COMUE L'UNAM
École Doctorale STIM
Sciences et Technologies de
l'Information et Mathématiques

Spécialité : Informatique
Laboratoire : Institut de Recherche en Communication et Cybernitique de Nantes
Equipe : Formal Methods for Bioinformatics

Exhaustive checking of dynamical properties in multi-valued Biological Regulatory Networks using Answer Set Programming

Emna Ben Abdallah
Mél : emna.ben-abdallah@irccyn.ec-nantes.fr

Abstract: The combination of numerous simple influences between the components of a Biological Regulatory Network (BRN) often leads to behaviors that cannot be grasped intuitively. They thus call for the development of proper mathematical methods to delineate their dynamical properties. As a consequence, formal methods and computer tools for the modeling and simulation of BRNs become essential. Our recently introduced discrete formalism called the Process Hitting (PH), a restriction of synchronous automata networks, is notably suitable to such study. In this paper, we propose a new logical approach to perform model-checking of dynamical properties of BRNs 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 simulations verifying the dynamical properties thanks to the use of Answer Set Programming. The merits of our methods are 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, that is, up to tens of components and interactions.

Keywords: *Process Hitting, multi-valued Biological Regulatory Networks, Answer Set Programming, logic programming , stable states, fixed points, reachability*
Collaborations : National Institute of Informatics, Tokyo, Japan.

1 Preliminary definitions

1.1 Process Hitting

The definition1 introduces the Process Hitting (PH) [1] 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 $a_i \to b_j \nearrow b_k$, 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: $a_i \to a_i \nearrow a_k$.

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 $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$;*
- $\mathcal{H} = \{a_i \to b_j \nearrow b_k \in \mathcal{L}_a \times \mathcal{L}_b^2 \mid (a, b) \in \Sigma^2 \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ *is the finite set of actions.*

Example 1 *Figure 1 represents a $\mathcal{PH}$ $(\Sigma, \mathcal{L}, \mathcal{H})$ with three sorts ($\Sigma = \{a, b, c\}$) and: $\mathcal{L}_a = \{a_0, a_1\}$, $\mathcal{L}_b = \{b_0, b_1\}$, $\mathcal{L}_z = \{z_0, z_1, z_2\}$.*

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 $s[a] = a_i$.

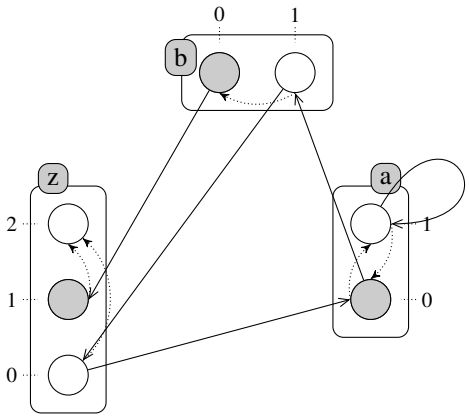

Figure 1: A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0 or 1 and z at either level 0, 1 or 2). Boxes represent the sorts (network components), circles represent the processes (component levels), and the 5 actions that model the dynamic behavior are depicted by pairs of arrows in solid and dotted lines. The grayed processes stand for the possible initial state: $\langle a_1, b_0, z_1 \rangle$.

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 $h = a_i \to b_j \nrightarrow b_k \in \mathcal{H}$ 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 $\forall c \in \Sigma, c \neq b \Rightarrow (s \cdot h)[c] = s[c]$.*

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, which were already formalized and tackled by other methods in [1, 2]. 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.

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, s is a fixed point if an only if no action is playable in this state:*

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

A finer and more interesting dynamical property consists in the notion of reachability property (definition 4). Such a property states that starting from a given initial state, it is possible to reach an other state. 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:*

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

2 Fixed point enumeration

2.1 Process Hitting translation in ASP

Before analyzing the dynamics of the network, we first need to translate the concerned PH network in ASP [1].

