).

It is also possible to compute the reachability of a set of states $S \subseteq \mathcal{L}$ by creating several actions $\tau_s \to \sigma_0 \Rsh \sigma_1$ in $\mathcal{H}^{(2)}$ for each state $s \in S$.

4 Large-scale Biological Example

In order to support the scalability and applicability of our under-approximation of reachability, we apply our new approach for the analysis of large-scale model of the T-cell receptor (TCR) signalling pathway [12]. This model gathers 94 interacting components and is specified as a Boolean network. The under-approximation presented in this paper has been implemented in the existing Pint software [2].

The Boolean model has been automatically encoded into a Process Hitting with 2 classes of priority [3]. Then, we verified the reachability for the independent activation of 4 outputs of the signalling cascade (SRE, AP1, NFkB, NFAT) from all possible input combinations (CD45, CD28, TCRlib) using our new reachability under-approximation (answering either *yes* or *inconclusive*) and a previously defined reachability over-approximation [11] (answering either *no* or *inconclusive*). All result in conclusive decisions, and the under-approximation has been satisfied in 12 cases (over 32) proving the satisfiability of the concerned reachability property in the encoded Boolean network (and non-satisfiability in the other cases).

Computations times are in the order of a few hundredths of a second on a 2.4GHz processor with 2GB of RAM. To give a comparison, we did the same experiments with a standard symbolic model-checker, libDDD [9], known for its good performances, the input model being the Boolean network expressed as a Petri net. However, due to the large scale of the model, the program runs out of memory for all the experiments.

While ensuring a low complexity for the analysis of reachability in Boolean and discrete networks, our under-approximation method reveals to be conclusive in numerous cases when applied to real large-scale biological models, which were not tractable with exact model-checking.

---

[2] Pint is freely available at http://process.hitting.free.fr.

[3] Model and scripts are available at http://www.irccyn.ec-nantes.fr/~folschet/underapprox-tcrsig94.zip.

5 Discussion & Conclusion

We introduced a new semantics to include priorities into the Process Hitting framework, which prove especially useful to model cooperations. Then, we developed a method to efficiently perform a reachability analysis of a sequence of objectives in a restricted class of Process Hitting models, but it is also useful to establish the reachability of a partial state. This analysis is based on an under-approximation of the true reachability solutions.

We showed that the class of Process Hitting models that can be handled by the aforementioned method are equivalent to Asynchronous Discrete Networks, and therefore to Asynchronous Boolean Networks. This allows to efficiently compute reachability results on large biological models provided that they are equivalent to Asynchronous Discrete Networks and that a translation from the original framework into a Process Hitting model is possible. Such a translation for interaction graphs of Thomas modelling was proposed in [10].

Further work can be derived from what have been presented in this paper. The over-approximation on Process Hitting models without priorities proposed in [11] is still accurate in the framework with priorities (by "flattening" all actions), but may be refined given the restrictions proposed in this paper, and a specific search of key processes or cut sets may be derived. Furthermore, a more general under-approximation could be developed in order to handle a larger class of Process Hitting models, that is, models with more than two classes of priorities, that do not only contain components of cooperative sorts, or whose behaviour may contain cycles or cyclic attractors. Finally, in order to take into account quantitative data in transition delays, the overall approximation method could be extended to handle evolutions that are chronometric instead of only chronologic.

References

[1] Bause, F., *Analysis of Petri nets with a dynamic priority method*, in: P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, Lecture Notes in Computer Science **1248**, Springer Berlin Heidelberg, 1997 pp. 215–234.

[2] Cleaveland, R. and M. Hennessy, *Priorities in process algebras*, Information and Computation **87** (1990), pp. 58–77, special Issue: Selections from 1988 IEEE Symposium on Logic in Computer Science.

[3] Cleaveland, R., G. Lüttgen and V. Natarajan, *Priority and abstraction in process algebra*, Information and Computation **205** (2007), pp. 1426–1458.

[4] Cleaveland, R., G. Lüttgen and V. Natarajan, *Priorities in process algebra* (1999).

[5] De Jong, H., *Modeling and simulation of genetic regulatory systems: a literature review*, Journal of computational biology **9** (2002), pp. 67–103.

[6] Harel, D., O. Kupferman and M. Y. Vardi, *On the complexity of verifying concurrent transition systems*, Information and Computation **173** (2002), pp. 143–161.

