# Exhaustive analysis of dynamical properties through simulation of biological regulatory networks

Emna Ben Abdallah*, Maxime Folschette†, Olivier Roux* and Morgan Magnin‡
*LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597
(Institut de Recherche en Communications et Cybernétique de Nantes), 1 rue de la Noë, 44321 Nantes, France.
Email: emna.ben-abdallah@irccyn.ec-nantes.fr
olivier.roux@irccyn.ec-nantes.fr
†School of Electrical Engineering and Computer Science, University of Kassel, Germany
Email: maxime.folschette@uni-kassel.de
‡National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.
Email: morgan.magnin@irccyn.ec-nantes.fr

***Abstract*—The regulation of gene expression is achieved through genetic regulatory networks structured by systems of interactions between cells, DNA, RNA and proteins. As most genetic regulatory networks of interest involve many components connected through positive and negative reactions, an intuitive understanding of their dynamics is hard to obtain. As a consequence, formal methods and computer tools for the modeling and simulation of genetic regulatory networks are indispensable. The complexity of Biological Regulatory Networks (BRNs) often defies the intuition of the biologist and calls for the development of proper mathematical methods to model their structures and to delineate their dynamical properties. Our new formalism the Process Hitting (PH) defined as a restriction of synchronous automata networks is notably suitable. Also it is not limited, to model and analyze efficiently BRNs. In this paper, we propose a new logical approach to perform model-checking of dynamical properties of biological regulatory networks modeled in PH. Our work here focuses on state reachability properties on the one hand and on the identification of fixed points on the other hand. The originality of our model-checking approach relies in the exhaustive enumeration of all possible solutions of a simulation as well as of a dynamical property. The merits of our methods is illustrated by applying them to biological examples of various sizes and comparing the results with some existing approaches. It turns out that our approach succeeds in processing large models with a high number of components and interactions.**

## I. Introduction

The paper is motivated by the problem of steady states identification as well as reachability problem in biological regulatory systems that describes genes and proteins interactions. Indeed the regulatory phenomena play a crucial role in biological systems, so they need to be studied accurately. Thus the study of Biological Regulatory Networks (BRNs) has been the subject of numerous researches [**?**], [**?**], [**?**]. BRNs consist in sets of either positive or negative mutual effects between the components. With the purpose of analyzing these systems, they are often modeled as graphs which makes it possible to determine the possible evolutions of all the interacting components of the system. Thus, in order to address the formal checking of dynamical properties within very large BRNs, new formalisms and conversely new techniques have been proposed during the last decade. For example, Boolean networks [**?**], [**?**] have been used due to their simplicity and ability to handle noisy data. But this binary representation of genes in the network is a way to lose data information. Besides, artificial neural networks omit using a hidden layer so that they can be interpreted, losing the ability to model higher order correlations in the data. We can cite some works that are manipulating with these models and which have been developed to verify the dynamic properties of BRNs modeled: GINsim (Gene Interaction Network simulation) [**?**], [**?**] is devoted to the modelling and simulation of genetic regulatory networks, based on the logical approach. This tool is very convinient to define a logical regulatory graph through a dedicated interface. However the inconvinient is when the user starts to manipulate large-scale networks. Indeed, in order to verify a dynamical property, the reachability, it begins by computing the whole boolean network from the corresponding Thomas network.

We can also cite libDDD (Library of Data Decision Diagrams) a library for symbolic model-checking of CTL & LTL properties [**?**], [**?**]. So it can especially be used to check reachability properties. But it cannot output the execution path solving the reachability. The computing tests (see section Section V) show that libDDD and GINsim take a long time to respond, and gets out of memory for the big examples (more than 40 components). Several other promising modeling and analysing techniques have been developed, using Boolean networks, Petri nets [**?**], Bayesian networks [**?**] trying to optimise the computational time as well as the result.

It is also common to model biological networks with a set of coupled ordinary differential equations (ODEs) [**?**], describing the kinetic reactions. But the temporal evolution of the systems modeled by ODEs is computed by a complex derivation approach. This complexity of computing the evolution so that the verification of the dynamic properties causes also a long time to respond.

New experimental results [**?**] [**?**] demonstrated that gene expression is a stochastic process. Thus, many biologists and bioinformaticians are now using the stochastic formalism [**?**], [**?**], [**?**]. This formalism permits to represent each gene expression [**?**] and small synthetic genetic networks, [**?**] [**?**].

