Exhaustive search of dynamical properties in Process Hitting using Answer Set Programming

Emna Ben Abdallah

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.

emma.ben-abdallah@eleves.ec-nantes.fr

Joint work with: Maxime Folschette, Olivier Roux, Morgan Magnin

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, 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 a biological example.

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) [3], 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 [4]). 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) [1] 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 Frameworks

2.1 The Process Hitting

Definition 1 introduces the Process Hitting (PH) [3] 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 $a_i \to b_j \,\text{⇱}\, b_k$, 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: $a_i \to a_i \,\text{⇱}\, a_k$.

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

- $\Sigma = \{a, b, \dots\}$ 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 $a \neq b \Rightarrow L_a \cap L_b = \emptyset$;
- $\mathscr{H} = \{a_i \to b_j \,\text{⇱}\, b_k \in L_a \times L_b^2 \mid (a,b) \in \Sigma^2 \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ is the finite set of *actions*.

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

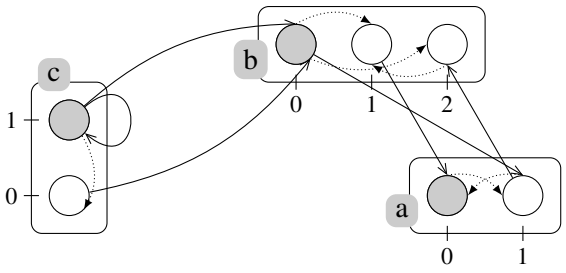

Figure 1: A PH model example with three sorts: a, b and c (a is either at level 0 or 1, c at either level 0 or 1 and b at either level 0, 1 or 2). Circles represent the processes, boxes represent the sorts, and the actions are drawn by pairs of arrows in solid and dotted lines.The grayed processes stand for a possible initial state.

2.2 Answer Set Programming

According to Baral [1], Answer Set Programming (ASP) programs which are written in the language of AnsProlog*, are composed of a set of facts together with a set of rules from which other facts can be derived. A set of consistent facts that can be derived from a program using the rules is known as an *answerset* of the program. These rules are of the form below:

$L_0 \leftarrow L_k, ..., L_m$, **not** L_{m+1},..., **not** L_n.

where each of the L_i is a literal in the sense of classical logic. Intuitively, the above rule means that if L_k,...,L_m are true and if L_{m+1},..., L_n can be safely assumed to be false then L_0 is true.

Example. The below example explains that if a variable X lays eggs so it will be considered as a bird, else if it engenders a baby so it will be considered as a mammal. Then to verify if X can fly or not, it should be a bird and not a mammal. For example we have the fact **D** for $X = tweety$, we can replace it in rule **A** so the result will be true for $bird(tweety)$, but in rules **B** it will be unkown for $mammal(tweety)$ than $notbird(tweety)$ is true. Finally, **C**: $fly(tweety)$, will be true and the conclusion is: $tweety$ can fly.

A: $bird(X) \leftarrow lays\_egg(X)$. **B**: $mammal(X) \leftarrow engender(X)$.

C: $fly(X) \leftarrow bird(X)$, **not** $mammal(X)$. **D**: $lays\_egg(tweety)$.

In fact, ASP is able to process very complex problems, specifically NP ones. In addition, ASP tackles the inherent complexity of the models we use, allowing a fast execution of the formal tools defined in this paper. Also it is convenient to enumerate a large set of possible answers, and it allows us to easily constrain the answers according to some properties. These advantages encouraged us to use and test ASP for the biological networks with a big number of sorts.

There is now a variety of tools for solving ASP. Among them, we find the grounder GRINGO, the solver CLASP, and their combinations within integrated systems CLINGO and ICLINGO.

Note that a numerical argument provided to either CLASP, CLINGO, or ICLINGO determines the maximum number of answer sets to be computed, where 0 stands for "compute all answer sets." By default, only one answer set is computed (if it exists).

According to [2] ICLINGO extends CLINGO by an incremental computation mode that incorporates both grounding and solving. Hence, its input language includes all constructs described in Section 3.1. In addition, ICLINGO deals with logic program declared in three parts: a static part, a part with an incremental step number and a part which is local to steps (all rules are dismissed before the next incremental step). In Section 4.2, we provide our program in which these conditions naturally hold.

