Efficient analysis on very large models

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.

Maxime.Folschette@irccyn.ec-nantes.fr

Joint work with: Loïc Paulevé, Morgan Magnin, Olivier Roux

Abstract

The Process Hitting is a recently introduced framework to model concurrent processes. It models a finite set of components gathering several local states, with a particular form for the actions. In this paper, we define the reachability problem that aims at deciding if, starting from a given initial state, it is possible to reach a given local state. We also explain the static analysis method that was developed to answer to such problems in polynomial time, instead of the exponential complexity of the usual model checkers that apply a brute force approach. Our method thus answers in hundredths of a second on models with hundreds of components, but at the price of being sometimes inconclusive.

1 Introduction

Creating a coherent model from an existing system is a very challenging task, but analyzing may turn out to be even harder. If we focus on discrete modeling, this kind of problems is recurrent. In biology, for example, it is rather common to build models using a series of local experiment in order to determine the interactions between several components of the system (genes, proteins, metabolites...). To achieve this, experimenters proceed to gene knockouts (that is, they prevent some genes to express and stop the production of the related protein) and then use microarrays to measure the concentration of each other proteins. The result of such *in vivo* studies is a model that can gather up to hundreds of components, but several models of this kind can be also combined into a more general pathway, which can contain thousands of components.

The problem of being able to efficiently study these models immediately emerges. Formal model checkers usually have to compute the whole dynamics of the model in order to analyze and confront it to good properties, often expressed in temporal logics. But the computing of a state graph (even partial) is of exponential complexity, thus preventing the application of these methods to large models. Some results allow to statically derive some results given the structure of the model. For example, the presence of some circuits in the regulations between genes is a necessary condition for interesting behaviors such as oscillations [1]. Although such methods can be of interest, they do not allow precise analysis of the dynamics in order to validate the functionality of a model or predict *in vivo* behaviors.

In this paper we present an efficient method to compute a specific problem of reachability, that consists of checking if, starting from a given initial state of the system, a given component can reach a certain local state. This question is interesting to determine if, for instance, given a configuration of inputs, a given output can be activated or not. The analysis we present here is applied to a specific restriction of synchronous automata networks, called Process Hitting [2]. The purpose of this formalism is to model a finite set of local levels, contained into a finite set of components; actions are used to

change the local level of a component, and can be triggered by at most one other local level of another component. Although originally designed for large biological networks, this formalism is general enough to represent any discrete system. The particular form of its actions allows an efficient analysis of the dynamics [3, 4] that we present in the following, which permits to answer reachability questions in polynomial time. The main drawback of this approach is the possibility to be inconclusive although this case did not occur on the tested examples.

2 The Process Hitting formalism

2.1 Definition

Definition 1 introduces the Process Hitting (PH) [2] which allows to model a finite number of local levels, called *processes*, grouped into a finite set of components, called *sorts*. A process is written a_i, where a is the sort's name, and i is the process identifier within the 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 $a_i \to b_j \nearrow b_k$, 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 $(\Sigma, L, \mathscr{H})$:

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

The set of all processes is noted: $\mathbf{Proc} = \{a_i \in L_a \mid a \in \Sigma\}$. Given a state $s \in L$, the process of sort $a \in \Sigma$ present in s is denoted by $s[a]$. An action $h = a_i \to b_j \nearrow b_k \in \mathscr{H}$ is *playable* in s if and only if $s[a] = a_i$ and $s[b] = b_j$. In such a case, $(s \cdot h)$ stands for the state resulting from the play of the action h in s, with $(s \cdot h)[b] = b_k$ and $\forall c \in \Sigma, c \neq b, (s \cdot h)[c] = s[c]$. A sequence of actions $\delta = (h_1, \dots, h_{|\delta|})$ that are successively playable in s is called a *scenario in* s, and we note: $(s \cdot \delta) = s \cdot h_1 \cdot \ldots \cdot h_{|\delta|}$. We denote by $\mathbf{Sce}(s)$ the set of scenarios in s.

Example. Figure 1 represents a PH $(\Sigma, L, \mathscr{H})$ with four sorts ($\Sigma = \{a, b, c, d\}$) and: $L_a = \{a_0, a_1\}$, $L_b = \{b_0, b_1, b_2\}$, $L_c = \{c_0, c_1\}$ and $L_d = \{d_0, d_1, d_2\}$.

2.2 The reachability problem

In this paper, we focus on the *reachability of a process* (Definition 2), 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?"

