HAL
open science

Identification of Bifurcations in Biological Regulatory Networks using Answer-Set Programming

Louis Fippo Fitime, Olivier Roux, Carito Guziolowski, Loïc Paulevé

To cite this version:

Louis Fippo Fitime, Olivier Roux, Carito Guziolowski, Loïc Paulevé. Identification of Bifurcations in Biological Regulatory Networks using Answer-Set Programming. Constraint-Based Methods for Bioinformatics Workshop, Sep 2016, Toulouse, France. hal-01361350

HAL Id: hal-01361350

https://hal.science/hal-01361350v1

Submitted on 7 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Identification of Bifurcations in Biological Regulatory Networks using Answer-Set Programming

Louis Fippo Fitime[1], Olivier Roux[1], Carito Guziolowski[1]§, and Loïc Paulevé[2]§

[1] LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597
(Institut de Recherche en Communications et Cybernétique de Nantes)
1 rue de la Noë – B.P. 92101 – 44321 Nantes Cedex 3, France.
[2] LRI UMR CNRS 8623, Univ. Paris-Sud – CNRS,
Université Paris-Saclay, 91405 Orsay, France

Abstract Aiming at assessing differentiation processes in complex dynamical systems, this paper focuses on the identification of states and transitions that are crucial for preserving or pre-empting the reachability of a given behaviour. In the context of non-deterministic automata networks, we propose a static identification of so-called bifurcations, i.e., transitions after which a given goal is no longer reachable. Such transitions are naturally good candidates for controlling the occurrence of the goal, notably by modulating their propensity. Our method combines Answer-Set Programming with static analysis of reachability properties to provide an under-approximation of all the existing bifurcations. We illustrate our discrete bifurcation analysis on several models of biological systems, for which we identify transitions which impact the reachability of given long-term behaviour. In particular, we apply our implementation on a regulatory network among hundreds of biological species, supporting the scalability of our approach.

1 Introduction

The emerging complexity of dynamics of biological networks, and in particular of signalling and gene regulatory networks, is mainly driven by the interactions between the species, and the numerous feedback circuits it generates [43,31,35,30]. One of the prominent and fascinating features of cells is their capability to differentiate: starting from a multi-potent state (for instance, a stem cell), cellular processes progressively confine the cell dynamics in a narrow state space, an attractor. Deciphering those decisional processes is a tremendous challenge, with important applications in cell reprogramming and regenerative medicine.

Discrete models of network dynamics, such as Boolean and multi-valued networks [42,4], have been designed with such an ambition. These frameworks model nodes of the network by variables with small discrete domains, typically

§ Corresponding authors: carito.guziolowski@irccyn.ec-nantes.fr, loic.pauleve@lri.fr

Boolean. Their value changes over time according to the state of their parent nodes. Exploring the dynamical properties of those computational models, such as reachability (the ability to evolve to a particular state) or attractors (long-run behaviors), allows to understand part of important cellular processes [36,1,5].

In classical control theory the reachability and safety control of discrete systems is an important topic which is both critical and challenging. Safety verification or reachability analysis aims at showing that starting from some initial conditions a system cannot evolve towards to some unsafe region in the state space. In a stochastic setting the different trajectories originating from one initial state have a different likelihood and one can then evaluate what is the probability that the system reaches the assigned set of states starting from a given initial distribution of the set of initial states. In safety problems, where the evolution of the system can be influenced by some control input, one should select it appropriately so as to minimize the probability that the state of the system will move to an unsafe set of states. In this work we address the computation of sets of states from which the system can evolve into an undesired set of states given a model represented as a discrete finite-state of interacting components, such as an Automata Network. This topic is in particular highly relevant for systems biology and systems medicine, since it can suggest intervention sites or therapeutic targets in Biological Regulatory Networks (BRNs) that may counteract pathological behavior.

Contributions. In this work we introduce the notion of *bifurcation* in Automata Networks (ANs) and provide a scalable method for their identification relying on declarative programming with Answer-Set Programming (ASP) [3]. A bifurcation corresponds to a transition after which the system looses the capability to reach a given goal state. Identifying bifurcations extensively relies on the reachability problem, which is PSPACE-complete in ANs and related frameworks [8]. In order to obtain an approach tractable on large biological networks, we show how to combine techniques from static analysis of ANs dynamics, from concurrency, and from constraint programming in order to relax efficiently the bifurcation problem. Our method identifies correct bifurcations only (no false positives), but, due to the embedded approximations, is incomplete (false negatives may exist). To our knowledge, this is the first integrated method to extract such decisive transitions from models of interaction networks.