[7] John, M., C. Lhoussaine, J. Niehren and A. Uhrmacher, *The attributed pi-calculus with priorities*, in: C. Priami, R. Breitling, D. Gilbert, M. Heiner and A. Uhrmacher, editors, *Transactions on Computational Systems Biology XII*, Lecture Notes in Computer Science **5945**, Springer Berlin Heidelberg, 2010 pp. 13–76.

[8] Kauffman, S. A., *Metabolic stability and epigenesis in randomly constructed genetic nets*, Journal of theoretical biology **22** (1969), pp. 437–467.

[9] LIP6/Move, *the libDDD environment* (libDDD), http://ddd.lip6.fr.

[10] Paulevé, L., M. Magnin and O. Roux, *Refining dynamics of gene regulatory networks in a stochastic π-calculus framework*, in: *Transactions on Computational Systems Biology XIII*, Springer, 2011 pp. 171–191.

[11] Paulevé, L., M. Magnin and O. Roux, *Static analysis of biological regulatory networks dynamics using abstract interpretation*, Mathematical Structures in Computer Science **22** (2012), pp. 651–685.

[12] Saez-Rodriguez, J., L. Simeoni, J. A. Lindquist, R. Hemenway, U. Bommhardt, B. Arndt, U.-U. Haus, R. Weismantel, E. D. Gilles, S. Klamt and B. Schraven, *A logical model provides insights into t cell receptor signaling*, PLoS Comput Biol **3** (2007), p. e163.

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

[14] Wagler, A. and J.-T. Wegener, *On minimality and equivalence of Petri nets*, in: L. Popova-Zeugmann, editor, *CS&P*, CEUR Workshop Proceedings **928** (2012), pp. 382–393.

[15] Wagler, A. and R. Weismantel, *The combinatorics of modeling and analyzing biological systems*, Natural Computing **10** (2011), pp. 655–681.

A Proof of Under-approximation (Theorem 3.11)

In the following, we denote: $\lceil E_\varsigma^\omega \rceil_Y^X = \lceil E_\varsigma^\omega \rceil \cap (X \times Y)$, with X, Y amongst **Proc**, **Obj** and **Sol**.

Proof We note $max\varsigma = \varsigma \Cap \mathsf{procs}(\lceil \mathcal{B}_\varsigma^\omega \rceil)$ the context supported by $\lceil \mathcal{B}_\varsigma^\omega \rceil$.

Let $(a_i, ps) \in \lceil E_\varsigma^\omega \rceil_{\mathbf{Sol}}^{\mathbf{Proc}}$ be an edge linking the required process of a cooperative sort to a solution set and suppose all children of ps are concretisable. We label all processes of ps by an integer: $ps = \{p_n\}_{n \in \mathbb{I}^{ps}}$. Let us prove by induction that for all $n \in \mathbb{I}^{ps}$, there exists a scenario δ_n so that: $\forall i \in [\![1; n]\!], (s \cdot \delta_n)[\Sigma(p_i)] = p_i$.

- It is straightforward for $\delta_0 = \varepsilon$.
- Suppose such δ_n exists and let $q = (s \cdot \delta_n)[\Sigma(p_{n+1})]$. By hypothesis, (a_i, ps) is coherent (Def. 3.10) and all processes of ps are processes of components; this means that none of the processes needed to solve p_{n+1} is another process of the same sort than another process of ps. Therefore, there exists $\delta' \in \ell_{s \cdot \delta_n}(q \upharpoonright^* p_{n+1})$, so that $\forall i \in [\![1; n+1]\!]$, $(s \cdot \delta_n \cdot \delta')[\Sigma(p_i)] = p_i$. Finally, by Theorem 2.18, there exists a scenario $\delta'' \in \mathbf{Sce}_{|1}(s \cdot \delta_n \cdot \delta')$ so that, if we denote $\delta_{n+1} = \delta_n \cdot \delta' \cdot \delta''$, we have: $\mathsf{update}(s \cdot \delta_n \cdot \delta') = s \cdot \delta_{n+1}$ and the same property about processes (by Theorem 2.20).

Therefore, $\delta = \delta_{|ps|}$ exists, and given its properties, we have: $(s \cdot \delta)[a] = a_i$ and $\mathsf{update}(s \cdot \delta) = s \cdot \delta$.

As there is no cycle in $\lceil \mathcal{B}_\varsigma^\omega \rceil$, we show by induction that $\forall s \in L, s \subseteq max\varsigma$, for all objective P in $\lceil V_\varsigma^\omega \rceil \cap \mathbf{Obj}$ so that $\mathsf{target}(P) \in s$, $\exists \delta \in \ell_s(P)$.

