Exhaustive dynamic analysis of reachability and stable states of biological regulatory networks modeled in Process Hitting using Answer Set Programming

Emna Ben Abdallah, Olivier Roux, Morgan Mangnin, and Maxime Folschette

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.
{emna.ben-abdallah,olivier.roux,morgan.magnin,
maxime.folschette}@irccyn.ec-nantes.fr
http://www.irccyn.ec-nantes.fr

Abstract. The Process Hitting is a recently introduced framework to model concurrent processes. It is notably suitable to model biological regulatory networks with partial knowledge of co- operations by defining the most permissive dynamics. In this paper, we explain the methods we developped with ASP to find the fixed points or states states in which it is not possible any more to have evolutions of the model. We also aim at solving the problem of reachability that consists of deciding if, starting from a given initial state, it is possible to reach a given local state. Finally, we illustrate the merits of our methods by applying them to various biological examples ans comparing the results with eexisted aproches. We show that our approach succeeds in processing large models.

Keywords: Process Hitting, stable states, fix points, reachability, Answer Set Programming

1 Introduction

As regulatory phenomena play a crucial role in biological systems, they need to be studied accurately. Biological Regulatory Networks (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 make it possible to determine the possible evolutions of all the interacting components of the system. Indeed, in order to address the formal checking of dynamical properties within very large BRNs, we recently use a new formalism, named the "Process Hitting" (PH) [12], 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 at most one other "process" in the system. This particular structure makes the formal analysis of BRNs with hundreds of components tractable.

PH is suitable, according to the precision of this information, to model BRNs with different levels of abstraction by capturing the most general dynamics. The objectives of the work presented in this paper are the following.

Firstly, we show that starting from one PH model, it is possible to find all possible stable states (fixed points [16]). We perform an exhaustive search of the possible states, combination processes, one process from each sort and then check if it is a fixed point.

The second phase of our work consists in computing the dynamics. It consists in determining from a known initial state the possible next states of the PH model. Finally we verify if we can reach a specific state of one or several component (gene or protein). The results are ensured to respect the PH dynamics.

Our contribution is from the results that allowed to determine the stable states, we propose to evaluate the benefits of the Answer Set Programming (ASP) [2] to compute them. ASP 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 the PH model. In this paper, we show that ASP turns out to be effective for these enumerative searches which justifies its use. The benefit of our approach is that it makes possible to get the minimal paths to reach our goal(s) also we can verify if it is possible after a given number of steps.

2 Preliminary definitions

In order to develop the nes methods we have used two main frameworks. The first is a logic programming language: the Answer Set Programming (ASP). Precisely it is a logic programming language on which I wrote my bibliographic research report. The second framework is the Process Hitting (PH) a new formalism for the representation of biological regulatory networks. We will see later in this section more details about these two frameworks.

2.1 Answer Set Programming

The ASP is a declarative programming paradigm with known semantics as the semantics of answer sets. This paradigm allows the programmer to specify what the problem to be solved and not how to solve it. ASP programs are written in AnsProlog* (short for "Answer Set Programming in Logic" with superscript *). These programs are composed of a set of facts and a set of rules from which other facts can be derived. A consistent set of facts that can be derived from a program using the rules is called "answer set" for the program. The sets of possible responses to an AnsProlog* program are calculated with a program called a solver.

Logic program Consider an ASP logic program, as each rule is an ordered pair,[2] :

$$head \leftarrow body. \tag{1}$$

with *Head* and *Body* sets of literals.

This simple language has many advantages that make it one of the most effective in terms of knowledge representation, reasoning and declarative problem solving.