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 *Representation of a PH model of figure 1 in ASP*

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

```
1 process("a", 0..1). process("b", 0..1). process("z", 0..2).
2 sort(X) ← process(X,I).
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).
```

In line 1 we create the list of processes corresponding to each sort, for example the sort `"z"` *has 3 processes numbered from* `0` *to* `2`*; this specific predicate will in fact expand into the three following predicates:* `process("z", 0)`, `process("z", 1)`, `process("z", 2)`. *Line 2 enumerates every sort of the network from the previous information. Finally, all the actions of the network are defined in lines 3-4; for example, the first predicate* `action("a",0,"b",1,0)` *represents the action* $a_0 \to b_1 \curvearrowright b_0$.

2.2 Search of fixed points

The enumeration of fixed points requires first the elimination of all processes involved in a self-hit; the others are recorded by the predicate `shownProcess`:

```
5 hiddenProcess(A,I) ← action(A,I,B,J,K), A=B, process(A,I),process(B,J), process(B,K).
6 shownProcess(A,I) ← not hiddenProcess(A,I), 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 one process from each sort:

```
7 1 { selectedProcess(A,I) : shownProcess(A,I) } 1 ← sort(A).
```

Finally, the last step consists of filtering any state that is not a fixed point. For this, we use a constraint:

```
8 ← hit(A,I,B,J), selectedProcess(A, I), selectedProcess(B, J), A != B.
```

Example 3 (Fixed points enumeration) *The PH model of figure 1 contains 3 sorts:* a *and* b *have 2 processes and* z *has 3; therefore, the whole model has* $2 * 2 * 3 = 12$ *states. We can check that this model contains only one fixed point:* $\langle b_0, z_2, a_0 \rangle$. *If we execute the ASP program detailed below, we obtain exactly one answer set as well:*

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

3 Dynamical analysis

3.1 Future states identification

The predicate `time(0..n)` which sets the number of steps we want to play. The value of `n` can be arbitrarily chosen; In order to specify an initial state, we add several facts of this form "`init(activeProcess("a",0)).`" where `"a"` is the name of the sort and `"0"` the index of the active process. Identifying the future states requires to first identify the playable actions for each state. Therefore, we define an ASP predicate `playableAction(A,I,B,J,K,T)` that is true when the processes $\texttt{A}_\texttt{I}$ and $\texttt{B}_\texttt{J}$ are active at step `T`. 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 `activeFromTo(B,J,K,T)` which means that in sort `B`, the active process changes from index $\texttt{B}_\texttt{J}$ to $\texttt{B}_\texttt{K}$ between steps `T` and `T+1`.

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

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

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

3.2 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?" For this, we first use a predicate to list the processes we want to reach, called `goal`, and we add as many rules of the following form as there are objective processes: "`goal(activeProcess("a",1)).`" Then, the literal `reached(F, T)` checks if a given process `F` of the goal is contained in the state of step `T`, as defined in the rule of line 16. Else the answer will be eliminated by a constraint (line 20) which verifies if all processes of the goal are satisfied.

```
16 reached(F,T) ← goal(F), instate(F,T).
17 getNbreGoals(X) ← X={ goal(_) }.
18 reachedAllGoals(T) ← X={ reached(F,T) : goal(F) }, getNbreGoals(Y), X=Y, time(T).
19 reachedAllGoals ← reachedAllGoals(T).
20 ← not reachedAllGoals.
```

4 Conclusion

We proposed a new logical approach to address some dynamical properties of Process Hitting models that was published in [11]. The originality of our work consists in using ASP, a powerful declarative programming paradigm. Thanks to the encoding we introduced, we are not only able to tackle the enumeration of fixed points but also to check reachability properties. The major benefit of such a method is to get an exhaustive enumeration of all corresponding paths while still being tractable for models with dozens of interacting components.

References

[1] Loïc Paulevé, Morgan Magnin, and Olivier Roux. Refining dynamics of gene regulatory networks in a stochastic π-calculus framework. In Transactions on Computational Systems Biology XIII, volume 6575 of Lecture Notes in Comp Sci, pages 171–191. Springer, 2011.

[2] Loïc Paulevé, Morgan Magnin, and Olivier Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. Mathematical Structures in Computer Science, 22(04):651–685, 2012.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Emna Ben Abdallah, Maxime Folschette, Olivier Roux, and Morgan Magnin. Exhaustive analysis of dynamical properties of biological regulatory networks with answer set programming. In Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on, pages 281–285. IEEE, 2015.