- If $(P, \emptyset) \in \lceil E_\varsigma^\omega \rceil_{\mathbf{Sol}}^{\mathbf{Obj}}$, either $\mathsf{target}(P) = \mathsf{bounce}(P)$ and $\delta = \varepsilon$; or $\forall \zeta \in \mathbf{BS}(P), \zeta \in \mathbf{Sce}(s) \wedge \Sigma(\zeta) = \{\Sigma(P)\}$ and $\delta = \delta_1 \cdot \zeta_1 \cdot \ldots \cdot \delta_{|\zeta|} \cdot \zeta_{|\zeta|}$ is a valid sequence given by Theorem 2.19.
- Suppose all children objectives of P are concretizable. If $\exists (P, Q) \in \lceil E_\varsigma^\omega \rceil_{\mathbf{Obj}}^{\mathbf{Obj}}$, then by hypothesis, $\ell_s(\mathsf{target}(P) \upharpoonright^* \mathsf{target}(Q) :: Q) \neq \emptyset$, thus $\ell_s(P) \neq \emptyset$. Else, by Def. 3.8, the concretizations of the children of P require no process of sort

$\Sigma(P)$. Furthermore, there exists $\zeta \in \mathbf{BS}(P)$ so that $(P, \zeta^\wedge) \in \lceil E^\omega_\zeta \rceil^{\mathbf{Obj}}_{\mathbf{Sol}}$. We show by induction that for all $n \in \mathbb{I}^\zeta$, there is a scenario δ_n so that $(s \cdot \delta_n)[\Sigma(P)] = \mathsf{bounce}(\zeta_n)$.

- Suppose that δ_n exists and let $\zeta_n = b_i \to a_j \not\vdash a_k$. By hypothesis there exists $\delta' \in \ell_{s\cdot\delta_n}(\star \not\vdash^* b_i)$ with $\Sigma(P) \notin \Sigma(\delta')$ (by Def. 3.8). By Theorem 2.18 there exists $\delta'' \in \mathbf{Sce}|_1(s\cdot\delta')$ so that $\mathsf{update}(s\cdot\delta') = s\cdot\delta'\cdot\delta''$. Furthermore, $(s \cdot \delta' \cdot \delta'')[b] = b_j$ (by Theorem 2.19 if $b \in \Gamma$ or Theorem 2.20 if $b \in \Delta$). Therefore, $\delta_{n+1} = \delta_n \cdot \delta' \cdot \delta'' \cdot \zeta_n$.

Thus, $\delta_{|\zeta|} \in \ell_s(P)$.

Finally, as $\ell_{max_\varsigma}(\omega) \neq \emptyset$, $\ell_\varsigma(\omega) \neq \emptyset$ (Theorem 3.7). □

B Weak Bisimulation of Asynchronous Discrete Networks (Subsect. 2.2)

We exhibit an encoding of Asynchronous Discrete Networks (ADN) with the Process Hitting using two classes of priorities, and prove a weak bisimulation relation.

A Discrete Network gathers a finite number of components $i \in [\![1; n]\!]$ having a discrete finite domain $\mathbb{F}^i$ that we note $\mathbb{F}^i = [\![1; l_i]\!]$. For each component $i \in [\![1; n]\!]$, a map $\mathbb{F} \to \mathbb{F}^i$ is defined, where $\mathbb{F} = \mathbb{F}^1 \times \cdots \times \mathbb{F}^n$, giving the next value of the component with respect to the global state of the network. Typically f^i depends on a subset of components that we denote $\mathsf{dep}(f^i)$. In the case of Asynchronous Discrete Networks (ADN), a transition relation $\to_{ADN} \subseteq \mathbb{F} \times \mathbb{F}$ is defined such that $x \to_{ADN} x'$ if and only if there exists a unique $i \in [\![1; n]\!]$ such that $x'[i] = f^i(x)$ and $\forall j \in [\![1; n]\!], j \neq i, x'[j] = x[j]$, i.e. one and only one component has been updated. This is formalised in Def. B.1.