The syntax and semantic of ASP

1. **The alphabet**
   According [2], the alphabet of the axiom (or just the alphabet) of the framework of the answer set is comprised of seven classes of symbols:
   - Variables,
   - Constants,
   - Symbols of functions,
   - Symbols of predicates,
   - Connectors,
   - Symbols of punctuation, and
   - Special symbol $\bot$

   All these classes vary from one alphabet to another, but the sets of 5^{th} and 6^{th} classes (Connectors and symbols of punctuation) are defined as follows:

   - Connectors with {¬, or, ←, not, ‘,’}
   - Symbols of punctuation with {‘(’, ‘)’, ‘.’}

   The other classes remain constant as they meet some informal agreement. In general, variables start with a capital letter and contain letters and numbers(X,Y,..). The constants, symbols and predicates follow the same rule, but they begin with a lowercase letter(f,g,a,b...). Sometimes there is the addition of a supplementary agreement which covers the letters used [2] :

   What seems a little fuzzy is the concept of predicate. Indeed the word predicate can be an innovation of the new grammar. Let’s consider the following sentence:

   Socrates was an athenian.

   Socrate is the subject and *was an athenian* is the predicat. This sentence can be noted in ASP by :

   athenian(Socrate) $\leftarrow \top.$ with $\top$ is the symbol of True.

2. **The rules** Comme nous l’avons déjà mentionné, une règle est de la forme:

$$head \leftarrow body. \quad ((1))$$

$$A \leftarrow B^{+}, \textbf{ not } B^{-}. \quad (2)$$

This rule is then equivalent to:

$$L_0 \leftarrow L_1, ..., L_m, \mathbf{not} L_{m+1}, ..., \mathbf{not} L_n. \tag{3}$$

And this rule states:

$$\left.\begin{array}{l}\text{Si } L_1, ..., L_m \text{ are } \mathbf{true} \\ \text{et } L_{m+1}, ..., L_n \text{ are } \mathbf{false}\end{array}\right\} \text{Then } L_0 \text{ is } \mathbf{true}$$

with L_i literals and k $\geq$ 0, m $\geq$ k and n $\geq$ m.
Head is also called conclusion
Body is also called premise
Special cases rules [9] and [2] :

- A rule is **constant** if all litterals are constants (noted with **ground**);
- A **fact** or a **reality** : it is a rule with an empty body. It can be written even without the arrow $\leftarrow$:

$$L_0. \ \ or \ \ L_0 \ \leftarrow \ \top. \tag{4}$$

- A **constraint**: This false symbol to the head (*head*) is often eliminated and the constraint will be written generally as:

$$\leftarrow \ L_1, ..., L_m, \ \mathbf{not} \ L_{m+1}, ..., \ \mathbf{not} \ L_n. \tag{5}$$

  We say that a set of literals X violates the constraint (5) if $\{L_1, ..., L_m\} \subseteq X$ and $\{L_{m+1}, ..., L_n\} \nsubseteq X$. If we have a program contain this type of rule 5, then X is an answer set of the program π if and only if X is an *answer set* of $\pi \setminus \{r\}$, with r such a constraint 5 and X does not violate the constraint 5.
- Cardinality constraints:
  It is literals extended in the following form:

$$l \ \{q_1, \ ..., \ q_m\} \ u. \tag{6}$$

  with $m \geq 1$, l an integer and u can be an integer or by default the infinite if it does not exist. l and u are the lower and upper limits of the cardinality of the subsets of $\{q_1, \ ..., \ q_m\}$ that satisfy the answer sets. These literals are constrained (q_i) may occur in the head and body of the rule. Cardinality constraint is satisfied in an answer set X, if the number of atoms of $\{q_1, \ ..., \ q_m\}$ belonging to X is between l and u. In other words:

  $l \ \leq \ |\{q_1, ..., \ q_m\} \ \cap \ X| \ \geq \ u$

  with $\cap$ symbol of intersection and $|A|$ is the cardinality of the set A.

Modelisation of a problem with ASP We can consider that the construction of models is one of the fundamental components of the scientific process. It concerns all systems we seek to control. A model has two main features [1]:

- it is a simplification of a given system
- it allows an action on the system

Models offer the possibility to provide a solution to a problem identified as such. This concept model can address another angle issues related to representation process.