Such that our new formalism named Process Hitting (PH) [**?**]. In order to address the formal checking of dynamical properties within very large BRNs, we recently introduced this new formalism (PH) [**?**], to model concurrent systems having components with a few qualitative levels. A PH describes, in an atomic manner, the possible evolutions of a "process" (representing one component at one level) triggered by the hit of other "processes in the system. Comparing with other models of BRNs, this particular structure of PH makes the formal analysis of BRNs with hundreds of components easier to be tractable. This was proved by a first work on the PH in [**?**], [**?**] which analysis big networks and gives a response through a short time. But this developed technic was based on abstract methods computing approximations of the dynamics that could be inconclusive in some cases. Moreover, in the case of a positive answer, it currently does not return the execution of the path achieving the desired reachability, but only outputs its conclusion.

Our goal in this paper is to develop exhaustive methods to analyze Biological Regulatory Networks modeled in Process Hitting. With respect to PH dynamic, this analysis consists into:

- Simulating the evolution of the biological networks,
- Finding all possible steady states of BRNs in short time, and
- Computing the shortest execution path to reach a property.

The particularity of our contribution relies in the use of Answer Set Programming (ASP) [**?**] to compute the result of theis inference. This declarative programming framework has been proven efficient to tackle models with a large number of components and parameters. Our aim here is to assess its potential w.r.t. the computation of some dynamical properties of PH models.

The paper is organised as follows. II-A definition of the PH framework and the dynamical properties. We conclude the paper with a short discussion.

## II. Preliminary definitions

### A. Process Hitting

Definition 1 introduces the Process Hitting (PH) [**?**] which allows to model a finite number of local levels, called processes, grouped into a finite set of components, called sorts. A process is noted $a_i$, where $a$ is the sort's name, and $i$ is the process identifier within sort $a$. At any time, exactly one process of each sort is active, and the set of active processes is called a state.

The concurrent interactions between processes are defined by a set of actions. Each action is responsible for the replacement of one process by another of the same sort conditioned by the presence of at most one other process in the current state. An action is denoted by $a_i \to b_j \Rsh b_k$, which is read as "$a_i$ hits $b_j$ to make it bounce to $b_k$", where $a_i$, $b_j$, $b_k$ are processes of sorts $a$ and $b$, called respectively hitter, target and bounce of the action. We also call a self-hit any action whose hitter and target sorts are the same, that is, of the form: $a_i \to a_i \Rsh a_k$.

The PH is therefore a restriction of synchronous automata, where each transition changes the local state of exactly one automaton, and is triggered by the local states of at most two distinct automata. This restriction in the form of the actions was chosen to permit the development of efficient static analysis methods based on abstract interpretation [**?**] as well as the dynamic analysis that we present in this paper. In addition due to this restriction we need to add non biological components into the system that allow to present the cooperation between 2 or more components which we call cooperative sorts. So it maintains the effectiveness of the static and dynamic analysis.

*Definition 1 (Process Hitting):* A Process Hitting is a triple $(\Sigma, \mathcal{L}, \mathcal{H})$ where:

- $\Sigma = \{a, b, \dots\}$ is the finite set of sorts;
- $\mathcal{L} = \prod_{a \in \Sigma} \mathcal{L}_a$ is the 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 $l_a$ is a positive integer, with $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$;
- $\mathcal{H} = \{a_i \to b_j \Rsh b_k \in \mathcal{L}_a \times \mathcal{L}_b^2 \mid (a, b) \in \Sigma^2 \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ is the finite set of actions.