Definition B.1 [Asynchronous Discrete Network (ADN)] An ADN is defined by a couple $(\mathbb{F}, \langle f^1, \ldots, f^n\rangle)$ where $\mathbb{F} = \mathbb{F}^1 \times \cdots \times \mathbb{F}^n$, and $\forall i \in [\![1; n]\!]$, $f^i : \mathbb{F} \to \mathbb{F}^i$ with $\mathbb{F}^i = [\![1; l_i]\!]$. Given two states $x, x' \in \mathbb{F}$, the transition relation $\to_{ADN}$ is given by

$$x \to_{ADN} x' \iff \exists i \in [\![1; n]\!], f^i(x) = x'[i] \wedge \forall j \in [\![1; n]\!], j \neq i, x[j] = x'[j] \ ,$$

where $x[i]$ is the i-th component of x. We note $\mathsf{dep}(f^i) \subseteq [\![1; n]\!]$ the set of components on which the value of f^i depends: $\forall x, x' \in \mathbb{F}$ such that $\forall j \in \mathsf{dep}(f^i), x[j] = x'[j]$, necessarily $f^i(x) = f^i(x')$.

Let us denote the encoding of the ADN $(\mathbb{F}, \langle f^1, \ldots, f^n\rangle)$ in Process Hitting with 2 classes of priorities by $\mathbf{PH}(\mathbb{F}, \langle f^1, \ldots, f^n\rangle)$ (Def. B.2). For each component $i \in [\![1; n]\!]$ of the ADN, two sorts are built: a^i acting for the component value, and f^i acting for a cooperative sort between the components $\mathsf{dep}(f^i)$. Sorts a^i have one process a^i_k per element in $k \in \mathbb{F}^i$. Sorts f^i have one process f^i_ς per state $\varsigma \in \bigtimes_{j\in\mathsf{dep}(f^i)} \mathcal{L}_{a^j}$. Two classes of actions are then defined: $\mathcal{H}^{(1)}$ is the set of actions updating the cooperative sorts according to the current state of the components: if $j \in \mathsf{dep}(f^i)$, a^j_k hits each process f^i_ς where $\varsigma[a^j] \neq a^j_k$ to make it bounce to the process $f^i_{\varsigma \uplus a^j_k}$. $\mathcal{H}^{(2)}$ is the set

of actions encoding the transitions in the ADN: f^i_ς hits the processes of sort a^i to make them bounce to the process $a^i_{k'}$ if and only if $k' = f^i([\![\varsigma]\!])$; $[\![\varsigma]\!]$ being the ADN state corresponding to the PH (partial) state ς (note that $f^i([\![\varsigma]\!])$ is fully defined because $[\![\varsigma]\!]$ specifies the state for all the components in $\mathsf{dep}(f^i)$).

Definition B.2 $\mathbf{PH}(\mathbb{F}, \langle f^1, \dots, f^n \rangle) = (\Sigma, \mathcal{L}, (\mathcal{H}^{(1)}, \mathcal{H}^{(2)}))$ is the Process Hitting with 2 classes of priority encoding the ADN $(\mathbb{F}, \langle f^1, \dots, f^n \rangle)$, with:

- $\Sigma = \{a^1, \dots, a^n\} \cup \{f^1, \dots, f^n\}$, the sorts for components (a^i) and cooperative sorts (f^i);
- $\mathcal{L} = \bigtimes_{i \in [\![1;n]\!]} \mathcal{L}_{a^i} \times \bigtimes_{i \in [\![1;n]\!]} \mathcal{L}_{f^i}$, where $\mathcal{L}_{a^i} = \{a^i_0, \dots, a^i_{l_i}\}$, and $\mathcal{L}_{f^i} = \{f^i_\varsigma \mid \varsigma \in \bigtimes_{j \in \mathsf{dep}(f^i)} \mathcal{L}_{a^i}\}$ if $\mathsf{dep}(f^i) \neq \emptyset$, otherwise $\mathcal{L}_{f^i} = \{f^i_\emptyset\}$;
- $\mathcal{H}^{(1)} = \{a^j_k \to f^i_\varsigma \Rsh f^i_{\varsigma'} \mid i \in [\![1;n]\!] \wedge j \in \mathsf{dep}(f^i) \wedge a^j_k \in \mathcal{L}_{a^j} \wedge f^i_\varsigma \in \mathcal{L}_{f^i} \wedge \varsigma[a^j] \neq a^j_k \wedge \varsigma'[a^j] = a^j_k \wedge (\varsigma'[a^l] = \varsigma[a^l], \forall l \in [\![1;n]\!], l \neq j)\}$, the set of actions with priority 1 that update cooperative sorts;
- $\mathcal{H}^{(2)} = \{f^i_\varsigma \to a^i_k \Rsh a^i_{k'} \mid i \in [\![1;n]\!] \wedge f^i_\varsigma \in \mathcal{L}_{f^i} \wedge a^i_k \in \mathcal{L}_{a^i} \wedge k \neq k' \wedge f^i([\![\varsigma]\!]) = k'\}$, the set of actions with priority 2 for updating the components using their respective discrete maps. $[\![\varsigma]\!]$ is defined below.