1. **Modeling steps** The modeling process can be regarded as a special form of representation whose operations are detailed in [2]. Logical ASP programs follow the strategy of "generate and test". This strategy includes four steps:
   - Enumerate with facts;
   - Explain with the rules;
   - Generate all the possibilities with cardinalities, and finally;
   - Filter with constraints.
2. **Resolving an ASP program**
   From [9] and [6], the couple Grounder and Solver usually work together: the grounder used to remove variables in order to achieve a constant program and the solver computes all anwser sets for stabilized programs generated by the grounder.
   Example of Grounder: `GRINGO`, `DLV`, `LPARSE`
   Example of ASP Solver : `SMODELS`, `DLV`, `CMODELS`, `CLASP`...
   The combined use of the Grouder and the Solver specifies major programs in a compact, using rules with schematic variables and other abbreviations. Both systems employ grounding algorithms that work quickly and simplify the program. In developping these new methods we worked with `CLINGO` which is a combination of grouder `GRINGO` and solver `clasp`.

2.2 Process Hitting

Definition 1 introduces the Process Hitting (PH) [12] 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*. Actions describe the replacement of a 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 $a_i \to b_j \not\uparrow b_k$, 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: $a_i \to a_i \not\uparrow a_k$.

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

- $\Sigma = \{a, b, \dots\}$ *is the finite set of* sorts*;*
- $L = \prod_{a \in \Sigma} L_a$ *is the set of states with* $L_a = \{a_0, \dots, a_{l_a}\}$ *the finite set of* processes *of sort* $a \in \Sigma$ *and* l_a *a positive integer, with* $a \neq b \Rightarrow L_a \cap L_b = \emptyset$*;*
- $\mathcal{H} = \{a_i \to b_j \not\uparrow b_k \in L_a \times L_b^2 \mid (a, b) \in \Sigma^2 \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ *is the finite set of* actions.

Example. Figure 1 represents a PH $(\Sigma, L, \mathcal{H})$ with three sorts ($\Sigma = \{a, b, c\}$) and: $L_a = \{a_0, a_1\}$, $L_b = \{b_0, b_1, b_2\}$, $L_c = \{c_0, c_1\}$.

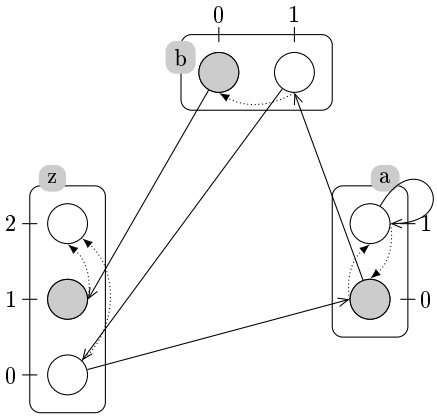

Fig. 1. A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0 or 1 and z at either level 0, 1 or 2). Boxes represent the *sorts* (network components), circles represent the *processes* (component levels), and the 4 *actions* that assure the dynamic behavior are drawn by pairs of arrows in solid and dotted lines. The grayed processes stand for a possible initial state.

A network state is a set of active processes and there is a single process for each kind. In a state $s \in L$, we say that a process $a_i \in s$ with a the name of the sort and i the level of this sort at this state, and we denote by $s[a] = a_i$.

Definition 2 (Playable action).
Let $PH = (\Sigma, L, \mathcal{H})$ *be a Process Hitting and* $s \in L$ *a state of* PH*. We say that the action* $h = a_i \to b_j \not\uparrow b_k \in \mathcal{H}$ *is playable at the state* s *if and only if* $hitter(h) = a_i \in s$ *and* $target(h) = b_j \in s$ *(i.e* $s[a] = a_i$ *et* $s[b] = b_j$ *)*

The resulting state after playing an action h at an initial state s is denoted by $(s \cdot h)$ or $(s \cdot h)[b] = b_k$ and $\forall c \in \Sigma,\ c \neq b, (s \cdot h)[c] = s[c]$.

Dynamic properties

The study of the dynamics of biological networks was the focus of many researchers, justifying the diversity of network modelisations and the different methods developed in order to verify the dynamic properties. In this paper we present 2 properties: the stable state and the reachability. We develop in this section these properties and how they could be verified in PH network.

The dynamic of a network or its evolution is realised through its actions. In fact when an action $h = a_i \rightarrow b_j \Rsh b_k$ is played at a state s, the network evolves to the next state results s'. We found so in the state s' all processes of s with the replacement of process $b_j = cible(h)$ by $b_k = bond(h)$.