Related work. Many works have been devoted to the control theory and system verification; in particular in reachability analysis and safety control for system verification. We limit here to those developed for discrete systems.

In the case of deterministic systems, a large number of methods have been proposed to aid in the verification of safety-critical computer software and hardware. Due to the inherently discrete nature of these systems, early attempts in this area have concentrated on purely discrete systems [6]. A verification procedure has been proposed for discrete event dynamic systems in [25]. They use a finite state machine representation to model discrete event dynamic systems, temporal logic statements (to represent specifications imposed on their operations) and a model-checking verification method introduced by Clarke et al. [10]

to test if the given relationship between the model and the specifications holds. The same method has been applied to the verification of programmable logic controllers [27] and control procedures for batch plants [26].

For systems with complex behavior, the development of modeling frameworks and verification methods has been done among others by [20]. Because of the significant increase of the state space, many approximation methods were proposed for reachability computation and can be grouped in two main approaches. The first one is based on an approximate simulation relation to obtain an abstraction of the original system [19]. In the second approach, static analysis methods inspired from Cousot et al. [11] analyze the system dynamic and cope with the state-space explosion. In this group we can cite the development of static analysis of reachability properties based on an over- and under-approximation using the Process Hitting [30] and extended for the ANs. Besides methods that approximated the computation of reachability, authors in [33,34] proposed methods for safety verification that do not require the computation of reachable sets, but instead relied on the notion of barrier certificates [32].

A key notion underlying network behavior is that state-space is organized into a number of basins of attraction, connecting states according to their transitions, and summing up the network's global dynamics. For systems having many attractors, some of them may correspond to desired behaviors, while others, to unsafe regions. In the global network dynamic a system can move towards an unsafe region. Therefore, beyond ensuring the properties reachability and safety verification, proposing strategies to interfere on a system and force a desired behavior is important. Previous works introduced the concept of Minimal Intervention Sets (MISs) [23] for biological networks and later generalized in [37] for addressing Boolean models of signaling networks. This concept allowed one to choose an intervention strategy to provoke a desired/observed response in certain target nodes. Compared to the approach present of this paper, they do not take into account the transient dynamics of the system, which limits considerably the domain of predictions. [28,2] proposed approaches based on cut sets to identify nodes/reactions whose perturbation would prevent the occurrence of a given state/reaction. Whereas those prediction can help to control the reachability of an attractor, they do not allow to capture the differentiations processes, as do bifurcations.

Outline Section 2 introduces the Automata Network formalisms on which our method relies. The notion of bifurcation in ANs is introduced in section 3. In section 4, we give a brief overview of methods for checking reachability properties which is a core task for the characterization of bifurcations. In section 5, we combine static analysis, dynamics unfolding, and ASP, aiming at providing a scalable identification of the bifurcation transitions for a given goal. In section 6, we evaluate a prototype implementation on several large biological models in order to support the scalability of our approach. Section 7 concludes the paper.

2 Automata Networks

We consider an Automata Network (AN) as a finite set of finite-state machines having transitions between their local states conditioned by the state of other automata in the network. An AN is defined by a triple (Σ, S, T) (definition 1) where Σ is the set of automata identifiers; S associates to each automaton a finite set of local states: if $a \in \Sigma$, $S(a)$ refers to the set of local states of a; and T associates to each automaton the list of its local transitions. Each local state is written of the form a_i, where $a \in \Sigma$ is the automaton in which the state belongs to, and i is a unique identifier; therefore given $a_i, a_j \in S(a)$, $a_i = a_j$ if and only if a_i and a_j refer to the same local state of the automaton a. For each automaton $a \in \Sigma$, $T(a)$ refers to the set of transitions of the form $t = a_i \xrightarrow{\ell} a_j$ with $a_i, a_j \in S(a)$, $a_i \neq a_j$, and ℓ the enabling condition of t, formed by a (possibly empty) set of local states of automata different than a and containing at most one local state of each automaton.