3 Fixed point

The study of fixed points (or basins of attraction) provides an important understanding of the different behaviors of a BRN [4]. The fixed point is a stable state of the RRB 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 a $|\Sigma| - clique$ of the hitless associated graph [3] i.e. a set of processes with exactly on process of each sort and every process have no hit with the others selected ones.

Example. Considering the last graph of Figure 1, we construct the corresponding hitless graph. So first we eliminate all processes with self-hit then we add the edges between all two processes which did not have an action between them in the graph with-hit. Then we verify wether there exists an $|\Sigma| - clique$. So we deduce that we have two fixed points: $< a\_1, b\_1, c\_0 >$ and $< a\_0, b\_2, c\_0 >$.

3.1 ASP Program

At the beginning, the idea was to try to implement an algorithm that checks the above definition, transforming the graph with hits (oriented) in a graph without-hit (non-oriented). So in this new graph firstly we eliminate the processes with self-hit by the predicate "$hiddenProcess(A,I)$". Then, we construct the new graph whose edges between two processes of different sorts correspond to the non existence of a hit between them in the graph of PH. In the ASP program these edges are called by "$noAction(B,J,A,I)$" (r1) where A and B are sorts and I and J the indexes of processes.
r1: $noAction(B,J,A,I) \leftarrow$ **not** $hit(A,I,B,J)$, **not** $hit(B,J,A,I), A! = B, showProcess(A,I), showProcess(B,J)$.

Then we have to browse this graph and extract all possible combinations of shown processes (without self-hit) by choosing a process from each sort.
r2: $1\{selectProcess(A,I) : showProcess(A,I)\}1 \leftarrow sort(A)$.
This special rule creates as many answer sets as necessary, with each one containing exactly one $selectProcess(A,I)$ atom for each sort A. Now we have to find wich combinations verifies the characteristics of the fixed point, i.e., to check if the selected process from each sort is related with all the other selected processes of this combination. We present this relation with the predicate "$noHit$".
r3: $noExistFixPoint \leftarrow X < N, getNumberNoHit(X), N = \{sort(\_)\}$.
r4: $\leftarrow noExistFixPoint$.

In (**r3**), sort(_) simply returns the total number of sorts. The rule (**r4**), is a contraint wich eliminates all answers that verify the not N-clique property (**r3**) where N is the number of sorts of the network. Finally the combination of the selected processes verifying all conditions constitute the fix point.
r5: $fixProcess(A,I) \leftarrow selectProcess(A,I)$.

Example. If we apply the method above to the PH of Figure 1 , if we try to find the solution of the ph model, our ASP program will return the below answers:

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

In a biological network, there are sometimes thousands of sorts, so it will be better if the program runs faster. That is why the idea was to try to optimize this ASP script by having less number of predicates. Subsequently, the first improvement is to eliminate the predicates "$noHit(A,I,B,J)$" and "$getNumberNoHits(X)$" which computes the number of the predicate "$noHit$" (edge between 2 selected processes), as a result also the predicate "$noExistFixPoint$" (r3 and r4). These predicates have been replaced by only one constraint:
r6: $\leftarrow 1\{hit(A,I,B,J)\}, selectProcess(A,I), selectProcess(B,J), A! = B$.
We note that the predicate "$hit(A,I,B,J)$" is true when there is an action, within the network, which hitter process (A,I) and target process (B,J) are active. In fact, this above constraint shows that it is sufficient to have two selected processes verifing the predicate *hit* in the hitless to eliminate a solution.

3.2 Results on large networks

Note that both methods give the same result but the second is faster. The following table describes the comparaison and proves that the 2nd method is almost faster than PINT (a library developed to parse and study PH models). The computation is done with a desktop computer (core i5 and 4GB RAM):

Model	#sorts	#procs	#actions	#states	#fix-point	PINT	ASP mthd1	mthd2
ERBB_G1	42	152	399	2^{70}	3	0.017s	0.220s	0.000s
tcrsig40	54	156	305	2^{73}	1	0.021s	0.220	0.020s
tcrsig94	133	488	1124	2^{194}	0	0.027s	2.540	0.060s
egfr104	193	748	2356	2^{320}	0	0.074s	8.220	0.140s

Figure 2: Excecution time of ASP metods applied for biological networks with a desktop computer

4 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 what are the specific evolutions which the evolutions that allow the achievement of goals (future active processes) from a known initial state?