Definition 3 (Next state).
Let $(\Sigma, L, \mathcal{H})$ be a Process Hitting and $s \in L$ be one of its states. The set of the next possible states for s are computed as follows:

$$next(s) = s[b_k/b_j] | \exists (ai, bj) \in s^2, \exists b_k \in L_b, a_i \rightarrow b_j \Rsh b_k \in \mathcal{H}$$

We call the set of succession of actions $(h_0 \cdot h_1 \cdot h_2 ...)$ from a state s_0 a *scenario* noted $\mathbf{Sce}(s_0)$ corresponding to the succession of states $(s_0 \mapsto s_1 \mapsto s_2 ...)$.

Definition 4 (Stable state or fixed point).
Let $PH = (\Sigma, L, \mathcal{H})$ be a Process Hitting and $\{s_0, ..., s_n\} \in L$ be state, s_n is a stable state for PH if and only if $next(s_n) = \emptyset$.

$$s_0 \mapsto s_1 ... \mapsto s_n \mapsto \emptyset$$

Definition 5 (Lemme: Stable state or fixed point). *Let $PH = (\Sigma, L, \mathcal{H})$ be a Process Hitting and $s \in L$ be a state, s is a stable state for PH there is no playable action at the state s:*

$$\forall h \in \mathcal{H},\ frappeur(h) \notin s \ \vee \ cible(h) \notin s)$$

Definition 6 (Reachability question).
If $s \in L$ is a state and $A \in$ **Proc** *a set of processes, we note $\mathcal{P}(s, A)$ for the* the reachability question*:*

$$\exists ? \delta \in \mathbf{Sce}(s), A \subseteq ([s] \cdot \delta) \ (i.e\ \forall a_i \in A, (s \cdot \delta)[a] = a_i).$$

with $\mathbf{Sce}(s)$ *is the set of successively playable actions from the state s.*

3 Fixed point implementation

The study of fixed points (or basins of attraction) provides an important understanding of the different behaviors of a BRN (Biological Regulatory Network) [16]. The fixed point is a stable state of the BRN in wich it is not possible any more to have new changes. Let (Σ, L, H) be a Process Hitting . It has been shown that a state $s \in L$ is a fixed point of the Process Hitting if and only if s is has no next state [12] i.e. there is no playable action at this state. In fact a steady state od a Process Hitting network is a set of processes with exactly one process of each sort as well as every process has no hit with the other selected ones (process with self-hit cannot be apart of a stable state).

3.1 Process Hitting network traduction

In order to handle a PH biological network, it was necessary first to present it with ASP. To do this we chose the predicates: *sort*, *process* and *action*. Below is an example of PH network shown in ASP.

Example (Example PH network with ASP). If we try to present the network of Figure 1 we will have:

```
1 sort("a"). sort("b"). sort("z").
2 process("a", 0..1). process("b", 0..1). process("z", 0..2).
3 action("a",0,"b",1,0). action("a",1,"a",1,0). action("b",1,"z",0,2).
4 action("b",0,"z",1,2). action("z",0,"a",0,1).
```

Line 1 shows every sort of network with the predicate comes out and in parentheses the name of the sort. In line 2 there is a list of processes corresponding to each *sort* for example the sort "z" has 3 process numbered from 0 to 2, this numbering is provided by the 2^{nd} parameter of the predicate $process(\text{"}z\text{"}, 0..2)$. Finally we find all network actions that are defined in lines 3 and 4, for example the first action $action(\text{"}a\text{"}, 0, \text{"}b\text{"}, 1, 0)$ is an action from a_0 to b_1 to bound so b from b_1 into b_0.