Definition 1 (Automata Network (Σ, S, T)). *An* Automata Network *(AN) is defined by a tuple (Σ, S, T) where*

- *Σ is the finite set of automata identifiers;*
- *For each $a \in \Sigma$, $S(a) = \{a_i, \ldots, a_j\}$ is the finite set of local states of automaton a; $S \triangleq \prod_{a \in \Sigma} S(a)$ is the finite set of global states;*
  $\mathbf{LS} \triangleq \bigcup_{a \in \Sigma} S(a)$ denotes the set of all the local states.
- *$T = \{a \mapsto T_a \mid a \in \Sigma\}$, where $\forall a \in \Sigma, T_a \subseteq S(a) \times 2^{\mathbf{LS} \setminus S(a)} \times S(a)$ with $(a_i, \ell, a_j) \in T_a \Rightarrow a_i \neq a_j$ and $\forall b \in \Sigma, |\ell \cap S(b)| \leq 1$, is the mapping from automata to their finite set of local transitions.*

We write $a_i \xrightarrow{\ell} a_j \in T \overset{\Delta}{\Leftrightarrow} (a_i, \ell, a_j) \in T(a)$.

At any time, each automaton is in one and only one local state, forming the global state of the network. Assuming an arbitrary ordering between automata identifiers, the set of global states of the network is referred to as S as a shortcut for $\prod_{a \in \Sigma} S(a)$. Given a global state $s \in S$, $s(a)$ is the local state of automaton a in s, i.e., the a-th coordinate of s.

A local transition $t = a_i \xrightarrow{\ell} a_j \in T$ is applicable in a global state $s \in S$ when a_i and all the local states in ℓ are in s. The application of the local transition, noted $s \cdot t$, replaces the local state of a with a_j (definition 2). It results in a (global) transition $s \xrightarrow{t} s'$ where $s' = s \cdot t$. In this paper, we consider the *asynchronous* semantics of ANs: only one local transition can be applied at a time, meaning only one automaton changes its local state by the transitions between two global states. In this semantics, different local transitions may be applicable to a same state, which may potentially lead to very different dynamics. The choice of the transition is *non-deterministic*. A global state s' is reachable from s, noted $s \rightarrow^* s'$, if and only if there exists a (possibly empty) sequence of transitions leading from s to s'.

Definition 2 (Transition, reachabilility). *Given a state $s \in S$ and a local transition $t = a_i \xrightarrow{\ell} a_j \in T$ such that $s(a) = a_i$ and $\forall b_k \in \ell$, $s(b) = b_j$, $s \cdot t$ is the state s where a_i has been replaced by a_j:*

$$\forall b \in \Sigma, (s \cdot t)(b) = \begin{cases} a_j & \text{if } b = a \\ s(b) & \text{otherwise} \end{cases}$$

We then write $s \xrightarrow{t} s'$ where $s' = s \cdot t$. The reachability *binary relation $\rightarrow^* \subseteq S \times S$ satisfies*

$$s \rightarrow^* s' \overset{\Delta}{\Leftrightarrow} s = s' \vee \exists t \in T : s \xrightarrow{t} s'' \wedge s'' \rightarrow^* s'$$

Figure 1 represents an AN (Σ, S, T) of 3 automata ($\Sigma = \{a, b, c\}$), with $S(a) = \{a_0, a_1, a_2\}$, $S(b) = \{b_0, b_1\}$, $S(c) = \{c_0, c_1, c_2\}$, and 8 local transitions defined as follows:

$$T(a) = \{t_1 = a_1 \xrightarrow{\emptyset} a_0, t_2 = a_0 \xrightarrow{b_0} a_1, t_3 = a_0 \xrightarrow{b_0, c_0} a_2\}$$
$$T(b) = \{t_4 = b_0 \xrightarrow{\emptyset} b_1, t_5 = b_1 \xrightarrow{a_0} b_0\}$$
$$T(c) = \{t_6 = c_0 \xrightarrow{a_1} c_1, t_7 = c_1 \xrightarrow{b_1} c_0, t_8 = c_1 \xrightarrow{b_0} c_2\}$$

From the given initial state $s_0 = \langle a_0, b_0, c_0 \rangle$, 3 transitions can be applied: t_2, t_3, and t_4; the application of the latter results in $s_0 \cdot t_4 = \langle a_0, b_1, c_0 \rangle$ (automaton b is now in state b_1).

3 Bifurcations

From an initial state s_0 and a *goal* local state, we call a *bifurcation* a transition from a state where the goal is reachable to a state where the goal is not reachable, i.e., there exists no sequence of transition that lead to a state containing the goal local state.