Figure 1 represents a PH $(\Sigma, \mathcal{L}, \mathcal{H})$ with four sorts

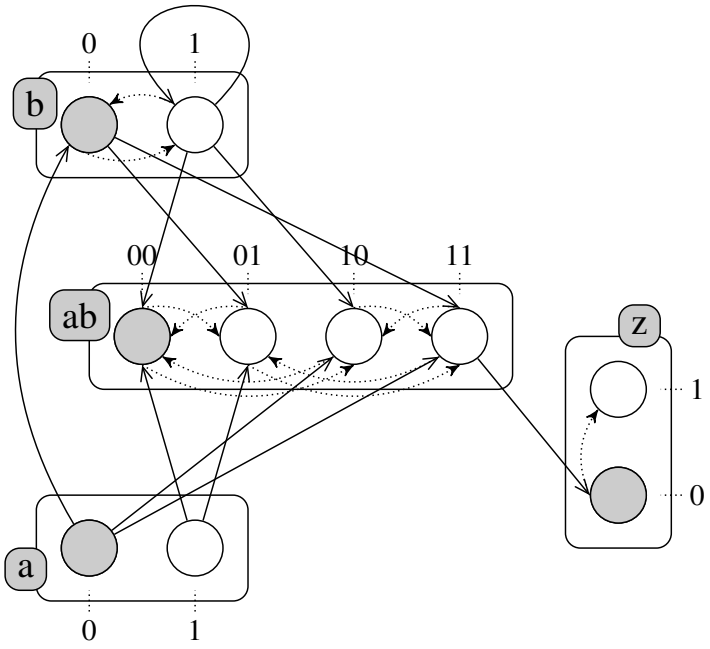


Figure 1. A PH model example with four sorts: $a$, $b$, $ab$ and $z$. Boxes represent the sorts (network components), circles represent the processes (component levels), and the actions that model the dynamic behavior are depicted by pairs of arrows in solid and dotted lines. $a$ is either at level 0 or 1, $b$ at either level 0 or 1, $z$ at either level 0, 1 or 2 and the cooperative sort $ab$ has 4 levels corresponding to the combinaison of the levels of the corresponding sorts $a$ and $b$. The grayed processes stand for the possible initial state $\langle a_0, b_0, ab_{00}, z_0 \rangle$.

*Example 1:*

A state of the networks is a set of active processes containing a single process of each sort. The active process of a given sort $a \in \Sigma$ in a state $s \in \mathcal{L}$ is noted $s[a]$. For any given process $a_i$ we also note: $a_i \in s$ if and only if $s[a] = a_i$. It means that the component, or the sort, $a$ is at level $i$ during the state $s$.

*Definition 2 (Playable action):* Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ be a Process Hitting and $s \in \mathcal{L}$ a state of $PH$. We say that the action $h = a_i \to b_j \Rsh b_k \in \mathcal{H}$ is playable in state $s$ if and only if $a_i \in s$ and $b_j \in s$ (i.e., $s[a] = a_i$ and $s[b] = b_j$). The resulting state after playing $h$ in $s$ is called a successor of $s$ and is denoted by $(s \cdot h)$, where $(s \cdot h)[b] = b_k$ and $\forall c \in \Sigma, c \neq b \Rightarrow (s \cdot h)[c] = s[c]$.

PH was chosen for several reasons. It is a general framework that was mainly used for biological networks. Besides, it allows to represent any kind of dynamical model, and converters to several other representations are available and included into PINT[1] [**?**]. Espacially for the particular form of the actions in a PH model allows to easily represent them using the Answer Set Programming [**?**], [**?**], [**?**], [**?**], with one fact per action. We note that although an efficient dynamical analysis already exists for this framework, based on an approximation of the dynamics, it is interesting to identify its limits and compare them to the exhaustive approach we present in this paper.

### B. Dynamical properties

The study of the dynamics of biological networks was the focus of many works, explaining the diversity of network modelings and the different methods developed in order to check dynamic properties. In this paper we focus on 2 main properties: the stable states and the reachability. In the following, we consider a PH model $(\Sigma, \mathcal{L}, \mathcal{H})$, and we formally define these properties and explain how they could be verified on such a network.

The notion of fixed point, also called stable state, is given in Definition 3. A fixed point is a state which has no successor. Such states have a particular interest as they denote states in which the model stays indefinitely, and the existence of several of these states denotes a switch in the dynamics [**?**].

*Definition 3 (Fixed point):* A state $s \in \mathcal{L}$ is called a fixed point (or equivalently stable state) if and only if it has no successors. In other words, and in $PH$ symantic, $s$ is a fixed point if and only if no action is playable in this state:

$$\forall a_i \to b_j \Rsh b_k \in \mathcal{H}, a_i \notin s \vee b_j \notin s \ .$$

A finer and more interesting dynamical property consists in the notion of reachability. Such a property, defined in Definition 4, states that starting from a given initial state, it is possible to reach a given goal, that is, a state that contains a process or a set of processes. Checking such a dynamical property is considered difficult as in usual model-checking techniques, it is required to build (a part of) the state graph, which has an exponential complexity.

In the following, if $s \in \mathcal{L}$ is a state, we call scenario in $s$ any sequence of actions that are successively playable in $s$. We also note $\mathbf{Sce}(s)$ the set of all scenarios in $s$. Moreover, we denote by $\mathbf{Proc} = \bigcup_{a \in \Sigma} \mathcal{L}_a$ the set of all process in $\mathcal{PH}$.

*Definition 4 (Reachability property):* If $s \in \mathcal{L}$ is a state and $A \subseteq \mathbf{Proc}$ is a set of processes, we denote by $\mathcal{P}(s, A)$ the following reachability property:

$$\mathcal{P}(s, A) \equiv \exists \delta \in \mathbf{Sce}(s), \forall a_i \in A, (s \cdot \delta)[a] = a_i \ .$$

The rest of the paper focuses on the resolution of the previous issues. We demonstrate our algorithms about: the simulation of a biological regulatory network modeled in PH, the enumeration of all fixed points (Section III and the verification of the reachability property will be handled in XXXXX (Section IV).