Definition 2 (Reachability question)**.** If $\varsigma \in L$ is a state and $a_i \in \mathbf{Proc}$ is a process, we note $P(\varsigma, a_i)$ the *reachability question*: "Is there a scenario in ς so that, if we play all the actions of this scenario starting from ς, a_i is active in the resulting state?"; that is, in formal terms:

$$P(\varsigma, a_i) \equiv \exists ?\delta \in \mathbf{Sce}(\varsigma), (\varsigma \cdot \delta)[a] = a_i \ .$$

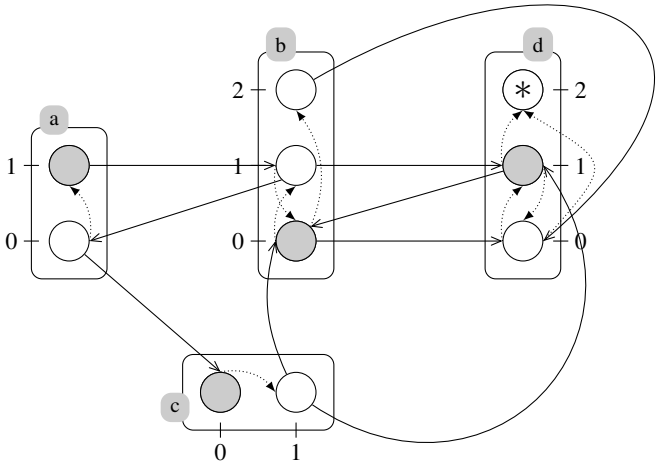

Figure 1: A PH model example with four sorts. Circles represent the processes, boxes represent the sorts, and the actions are drawn by pairs of arrows in solid and dotted lines. The grayed processes and the asterisk indicate an example of reachability problem $P(\langle a_1, b_0, c_0, d_1\rangle, d_2)$ as explained in Section 2.2.

In fact, this question can be extended in several manners that will not be tackled here. The original paper deals with the question of the *successive reachability of a sequence of processes* [3], in which several processes are required to be active successively, which is useful, for example, to state that a component can oscillate. Another interesting question is the *simultaneous reachability of several processes*; the addition of priorities in the PH [4], in addition to increasing the expressivity of the formalism, allows to easily answer this question. Finally, we note that the static analysis developed in the following sections can also be to some extent generalized to more synchronous automata networks [5].

2.3 Overview of the solution

The method developed here relies on the approximation of the dynamics of a PH model, instead of computing the exact dynamics, which requires an exponential time and memory usage in the size of the model. Thus, the static analysis we propose only focuses on local dynamics of the model, that is to say, only computes the dynamics of single sorts each time it is necessary. This approximation is based on the fact that an action $a_i \to b_j \nearrow b_k$ can only be triggered by (at most) a process a_i of another sort (in the case where $a_i \neq b_j$). For example, solving $P(\varsigma, b_k)$ would simply require to recursively solve $P(\varsigma, a_i)$ if $\varsigma[b] = b_j$. In the general case, several actions are needed but the idea is the same. This drastically drops the complexity of the method to an almost polynomial complexity in the number of sorts in the whole model, allowing to answer reachability questions in less than a second on models with hundreds or thousands of sorts.

However, an approximation does not come without drawbacks. Our method relies on two approximations: the *under-approximation* Q is a sufficient condition to answer "Yes" to the reachability question, and the *over-approximation* R is a necessary condition, used to answer "No". Indeed, instead of directly checking a given reachability question $P(\varsigma, a_i)$, we check Q and R, which is easier as explained above, and use the fact that $Q \Rightarrow P(\varsigma, a_i) \Rightarrow R$. But in the case where Q is false and R is true, the answer turns out to be "Inconclusive", and classical model checking techniques have to be used. Nevertheless, our method has always been conclusive on the examples tested in [3].

3 Graphs of local causality

The approximations developed in [3, 4] rely on the construction of graphs called *graphs of local causality* (GLCs), that permit approximating the dynamics of a PH model. One graph is dedicated to the under-approximation, while the other focuses on the over-approximation. We present an intuitive definition of these graphs in the following, and illustrate it with the PH model of Figure 1 and the reachability question $P(\varsigma, d_2)$ where $\varsigma = \langle a_1, b_0, c_0, d_1 \rangle$.

3.1 Nodes

GLCs contain process nodes, objective nodes and solution nodes as detailed below.

- Process nodes (in **Proc**) allow to state that the reachability of a given process is needed.

For example, for the reachability question $P(\varsigma, d_2)$, the process d_2 will naturally be a node of the graph.