Let us consider the AN of figure 1, with $s_0 = \langle a_0, b_0, c_0 \rangle$ and the goal a_2. Figure 2 shows all the possible transitions from s_0. The states with a gray background are connected to a state containing a_2 (in thick/blue). The transitions in thick/red are bifurcations: once in a white state, there exist no sequence of transitions leading to a_2. In other words, bifurcations are the transitions from a gray state to a white state. In this example, t_8 is the (unique) local transitions responsible for bifurcations from s_0 to a_2.

In this paper, given an AN (Σ, S, T), we are interested in identifying the local transitions $t_b \in T$ that trigger a bifurcation from a state reached from $s_0 \in S$ for a given goal, describing a set of states $S_g \subseteq S$. We call s_b a global state where a bifurcation occurs, and s_u the global state after the bifurcation: $s_u = s_b \cdot t_b$. The goal is reachable from s_b but not from s_u. This is illustrated by figure 3. Note that, as illustrated, s_b is not inevitably reached: we allow the existence of alternative paths of transitions to the goal.

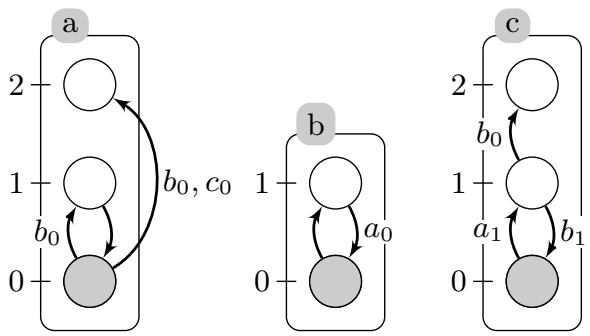

Figure 1. An example of Automata Network (AN).‘ Automata are represented by labelled boxes, and local states by circles where ticks are their identifier within the automaton – for instance, the local state a_0 is the circle ticked 0 in the box a. A transition is a directed edge between two local states within the same automaton. It can be labelled with a set of local states of other automata. Grayed local states stand for the global state $\langle a_0, b_0, c_0 \rangle$.

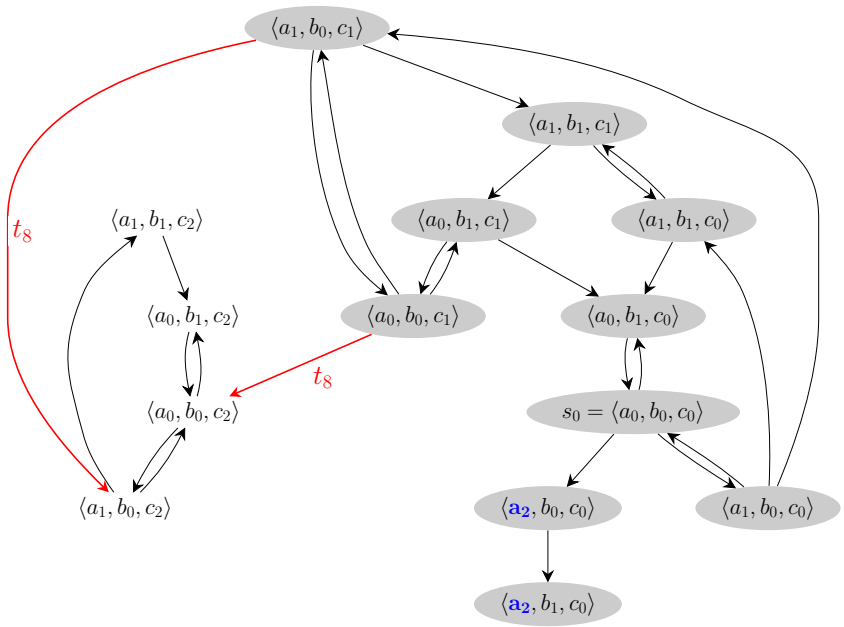

Figure 2. Transition graph of the AN in figure 1 from the initial state $s_0 = \langle a_0, b_0, c_0 \rangle$. The goal a_2 is in thick/blue; the states connected to the goal are in gray; the bifurcations to the goal in thick/red, labelled with the local transitions in the AN definition.