## III. Fixed point enumeration

The study of fixed points (and, more generally, basins of attraction) provides an important understanding of the different behaviors of a Biological Regulatory Network (BRN) [**?**]. Indeed, a system will always eventually end in a basin of attraction, and this may depend on biological switch or other complex phenomena. A fixed point is a state of the BRN in which it is not possible to have any new dynamic evolutions anymore; in other words, it is a basin of attraction that is composed of only one state.

In the following, we consider a Process Hitting $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$. A state $s \in L$ is a fixed point (or stable state) of the Process Hitting model if and only if it has no successor state, i.e., regarding the active processes in $s$ there is no playable action. Therefore, it has been shown [**?**] that a stable state in a Process Hitting network is a state so that every active process does not hit or is not hit by another process in the same state. We note that given this result, processes involved in a self-hit (an action whose hitter and target processes are the same) cannot be part of a stable state.

### A. Process Hitting translation in ASP

Before analyzing the dynamics of the network, we first need to translate the concerned PH network into ASP [2]. To do this we use the following self-describing predicates: `sort` to define sorts, `process` for the processes and `action` for the network actions. Example 2 shows how a PH network is defined with these predicates.

*Example 2 (Representation of a PH network in ASP):* The representation of the PH network of Figure 1 in ASP is the following:

```
process("a", 0..1). process("b", 0..1).
process("z", 0..1). process("ab", 0..3).
sort(X) ← process(X,_).
action("a",0,"b",0,1). action("b",1,"b",1,0).
action("b",0,"ab",1,0). action("b",0,"ab",3,2).
action("b",1,"ab",0,1). action("b",0,"ab",2,3).
action("a",0,"ab",2,0). action("a",0,"ab",2,1).
action("a",1,"ab",0,2). action("a",1,"ab",1,3).
action("ab",3,"z",0,1).
```

[1] PINT version 2015-11-14: http://loicpauleve.name/pint/

[2] All programs, including this translation and the methods described in the following, are available online at: https://github.com/EmnaBenAbdallah/verification-of-dynamical-properties_PH

In lines 1-2 we create the list of processes (expression level) corresponding to each sort, for example the sort `"a"` has 2 processes numbered from `0` to `1`; this predicate will in fact expand into the two following predicates:

```
process("a", 0). process("a", 1).
```

The sort `"ab"` is a cooperative sort that does not represent any biological component but only a cooperation between biological components `"a"` and `"b"`. In some semantic it is also called a multiplex. Indeed the sort `"ab"` has 4 processes represent all possible combinations between the expression levels of the sorts `"a"` and `"b"` ($4 = 2*2$). Line 3 enumerates every sort of the network from the previous information. In ASP the underscore is a placeholder for any value. Finally, all the actions of the network are defined in lines 4-9; We can explain the first predicate `action("a",0,"b",0,1)` represents the action $a_0 \to b_0 \Rsh b_1$.

### B. Search of fixed points

The enumeration of fixed points requires to translate the definition of a stable state (Definition 3) into an ASP program. The first step consists of eliminating all processes involved in a self-hit; the other processes are recorded by the predicate `shownProcess`.

```
11 hiddenProcess(A,I) ← action(A,I,B,J,K), A=B.
12 shownProcess(A,I) ← not hiddenProcess(A,I),
13                     process(A,I).
```

Then, we have to browse all remaining processes of this graph in order to generate all possible states, that is, all possible combinations of processes by choosing only one process from each sort (line 14-15). So that lines 11-13 is an optimization to reduce the number of possible states.

```
14 1 { selectedProcess(A,I) : shownProcess(A,I) } 1
15          ← sort(A).
```

The previous constraint rule lines 14-15 creates as many potential answer sets as there are possible states to take into account. Finally, the last step consists of filtering any state that is not a fixed point, or, in other words, eliminating all candidate answer sets in which an action could be played that could envolve the network to another state. For this, we use a constraint: any solution that satisfy the body of this constraint will be removed from the answer set. Regarding our problem, a state is eliminated if there exists an action between two selected processes:

```
16 ← action(A,I,B,J,_), selectedProcess(A,I),
17     selectedProcess(B,J), A!=B.
```

Finally the `selectedProcess(A,I)` permits to define the fixed point which is compound by these selected processes:

```
18 fixProc(A,I) ← selectedProcess(A,I).
```

*Example 3 (Fixed points enumeration):* The PH model of Figure 1 contains 4 sorts: $a$, $b$ and $z$ have 2 processes and $ab$ has 4; therefore, the whole model has $2*2*2*4 = 32$ states (whether they can be reached or not from a given initial state). We can check that this model contains 2 fixed points: $\langle b_0, a_1, ab_2, z_0\rangle$ and $\langle b_0, a_1, ab_2, z_1\rangle$ Indeed, there is no action between each two processes contained in this state so no execution is possible from these states. In this example, no other states verify this property.