3.2 Search of fixed points

Parler du point fixe et ses interrêts.
Decrire la nouvelle méthode qui permettent de determiner les états stables d'un réseau

In fact we have to translate the definition of a stable state in a method developped in ASP. So first we eliminate all processes with self-hit and we save them in the predicate "*shownProcess*":

```
5 hiddenProcess(A,I) ← action(A,I,B,J,K), A=B, process(A,I),process(B,J),
6                      process(B,K).
7 shownProcess(A,I) ← not hiddenProcess(A,I), process(A,I).
```

Then we have to browse this graph and extract all possible combinations of shown processes by choosing a process from each sort.

```
8 1 { selectProcess(A,I) : showProcess(A,I) } 1 ← sort(A).
```

It now remains to check each combination of processes whether it is a fix point. For this we use a special type of ASP rules: a **constraint**. The idea of constraints is based on the fact that a solution would be eliminated if it does not satisfy the constraint. For our problem a combination is eliminated if there is an action between two of the selected process:

```
9 ← hit(A,I,B,J), selectProcess(A, I), selectProcess(B, J), A!= B.
```

Example 1. Considering the last graphical presentation of a precess hitting Figure 1, there are 3 sorts a, b have 2 levels and z has 3 so we can find $2*2*3 = 12$

states (whatever they can be reached or not). If we verify wether there exist fixed points we deduce that we have only one: $< b\_0, z\_2, a\_0 >$. It is clear in Figure 1 that there is no action between each two processes. Besides our new ASP method proof this and returns also the same answer:

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

4 Dynamic network evolution

In this section, we will present firstly how to determine using ASP the possible evolution of a biological network after a finite number of steps. Then we answer to the reachability question; which evolutions that allow the achievement of goals (future active processes) from a known initial state?

4.1 Future states identification

From an initial known state, a PH network can evolve into several new states after a few steps. The predicate "$time(0..n)$" sets the number of steps we want to play. For example if the biologist wants to know which states are reachable after 10 stages, it has only to replace n by 10, and he will have "$time(0..10)$". To compute the future states it is necessary to define an initial state of the network. So for initializing the active processes we add in the ASP script representing the PH network the following rule for each sort. We note that this rules are generated automatically at the time of translating PH network to ASP and by default the level of all active processes is 0.

```
10 init(activeProcess("a",0)).
```

with a is the name of the sort and 0 the index of the active process.
The dynamic of a network is realised thanks to its actions. So to identify the future states we should start by identifying the playable actions for each state. We recall that an action is playable when both precesses: hitter and taget are active (Definition 2). In ASP the predicate "$playableAction(A, I, B, J, K, T)$" it is true when the processes (A, I) and (B, J) are active at the step T. Our approch is to study the asynchronous dynamic so only one action is playable at the same step (T). That's why between two successive states there will be one change of one sort. We present this by the predicate "$activeFromTo(B, J, K, T)$". It means that in the sort B the active process change from index J to K at time step T. At the same step we can find many possible changes , that's why the rule (line 13) offers a set of all these possible changes thanks to braces. In fact the rule is encoded with a count atom at its head, which makes it a choice rule. Rule in line 14 filters any answer with more than 1 change at the same time T to ensure the asynchronous evolution.

```
11 {activeFromTo(B,J,K,T)} ← playableAction(A,I,B,J,K,T),
12           instate(activeProcess(A,I),T), instate(activeProcess(B,J),T),
13           J!=K, time(T).
14 ← 2{ activeFromTo(B,J,K,T)}, time(T).
```

In order to determinate the next active processes at $T+1$ we use the following rules :

```
15 instate(activeProcess(B,K),T+1) ← activeFromTo(B,J,K,T), time(T).
16 instate(activeProcess(A,I),T+1) ← instate(activeProcess(A,I),T),
17           activeFromTo(B,J,K,T), A!=B, time(T).
```

At the next step $T + 1$ we find the new active process resulted from the predicate *activeFromTo* (line 15) as well as all the unchanged processes that correspond to the other sorts (line 17).
As a result it will be displayed all possible evolutions of the netwoks.