4.1 Computing the dynamics

From an initial known state, a PH 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 what states are reachable (in the PH) after 10 stages, it has only to replace n by 10, and it will be "$time(0..10)$". For initializing, the active process state 0 was added the rule (**e1**) in the ASP script wich represent the PH network

e1: $init(activeProcess("a",0))$. ; a is the name of the sort and 0 the index of process.

According to the state of the network, specifically the processes that are active, it is possible to determine the actions that can be played and evolve the network to a new state. As for moving from one state to another it can play only one action then each step (T) is characterized by a single change in a single sort. The rule (**e2**) offer a set of the possible changes "$\{activeFromTo(B,J,K,T)\}$". This predicate means that in the sort B the active process change from number J to number K at time step T. This change is possible only if the action is playable at this step "$playableAction(A,I,B,J,K,T)$" meaning that processes (A,I) and (B,J) are active at T "$instate(activeProcess(A,I),T)$". The rule is encoded with a count atom at its head, which makes it a choice rule. Rule (**e3**) filters any answer with more than 1 change at the same time T.

e2: $\{activeFromTo(B,J,K,T)\} \leftarrow playableAction(A,I,B,J,K,T), instate(activeProcess(A,I),T), instate(activeProcess(B,J),T), J! = K, time(T)$.

e3: $\leftarrow 2\{activeFromTo(B,J,K,T)\}, time(T)$.

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

e4: $instate(activeProcess(B,K),T+1) \leftarrow activeFromTo(B,J,K,T), time(T)$.

e5: $instate(activeProcess(A,I),T+1) \leftarrow instate(activeProcess(A,I),T), activeFromTo(B,J,K,T), A! = B, time(T)$.

At the next step $T+1$ we find the new active process resulted from the predicate activeFromTo (**e4**) as well as all the unchanged processes that correspond to the other sorts (**e5**).

4.2 Reachability

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 process is 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 :
c1: $goal(activeProcess("a",1))$.
The rule c2 verifies if after the network evolution, its state satisfies the goals at step T. Else the answer will be eliminated.
c2: $satisfiable(F,T) \leftarrow goal(F), instate(F,T)$.
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 [2]). 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 contraint (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.
c3: $notSatisfiable(t) \leftarrow goal(F), notinstate(F,t)$.
c4: $\leftarrow notSatisfiable(t)$.

4.3 Results on real biological network

It should be noted that the only reachability analysis developed so far on the Process Hitting 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.
Example. In the example of Figure 1, at the beginning its state is <a_0, b_0, c_1> and we want to reach the process b_2. So our goal will be written $goal(activeProcess("b",2))$. PINT will retur the response "*True*", our ASP implementation, in addition to being satisfiable, returns the paths for two steps (0 and 1):

```
Answer:  1 activeFromTo("c",1,0,0) activeFromTo("b",0,2,1)
```

In the example of ERBB_G with 42 sorts, if we initialise the sorts at levels that model biological components then we fix the level for one sort to came our goal. The Table below describes the results:

Model	#sorts	#procs	#actions	#states	#steps	PINT	ASP	ASP iterative
ERBB_G	42	152	399	2^{70}	18	0.022s	10.620s	5.020s

5 Conclusion

We summarized in this paper a new developed dynamic analysis. This analysis is applicable to a class of models called Process Hitting and it aims at determining both the fixed points and if a condition on several components in the model can be attained from a given initial state. We think that this approach can be used also with other models such as Petri Nets.

Results show that, compared with Pint, method for fixed points search is is effective too but for reachability, it is much less than expected. However at the same time it gives more search results (the path) and it offers the possibility to ask more general questions covering several kinds of sorts.

Our perspective is to try to improve this method by eliminating the cycles in the iterative method (ICLINGO). Then we wish to extend the program to search for the attractors (a set of states from which it is not possible to get out).

References

[1] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

[2] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo, 2008.

[3] L. Paulevé. *Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique*. PhD thesis, École centrale de nantes, 2011.

[4] A. 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.