If we execute the ASP program detailed below, alongside with the description of the PH model given in Example 2, we obtain two answer sets, which matches the expected result:

```
Answer 1 : fixProc(a,1), fixProc(b,0),
                fixProc(z,0), fixProc(ab,2)
Answer 2 : fixProc(a,1), fixProc(b,0),
                fixProc(z,1), fixProc(ab,2)
```

## IV. Dynamical analysis

In this section, we present at first how to determine the possible behaviour in a PH model after a finite number of steps with an ASP program. Then we tackle the reachability question: are there scenarios from a given initial state that allow to reach a given goal? If yes is it possible to display the shortest path?

### A. Future states identification

In the previous section, enumerating the fixed points did not require to encode the full dynamics of PH, but only a condition. It is a static verification. In this section, we thus implement a dynamic simulation of the PH into ASP. Then it permits to apply an exhaustive analysis to search for the paths allowing to reach the goals.

Firstly, we focus on the simulation, evolution of models in a limited number of steps. We therefore define the predicate `time(0..n)` which sets the number of steps we want to play. The value of `n` can be arbitrarily chosen; for example, `time(0..10)` will compute the 11 first steps, including the initial state. In order to specify such an initial state, we add several facts to make a list of all processes included in this state:

```
23 init(activeProcess("a",0)).
24 init(activeProcess("b",0)).
25 init(activeProcess("ab",0)).
26 init(activeProcess("z",0)).
```

where `"a"` is the name of the sort and `"0"` the index of the active process. The dynamics of a network is described by its actions; therefore, identifying the future states requires to first identify the playable actions for each state. We remind that an action is playable in a state when both its hitter process and target process are active in this state (see Definition 2). Therefore, we define an ASP predicate `playable(action(A,I,B,J,K),T)` that is true when the processes $A_I$ and $B_J$ are active at step `T`.

The cardinality rule of lines 27 creates a set of as many predicates as there are possible evolutions from the current step, thus reproducing all possible branchings in the possible evolutions of the model in the form of as many potential answer sets. It is also needed to enforce the strictly asynchronous dynamic which state that exactly one process can change between two steps. We thus represent the change of the active process of a sort by the predicate `change(B,T+1)` which means that in sort `B`, the active process can change between steps `T` and `T+1`. To remove all scenarios where two

or more actions have been played between two steps, we use the constraint of line 28. Thus, the remaining scenarios contained in the answer sets all strictly follow the asynchronous dynamics of the PH.

```
27 0{play(Action,T)}1 ← playable(Action,T), time(T).
28 ← 2{play(Action,T)}, time(T).
29 change(B,T+1) ← play(action(_,_,B,_,_),T),time(T).
```

Finally, the active processes at step `T+1`, that represent the next network state depending on the chosen bounce, can be computed by the following rules:

```
30 instate(activeProcess(B,K),T+1) ←
31         play(action(_,_,B,_,K),T), time(T).
32 instate(activeProcess(B,K),T+1) ←
33         not change(B,T+1),
34         instate(activeProcess(B,K),T), time(T).
```

In other words, the state of step `T+1` contains one new active process $B_K$ resulting from the predicate `play(action(_,_,B,_,K),T)` (lines 30-31) as well as all the unchanged processes that correspond to the other sorts (lines 32-34).

The overall result of the pieces of program presented in this subsection is an answer set containing one answer for each possible evolution in `n` time steps, starting from a given initial state.

### B. Reachability verification

In this section, we focus on the reachability of a set of processes which corresponds to the reachability property (see Definition 4): "Is it possible, starting from a given initial state, to play a number of actions so that a set of given processes are active in the resulting state?" We now want to use the implementation of the dynamics computation of the previous section in order to solve this reachability problem. For this, we first use a predicate `goal` to list the processes we want to reach and we add as many rules of the following form as there are objective processes:

```
35 goal(activeProcess("z",1)).
```

Then, the literal `reached(F, T)` checks if a given active process `F` of the goal is contained in the state of step `T`, as defined in the rule of line 36. Else the answer will be eliminated by a constraint (not detailed here) which verifies if all processes of the goal are satisfied.

```
36 reached(F, T) ←  goal(F), instate(F, T).
```