Given a state $s \in \mathcal{L}$ of the Process Hitting, $[\![s]\!] = x$ is the corresponding state in the ADN: $\forall i \in [\![1;n]\!], s[a^i] = a^i_k \Rightarrow x[i] = k$.
Given a state $x \in \mathbb{F}$ of the ADN, $(\!|x|\!) = s$ is the corresponding state in the Process Hitting: $\forall i \in [\![1;n]\!], x[i] = k \Rightarrow s[a^i] = a^i_k$ and $\forall i \in [\![1;n]\!], s[f^i] = f^i_\varsigma$ with $f^i_\varsigma \in \mathcal{L}_{f^i}$ and $\forall j \in \mathsf{dep}(f^i), \varsigma[j] = s[a^j]$.

Theorem B.3 states the weak bisimulation relation between an ADN and its encoding in PH with 2 classes of priorities. Intuitively, actions updating cooperative sorts being prioritised, actions updating component sorts follow strictly the possible transitions of the ADN.

Theorem B.3 $((\mathbb{F}, \langle f^1, \dots, f^n \rangle) \approx \mathbf{PH}(\mathbb{F}, \langle f^1, \dots, f^n \rangle))$

(i) $\forall x, x' \in \mathbb{F}$, $x \to_{ADN} x' \Longrightarrow (\!|x|\!) \to^*_{PH} (\!|x'|\!)$, *where* $\to^*_{PH}$ *is a finite sequence of* $\to_{PH}$ *transitions.*

(ii) $\forall s, s' \in \mathcal{L}$, $s \to_{PH} s' \Longrightarrow [\![s]\!] = [\![s']\!] \vee [\![s]\!] \to_{ADN} [\![s']\!]$.

Proof (i) From Def. B.1, $x \to_{ADN} x' \Rightarrow \exists i \in [\![1;n]\!], f^i(x) = x'[i] \wedge \forall j \in [\![1;n]\!], i \neq j, x[j] = x'[j]$. Let us assume (without loss of generality) that $f^i(x) = k'$, $x[i] = k$ and $\varsigma \in \bigtimes_{j \in \mathsf{dep}(f^i)} \mathcal{L}_{a^j}$ such that $\forall j \in \mathsf{dep}(f^i), \varsigma[j] = a^j_{x[j]}$. From Def. B.2, $h = f^i_\varsigma \to a^i_k \Rsh a^i_{k'} \in \mathcal{H}^{(2)}$. From the definition of $(\!|x|\!)$, $a^i_k \in (\!|x|\!)$ and $f^i_\varsigma \in (\!|x|\!)$; moreover, as there is no action in $\mathcal{H}^{(1)}$ applicable in $(\!|x|\!)$, h is applicable in $(\!|x|\!)$: $(\!|x|\!) \to_{PH} (\!|x|\!) \cdot h$. In $(\!|x|\!) \cdot h$, the only applicable actions of priority 1 are those having $a^i_{k'}$ as hitter and hitting cooperative sorts, giving a finite number of transitions towards $(\!|x'|\!)$.

(ii) $s \to_{PH} s'$ only if there exists an action h applicable in s such that $s \cdot h = s'$. If $\mathsf{prio}(h) = 1$, then, by definition of $\mathcal{H}^{(1)}$, $[\![s]\!] = [\![s']\!]$. If $\mathsf{prio}(h) = 2$, then $\forall i \in [\![1;n]\!]$,

if $s[f^i] = f^i_\varsigma$, then, $\forall j \in \mathsf{dep}(f^i), \varsigma[a^j] = s[a^j]$. Let $i \in [\![1; n]\!]$ such that $s[a^i] \neq s'[a^i]$ (i is unique for this transition). By Def. B.2, if $s'[a^i] = a^i_{k'}$, necessarily $f^i([\![s]\!]) = k'$, hence $[\![s]\!] \to_{ADN} [\![s']\!]$. □