4.2 Reachability verification

In this section, we focus on the reachability of a process which corresponds to the question [**?**] :
"Is it possible, starting from a given initial state, to play a number of actions so that a given processes are active in the resulting state?"
Now we try to adapt the code of network evolution to resolve the reachability problem. First we define a predicate for the objective processes we call it "*goal*", we add a rule with this predicate to the script defining the PH :

```
18 goal(activeProcess("a",1)).
```

The rule line 18 verifies if after the network evolution, its state satisfies the goals at step T. Else the answer will be eliminated.

```
19 satisfiable(F,T) ← goal(F), instate(F,T).
```

The limitation of this method is that the user has to choose the number of steps of the evolution. It is an disadvantage because a search in N steps will find no solution if the shortest path to solve the rechability requires N+1 steps. The solution is to use the incremental computation mode (ICLINGO [4]). So we have almost the same program expect for incremental step numbers. In each step t, the program computes the playable actions $playableAction(A, I, B, J, K, t)$,the possible change $activeFromTo(B, J, K, t-1)$ and the new active processes for the next step $instate(activeProcess(A, I), t+1)$. Regarding the part of local steps we use a special constraint (rule **c4**) that means that the program should continue to the next step if it's not satisfiable so that we eliminate responses that do not meet the goals.

```
20 notSatisfiable(t) ← goal(F), not instate(F,t).
21 ← notSatisfiable(t).
```

5 Comparative performance analysis

In this section, we show the effectiveness of our approach on some examples. All computations were performed on a Pentium V, 3.2 GHz with 4 GB RAM.

5.1 Evaluation

To know the effectiveness of our new approach, we had to position ourselves against existing methods dealing with different biological network models. We have chosen the more known methods: GINsim (Gene Interaction Network Simulation)[5], [10] and [11], LIBDDD (Library of Data Decision Diagrams) [15] and [3], Rocca's et al. method [13] and PINT [12]. Each method use a specific model, we find logical regulatory network for GINsim , Instantiable Transition Systems for LIBDDD and state transition network for Rocca's et al. method [13].
For our comparative study we chose organic network of different sizes. As mentioned each method is applied on a specific model and translation into these models from the PH network or viceversa is done thanks to PINT.

Model	#procs	#states	target	Rocca	libddd	GINsim	ASP
tetard [7]	42	2^{19}	process	7m17.01s	XX	XX	0m01.90s
ERBB_G1-S [14]	152	2^{70}	state	-	1m55.38s	2m01.64s	0m11.84s
ERBB_G1-S	152	2^{70}	process	-	1m54.96s	-	0m05.02s
TCRsig40 [8]	156	2^{73}	process	-	out	out	0m05.02s

Fig. 2. Compared performances of Rocca et al. method, LIBDDD, GINSIM and our new method developped in ASP. For each test we find the name of the model, the number of its processes and its states then the target(s). The target can be a network state (set of processes) or one or some processes. "out" marks an execution asking too much time or memory and "-" marks that is not possible to do the test.

This table prensents the comparisons run for 4 dynamic analysis methods applied to examples of biological regulatory networks. The models of these networks where chosen as being representative of different scales, large networks and small networks. This table shows the effectiveness of our method compared to existing. In fact we develop in the sequel that our method provides more compared to other

- **Rocca's method :** It offers the possibility to verify model-checking CTL properties of transition state networks. But we note that it is very difficult to write a transition state network in ASP, for example the network of tetard [7] with only 42 processes, take 3 minutes to be translated from PH network.
- **GINSIM :** It is an analysis software and simulation of genetic interaction networks. It computes the stable states instantly as welle as our ASP method do it instantly.The difference that GINSIM does not return all stable states but only reachable ones. Regarding the reachability problem, its approach is to calculate all the transition state graph and then check if there is a path between two given states. But the large network transition state cannot be displayed. In order to determine the path it is necessary to have the entire

network transition states and both initial and final states. It was indicated in the third line of the table 5.1 that GINsim could not solve the problem because it is not possible to check the reachability of just one or a few components of the network indeed must be given a final state.

- **LIBDDD :** This is a library for model-checking (CTL, LTL and reachability). It returns the answer "True" or "False" after verfying the reachability. However our approach is used to return the activation path of the affected components. In addition it is clear that LIBDDD put much time to respond (23 times slower to example line 3 of the table). That's why for the big examples (2^{73} states) we don't have a response even after 12 minutes but we get it in less than 2 minutes by using the ASP method. Also the LIBDDD do not compute the stable states of a network.
- **PINT :** It should be noted that the only reachability analysis developed so far on the Process Hitting networks was implemented in the software Pint, and consists in an approximation: it is possible that it terminates but remains inconclusive (although this is rare). Moreover, it currently does not give us the path to activate the goal.