However, the limitation of the method above is that the user has to decide upstream the number of computed steps that should be sufficient to reach all the goals. It is a main disadvantage like in [**?**] because a search in $N$ steps will find no solution if the shortest path to the goal requires $K$ steps with $K > N$. It may also needlessly lengthen the resolution if the shortest path requires $n$ steps with $n << N$. One solution is then to use an incremental computation mode, which is especially tackled by the incremental solver of CLINGO [**?**]. The corresponding syntax separates the program in 3 parts. The `#program base` part contains only non-incremental elements and is thus used to declare general rules that do not depend on the time steps (such as the data related to the model). The body iteration is then written in the `#program step(t).` and `#program check(t).` parts, which are computed at each incremental step. The step number is not given by a variable but by a constant placeholder called "`t`" in the following. The first part comprises rules depending on the time step, and the second contains constraints that stops the iteration when needed.

When using this new syntax, the obtained program is almost identical to what was presented before, except that step numbers `T` are replaced by the constant placeholder `t`. In each step `t`, the program computes:
- the playable actions `playable(Action,t)`,
- the choosen action to be played `play(Action,t)`, - the possible bounces `change(B,t)`
- the new states `instate(activeProcess(A,I),t+1)`

in the `#program step(t).` part the same way than previously, but only for the current step. The solver then compares its current answer sets with the `t`-dependent constraint given in the `#program check(t).` part. Regarding our implementation, this constraint is given in line 38 and simply states that all goals have to be met. If this constraint invalidates all current answer sets, the computation continues in the next iteration in order to reach a valid answer set. As soon as some answer sets meet the constraints, they are returned and the computation stops.

```
37 notReached(t) ← goal(F), not instate(F,t).
38 ← notReached(t), query(t).
```

## V. COMPARATIVE PERFORMANCE ANALYSIS

In this section, we show the effectiveness of our approach on some examples, and compare it to other existing approaches. All computations (except the one called ASP-THOMAS) were performed on a Pentium V, 3.2 GHz with 4 GB RAM.

### A. Evaluation

To assess the efficiency of our new approach, we position ourselves with respect to existing methods dealing with different biological models. We have chosen the following tools, that are detailed below: GINSIM[3] (Gene Interaction Network Simulation) [**?**], [**?**], [**?**], LIBDDD[4] (Library of Data Decision Diagrams) [**?**], [**?**], PINT[5] [**?**] and the method for CTL model-checking proposed by Rocca *et al.* in [**?**], which was developed also in ASP but for states transitions networks. Each method uses a specific kind of representation[6]: Thomas models (a particular kind of logical regulatory networks) for GINSIM, instantiable transition systems for LIBDDD, state transition networks for the method of Rocca *et al.* and Process Hitting (PH) for PINT as well as for our method.

For this comparative study, we focus on biological network of different sizes: a tadpole tail resorption (TTR) model with

[3]GINSIM version 2.4 alpha: http://ginsim.org/

[4]LIBDDD version 1.8: http://move.lip6.fr/software/DDD/

[5]PINT version 2015-11-14: http://loicpauleve.name/pint/

[6]When available, we used the converters included into PINT for these translations.

12 biological components [**?**], an ERBB receptor-regulated G1/S transition (ERBB) model with 20 components [**?**] and a T-cell receptor (TCR) signaling network of 40 components [**?**]. These models were chosen to be of different sizes: from small (12 components) to large (40 components). We note however that the considered PH models may contain more sorts than the original number of biological components, due to the use of "cooperative sorts", which allow to model Boolean gates but do not necessarily have a biological meaning. The different model representations that are required for performing these benchmarks have been obtained by translations from the PH ensuring the conservation of the dynamical properties. The specification of the models and the results of our stable state enumeration are summed up in Table I. The time performance is roughly the same than the SAT implementation that comes with PINT. The results for several methods regarding reachability properties are summarized in Table II. The methods and the results provided by each of them are detailed in the following. The overall results show that our method is efficient in computing reachability from a given initial state; furthermore, it sometimes provides more information than the other existing ones.