Objectives have the form: $b_j \Rsh^* b_k$, which stands for the reachability of b_k from a state where b_j is present; it is thereby a refinement of a possible process node b_k in the GLC, by focusing on the reachability of a process inside a sort from a given initial process. We denote the set of all possible objectives as $\mathbf{Obj} = \{b_j \Rsh^* b_k \in L_b \times L_b \mid b \in \Sigma\}$. An objective of the form $b_j \Rsh^* b_j$ is called *trivial* because no action is needed to solve is.

- Objective nodes (in **Obj**) thus permit to name the source and destination processes of a local reachability.

For example, $d_1 \Rsh^* d_2$ denotes the reachability of d_2 from the initial process of d in ς, which is d_1.

Solving an objective $b_j \Rsh^* b_k$ implies to find a set of actions that locally solves it. This can be done by analyzing all bounces and targets of actions on the sort b. and computing the paths without loops that lead from b_j to b_k. Then, if a sequence of actions solving $b_j \Rsh^* b_k$ is found, we abstract it by extracting only the hitters of the actions because they are the only requirement to play the said actions. Such an abstracted set of hitters is called a *solution*, and we denote $\mathbf{Sol} = \wp(\mathbf{Proc})$ the set of all possible solutions. A solution indeed abstracts a sequence of actions because only the hitters are kept and the order of the actions is forgotten (as sets are not ordered). The solution for trivial objectives, or objectives that require only self-hits, is the empty set.

- Solution nodes (in **Sol**) are sets of processes that are required to locally solve an objective.

For example, $\{c_1, b_2\}$ and $\{b_1\}$ are the two minimal solutions of the objective $d_1 \Rsh^* d_2$ in Figure 1.

3.2 Edges

Edges in a GLC link the different nodes to their requirements. The GLC of the over-approximation is recursively built the following way:

- A process node b_k is linked to the objective of the form: $(\varsigma[b]) \Rsh^* b_k$;
- An objective node is linked to all the minimal solutions that solve it;
- A solution is linked to all the processes it contains.

If a solution is the empty set, then it obviously has no successor, which terminates the recursion.

The under-approximation GLC is built in a similar fashion, but a process may be linked to several additional objective nodes, in order to become sufficient. Indeed, if two processes of a sort b, say, b_j and

b_k are needed to solve two different objectives, then we have to ensure that b_j and b_k are both reachable from the initial state process $\varsigma[b]$, but also that b_j is reachable from b_k and b_k is reachable from b_j. Thus, new objectives $b_j \Rsh^* b_k$ and $b_k \Rsh^* b_j$ are added. This step is executed iteratively until a fixed point is reached.

We note that the complexity of building these graphs is polynomial in the number of sorts and exponential in the number of processes in each sort visited (due to the local solving of objectives in order to find solutions). However, if the model comprises few processes in every sort (i.e. less that four, which is usual in biological models), then our method can be considered almost polynomial in the number of sorts, which is very efficient compared to other solutions.

Example. Figure 2 gives two examples of GLCs, one for the over-approximation (top) and another for the under-approximation (bottom), both computed for two different reachability questions on the PH model of Figure 1.

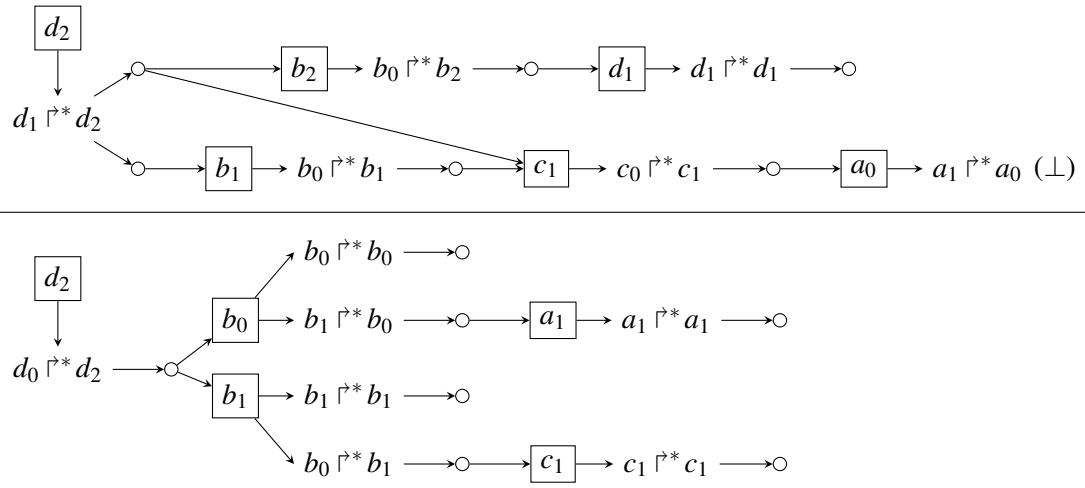

Figure 2: (top) Over-approximation GLC for $P(\langle a_1, b_0, c_0, d_1\rangle, d_2)$; (bottom) Under-approximation GLC for $P(\langle a_1, b_1, c_1, d_0\rangle, d_2)$; both computed on the PH of Figure 1. Process nodes are in boxes, solution nodes are the small circles (and are not detailed) and objective nodes are the nodes with no border.