5.2 Limitation

We developped new methods for the dynamic properties, identifying stable states and finding all possible paths to reach states. Comparing with other methods (Rocca's method, GINSIM and LIBDDD) is relatively faster and also permit to edit larger networks (about 2^{73} states) but we had also some inconclusif cases. In fact the program is inconclusif if the state is not reachable and there is some loops in the graph so the program may turn indefinitely in one of these loops since it is not possible to achieve the goals or to stop the calculation. We then proposed in this case to limit the increment to a maximum number that can be reached in the case of an endless loop and which may not be in the case where the targets are verified.

```
iClingo find_path.lp <network_name>.lp -imax=n
```

Should be given some interesting values for this integer n, which sets the maximum number of steps. For example, the total number of states is a maximum (it will never be exceeded by a minimum path). However, it can reduce the number of sorts if it is assumed that one type will be changed by step (which is often the case of Boolean networks) or the number of shares with the same assumption. Note that if in a way the goal is reached after a time n, the calculation stops and displays this path as solution.

6 Conclusion and future directions

References

1. Christian Anger, Kathrin Konczak, Thomas Linke, and Torsten Schaub. A glimpse of answer set programming. *KI*, 19(1):12, 2005.

2. Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.
3. Maximilien Colange, Souheib Baarir, Fabrice Kordon, and Yann Thierry-Mieg. Towards distributed software model-checking using decision diagrams. In *Computer Aided Verification*, pages 830–845. Springer, 2013.
4. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo, 2008.
5. A Gonzalez Gonzalez, Aurélien Naldi, Lucas Sanchez, Denis Thieffry, and Claudine Chaouiya. Ginsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Biosystems*, 84(2):91–100, 2006.
6. Robert Ibbotson. Extending and continuing the development of an integrated development environment for answer set programming. Master's thesis, University of Bath, 2010.
7. Zohra Khalis, Jean-Paul Comet, Adrien Richard, and Gilles Bernot. The smbionet method for discovering models of gene regulatory networks. *Genes, Genomes and Genomics*, 3(1):15–22, 2009.
8. Steffen Klamt, Julio Saez-Rodriguez, Jonathan Lindquist, Luca Simeoni, and Ernst Gilles. A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, 7(1):56, 2006.
9. Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002.
10. Aurélien Naldi, Duncan Berenguier, Adrien Fauré, Fabrice Lopez, Denis Thieffry, and Claudine Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009.
11. Aurélien Naldi, Denis Thieffry, and Claudine Chaouiya. Decision diagrams for the representation and analysis of logical models of genetic networks. In *Computational Methods in Systems Biology*, pages 233–247. Springer, 2007.
12. Loïc Paulevé. *Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique*. PhD thesis, École centrale de nantes, 2011.
13. Alexandre Rocca, Nicolas Mobilia, Éric Fanchon, Tony Ribeiro, Laurent Trilling, and Katsumi Inoue. Asp for construction and validation of regulatory biological networks. In Luis Fariñas del Cerro and Katsumi Inoue, editors, *Logical Modeling of Biological Systems*, pages 167–206. Wiley-ISTE, 2014.
14. Regina Samaga, Julio Saez-Rodriguez, Leonidas G Alexopoulos, Peter K. Sorger, and Steffen Klamt. The logic of egfr/erbb signaling: Theoretical properties and analysis of high-throughput data. *PLoS Computational Biology*, 5(8):e1000438, 2009.
15. Yann Thierry-Mieg, Denis Poitrenaud, Alexandre Hamez, and Fabrice Kordon. Hierarchical set decision diagrams and regular models. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–15. Springer, 2009.
16. Andrew Wuensche. Genomic regulation modeled as a network with basins of attraction. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klien, editors, *Pacific Symposium on Biocomputing*, volume 3, pages 89–102. World Scientific, 1998.