- **GINSIM** is a software for the edition, simulation and analysis of gene interaction networks. It allows to compute all reachable stable states from a given initial state instantly; however, it is not possible to compute all stable states independently from the initial state. Regarding the reachability problem, GINSIM only allows to check the reachability of full states, because its approach consists in computing (part of) the state-transition graph and then searching for a path between the two given states. Therefore, it was not possible to perform reachability checks on partial states (experiments #3 & #5). Small state-transition graphs can also be displayed by this tool.
- **LIBDDD** is a library for symbolic model-checking of CTL & LTL properties. It can thus especially be used to check reachability properties; however, as opposed to our method, it does not output an execution path solving this reachability. In addition, it relies on the construction of the state-transition graph which is then stored under the form of a binary decision diagram for a more efficient analysis. This computation explains why LIBDDD takes more time to respond, and gets out of memory in about 12 minutes for the biggest example which contains $2^{73}$ states (experiments #4 & #5). Finally, LIBDDD is not able to compute the stable states of a network.
- **PINT** is a library gathering tools and converters related to the PH. It should be noted that PINT contains the only reachability analysis developed so far natively for the Process Hitting, before the method proposed in this paper. It consists in an approximation that avoids to compute the state-transition graph; it is thus ensured to be really efficient, which explains the fastest results, but at the cost of possibly terminating without being conclusive. However, it is not designed for goals consisting of many processes, which are more likely to trigger an inconclusive response (such as for experiment #4), or an exponential research in sub-solutions (such as for experiment #2). This explains the high computation times for some tests in Table II. Moreover, in the case of a positive answer, it currently does not return the execution of the path achieving the desired reachability, but only outputs its conclusion.
- **ASP-THOMAS**[7] offers the possibility to model-check CTL properties of Thomas networks. There is however no automatic way that allows the modeling of Thomas networks in ASP, which currently has to be made by hand and requires labeling. As for our method, we use the PH whose actions are easily represented in ASP with one fact per action. In addition, the method "ASP-THOMAS" requires to provide a maximum number of steps for which the dynamics will be computed, which may be difficult to be predicted. However it is clear that this approach terminates very quickly when compared with others. Indeed it also shows that ASP is a good choice to run the dynamics of a model and check reachability properties. Furthermore, this method is able to check any kind of CTL formula, and not only the "EF" form that we focused on in this paper (see the discussion in Section V-B). Table III give a summary of the differences between our approach and ASP-THOMAS. The main difference with is that in ASP-THOMAS the search is bounded. If the number of step to reach the goal is part of the goal or if one can know a priori know the length of the path to reach the specified goal, this approach can provide very good performance. But if the bound is totally unknown, choosing a too small path length can lead to missing the goal and thus to conclude unreachable when it is reachable. On the other hand, choosing a to big bound will greatly impact the performance. On ERBB partial, the goal is reachable with a path of 18 steps. Using a bound of 21, ASP-THOMAS finish in 2.61s. If we fix the bound to 30 it takes several minutes. We can observe similar results on the others benchmarks. With our approach, if the goal is reachable, the run time only depend of the distance to reach the goal. In ASP-THOMAS, run time only depends of the chosen bound, because all paths of the chosen length will be generated before being checked.

### B. *Strengths and limitations of our method*

In the previous sections, we developed new methods in ASP to check dynamical properties, namely identifying stable states and finding all possible paths to reach a given goal. Compared to some other methods describes above (GINSIM and LIBDDD) our method is relatively faster and also permits to study larger networks (up to $2^{73}$ states in our tests). We exclusively study networks modeled in Process Hitting. This new formalism for network modeling is a restriction of synchronous automata and thus allows to represent any kind of dynamical model. Moreover it is easy to implement the dynamics of the PH into ASP. However, our ASP program may still be non-conclusive in the cases where the given goal is not reachable. In practice, we can detect the loops in the model dynamics and avoid to check the same path again. But still, iClingo will continue to iterate until the goal is reached, while remaining in the same state, thus it will never terminate. To our knowledge, there is still no possibility to force the incrementation to stop in iClingo except by putting loop detection as a goal, then the search will stop when a loop is found and will not try to search for reachability. It is still possible, however, to limit the number of iterations to an arbitrary maximum which will be eventually reached in

[7]The authors wish to thank Laurent Trilling for his help.

| Models | | PH representation | | | Fixed points enumeration | |
|---|---|---|---|---|---|---|
| Name | Components | Sorts | Processes | States | computation time | Nbr of results |
| TTR | 8 | 12 | 42 | $2^{19}$ | 0.004s | 0 |
| ERBB | 20 | 42 | 152 | $2^{70}$ | 0.017s | 3 |
| TCR | 40 | 54 | 156 | $2^{73}$ | 0.021s | 1 |

Table I. DESCRIPTION OF THE MODELS USED IN OUR TESTS AND RESULTS OF OUR FIXED POINT ENUMERATION. EACH MODEL IS REFERRED TO BY ITS SHORT NAME, WHERE TTR STANDS FOR THE TADPOLE TAIL RESORPTION MODEL [**?**], ERBB FOR THE RECEPTOR-REGULATED G1/S TRANSITION OF THE SAME NAME [**?**] AND TCR FOR THE T-CELL RECEPTOR SIGNALING NETWORK [**?**]. FOR EACH OF THEM, THIS TABLE GIVES THE NUMBER OF BIOLOGICAL COMPONENTS IN THE ORIGINAL REPRESENTATION, AND THE NUMBER OF SORTS, THE NUMBER OF PROCESSES AND THE NUMBER OF STATES IN THE PH MODEL. FINALLY, THE LAST COLUMN GIVES THE COMPUTATION TIME FOR THE ENUMERATION OF ALL FIXED POINTS AND THE NUMBER OF RESULTS RETURNED.