4 Necessary and sufficient conditions

Finally, the two computed GLCs can be used in order to conclude on the initial reachability question $P(\varsigma, a_i)$ (Theorem 3 and Theorem 4). Checking these theorems is of polynomial complexity in the number of nodes they contain, which is limited by the size of the model.

Theorem 3 (Over-approximation)**.** *If, starting from the process a_i in the over-approximation GLC, there exists no path with no loop such that, from an objective node, exactly one linked solution node is traversed and, from any other node, all linked nodes are traversed, then $P(\varsigma, a_i)$ is false.*

Theorem 4 (Under-approximation)**.** *If the under-approximation GLC has no cycle and all its leaves are solution nodes, then $P(\varsigma, a_i)$ is true.*

Example. By applying Theorem 3 on Figure 2(top), we can conclude that $P(\langle a_1, b_0, c_0, d_1\rangle, d_2)$ is wrong because all paths lead to $a_1 \Rsh^* a_0$ which has no solution. Conversely, with Theorem 4 we can conclude that $P(\langle a_1, b_1, c_1, d_0\rangle, d_2)$ is true because all leafs of on Figure 2(bottom) are solutions.

5 Application to large models

The method presented here was implemented into the Pint[1] library which gathers tools dedicated to PH, and applied on four biological models as detailed in [3]. In a nutshell, all models studied have between 42 and 193 sorts, amongst them what we call *inputs* (resp. *outputs*), that is, sorts whose processes are not the target (resp. the hitter) of any actions. For all the possible combinations of the inputs (that can be either "active" or "inactive") the reachability of an "active" state for each output was computed.

The implementation could answer in less than a tenth of a second for each reachability question. These computation times were compared to the classical model checkers Biocham [6] and libDDD [7], for which the solving of the same reachability questions took at least 1 second and in many cases did not terminate due to lack of memory. Furthermore, our method was always conclusive, that is, Theorem 3 or Theorem 4 could always be used in order to conclude, thus leading to no inconclusive analysis. Theses examples prove that the approximations developed in the previous sections are very efficient and allow an unprecedented study of very large models.

6 Conclusion

We summarized in this paper the static analysis developed in [3] and later extended in [4]. This analysis is applicable to a class of models called Process Hitting that is a restriction of synchronous automata networks, and aims at determining if a local state of a component in the model can be attained from a given initial state. It consists in approximating the dynamics of the whole model by considering local reachabilities (objectives) and sets of local state requirements (solutions) to solve them. It turns out to be very efficient by making tractable the study of models with up to hundreds of components, while classical model checkers take much more time of fail due to memory limitations.

References

[1] A. Richard and J.-P. Comet, "Necessary conditions for multistationarity in discrete dynamical systems," *Discrete Applied Mathematics*, vol. 155, no. 18, pp. 2403 – 2413, 2007.

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

[3] L. Paulevé, M. Magnin, and O. Roux, "Static analysis of biological regulatory networks dynamics using abstract interpretation," *Mathematical Structures in Computer Science*, vol. 22, no. 04, pp. 651–685, 2012.

[4] M. Folschette, L. Paulevé, M. Magnin, and O. Roux, "Under-approximation of reachability in multivalued asynchronous networks," *Electronic Notes in Theoretical Computer Science*, vol. 299, pp. 33 – 51, 2013. 4th International Workshop on Interactions between Computer Science and Biology (CS2Bio'13).

[5] L. Paulevé, G. Andrieux, and H. Koeppl, "Under-approximating cut sets for reachability in large scale automata networks," in *Computer Aided Verification* (N. Sharygina and H. Veith, eds.), vol. 8044 of *Lecture Notes in Computer Science*, pp. 69–84, Springer Berlin Heidelberg, 2013.

[6] F. Fages and S. Soliman, "Formal cell biology in Biocham," in *Formal Methods for Computational Systems Biology*, pp. 54–80, Springer, 2008.

[7] J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier, "Data decision diagrams for Petri net analysis," in *Application and Theory of Petri Nets 2002*, vol. 2360 of *Lecture Notes in Computer Science*, pp. 101–120, Springer, 2002.

[1] The source of Pint and the examples mentioned here are available at: `http://loicpauleve.name/pint/`