| Experiments | | | Results | | | | |
|---|---|---|---|---|---|---|---|
| | Model | Target | ASP-THOMAS | PINT | LIBDDD | GINSIM | ASP-PH |
| #1 | TTR | full state | 0m7.21s | 0m0.97s | 0m1.15s | 0m2.05s | 0m1.90s |
| #2 | ERBB | full state | 0m2.45s | out | 1m55.38s | 2m31.64s | 0m11.84s |
| #3 | ERBB | partial state | 0m2.61s | 0m0.03s | 1m54.96s | - | 0m5.02s |
| #4 | TCR | full state | 0m7.61s | Inconc | out | out | 6m27.93s |
| #5 | TCR | partial state | 0m2.12s | 0m0.02s | out | - | 1m35.08s |

Table II. COMPARED PERFORMANCES OF SEVERAL METHODS TO COMPUTE REACHABILITY ANALYSES: THE METHOD OF ROCCA *et al.* (DENOTED BY ASP-THOMAS), PINT, LIBDDD, GINSIM AND OUR NEW METHOD PRESENTED IN THIS PAPER, CALLED ASP-PH. FOR EACH TEST, THIS TABLE GIVES THE SHORT NAME OF THE CONSIDERED MODEL, AS GIVEN IN TABLE I, THE TYPE OF GOAL (EITHER A WHOLE STATE OR A SUB-STATE) AND THE COMPUTATION TIME OF THE DIFFERENT METHODS USED FOR THE TESTS, WHERE OUT MARKS AN EXECUTION TAKING TOO MUCH TIME OR MEMORY, - INDICATES THAT IS NOT POSSIBLE TO DO THE TEST, AND INCONC STATES THAT THE METHOD TERMINATES WITHOUT A RESPONSE.

such case. This is possible with the option "`--imax=n`" of ICLINGO, where `n` is the maximum number of steps.

Several values can be given to this parameter `n`. For example, the total number of states is an obvious maximum, as it will never be exceeded by a minimum path, but it is too hight to be very interesting. The total number of sorts is a more interesting value, under the hypothesis that each one will change its active process at most once, which is often the case for Boolean networks; or, with a similar reasoning, the total number of processes can be chosen. In these cases, however, a termination with no solution cannot be considered as a formal negative answer, unless one can prove that the chosen value `n` is bigger than the longest possible path without loop in the state-transition graph. Given our implementation, if the step `n` gives no valid path, the computation stops with an `unsatisfiable` response.

| | ASP-THOMAS | ASP-PH |
|---|---|---|
| Input Model | Thomas network | Process hitting |
| Input generation | Hand-made | **Automatic** |
| Reachability | Bounded | **Unbounded** |
| Non-reachability | Bounded | Bounded |
| Search | Depth first | Breadth first |

Table III. QUALITATIVE COMPARISON BETWEEN OUR APPROACH AND ASP-THOMAS.

## VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we gave a new method to compute some dynamical properties on Process Hitting models, a subclass of asynchronous automata. The main interest of our method is the use of ASP, a declarative programming paradigm which benefits from powerful solvers. We first focused on the enumeration of the fixed points of a model, which is tackled simply on such models given their particular form. We also considered the reachability problem, that is, checking if it is possible to reach a state with a given property from a given initial state, which thus corresponds to an EF operator in CTL logic. Our analysis is thus exhaustive, but can be limited to a number of steps, for which the dynamics of the model from the given initial state is computed. We gave an implementation of these problems into ASP, and applied them to several biological examples of various sizes, up to 40 biological components. Our results showed that our implementation is faster and deals with bigger models than other approaches, especially LIBDDD which is a symbolic model-checker.

Our work could benefit from several extensions. Of course, the set of applicable models can be extended, for example with the addition of priorities or neutralizing edges, or by considering synchronous dynamics or other representations such as Thomas modeling [**?**]. However, the range of the analysis can also be extended, by searching instead the set of initial states allowing to reach a given goal, or extending the method to universal properties (like the AF operator). Finally, the research of attractors in a more general fashion (such as cyclic or complex attractors) would be of major interest to fully understand the behavior of models.


## ACKNOWLEDGMENT

The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 259267.