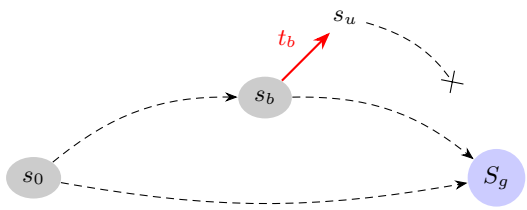

Figure 3. General illustration of a bifurcation. s_0 is the initial state, S_g is a set of states in which the goal local state is present. The dashed arrows represent a sequence (possibly empty) of transitions. The plain red arrow is a bifurcation from a global state s_b to s_u, and t_b is the associated local transition.

Definition 3 formalizes the notion of bifurcation, where the goal is specified by a local state g_1 (hence $S_g = \{s \in S \mid s(g) = 1\}$). Note that this goal specification does not loose generality, as one can build an automaton g with local states g_0 and g_1, and with a local transitions from g_0 to g_1 conditioned by each desired goal state.

Definition 3 (Bifurcation). *Given an AN (Σ, S, T), a global state $s_0 \in S$ and a goal local state g_1 with $g \in \Sigma$ and $g_1 \in S(g)$, a* bifurcation *is a transition $s_b \xrightarrow{t_b} s_u$ of the AN with $s_b, s_u \in S$ and $t_b \in T$, such that $s_0 \rightarrow^* s_b$ and $\forall s' \in S$ where $s_u \rightarrow^* s'$, $s'(g) \neq g_1$.*

4 Reachability and formal approximations

The identification of a bifurcation as presented in the previous section can be decomposed in three steps: we want to find $s_b \in S$ and $t_b \in T$ such that (1) s_b is reachable from s_0 ($s_0 \rightarrow^* s_b$); (2) g_1 is reachable from s_b ($\exists s_g \in S : s_g(g) = g_1 \wedge s_b \rightarrow^* s_g$); (2) g_1 is not reachable from $s_u = s_b \cdot t_b$ ($\forall s \in S, s_u \rightarrow^* s \Rightarrow s(g) \neq g_1$). Therefore, alongside the enumeration of candidate s_b and t_b, reachability is at the core of our bifurcation identification.

In this section, we give a brief overview of the basics of reachability checking, stressing the methods we use in this paper.

4.1 State graph and partial order reductions

Given two states s, s' of an AN (or an equivalent Petri net), verifying $s \rightarrow^* s'$ is a PSPACE-complete problem [8].

The common approach for reachability checking is to build the (finite) set of all the states reachable from s until finding s', by exploring all the possible transitions. However, such a set can be rapidly intractable with large models. Techniques relying on symbolic representations, notably using Binary Decision Diagrams (BDDs) and variants [21] can improve the scalability of this approach by several orders of magnitude [9].

In many cases, numerous transitions modelled by ANs are *concurrent*: their application is independent from each other. For instance, if t_1 and t_2 are concurrent in a state s, one can apply indifferently $s \cdot t_1 \cdot t_2$ and $s \cdot t_2 \cdot t_1$. Such features can be exploited to provide very compact representations of the reachable states in a concurrent system, taking into account the partial order of transition applications. Unfoldings, and more precisely their complete finite prefixes [13], allow to compute efficiently such compact representations.

In this paper, part of our method uses complete finite prefixes of unfoldings to compute the states that are reachable from the fixed initial state s_0. Indeed, because biological networks are typically very large, but also very sparse (each node/automaton interacts with a few others, compared to the size of the network), they exhibit a high degree of concurrency for their transitions, making unfolding approaches very effective.

4.2 Formal approximations

When facing a large AN, it may turn out that the reachable state space is too large for the aforementioned exact verification of reachability. Moreover, the complexity of the reachability problem can be prohibitive when numerous verifications have to be done, for instance when enumerating candidate initial states (such as s_b in our case for checking goal reachability from it).

In this paper, we rely on the reachability approximations for ANs introduced in [29,15]. We will use both *over-approximations* (OA) and *under-approximations* (UA) of the reachability problem: $s \rightarrow^* s'$ is true only if $\mathsf{OA}(s \rightarrow^* s')$ is true and $s \rightarrow^* s'$ is true if $\mathsf{UA}(s \rightarrow^* s')$ is true; but the converses do not hold in general:

$$\mathsf{UA}(s \rightarrow^* s') \Rightarrow s \rightarrow^* s' \Rightarrow \mathsf{OA}(s \rightarrow^* s')$$

The approximations rely on static analysis by abstract interpretation of AN dynamics. We give here the basic explanations for the over- and under-approximation. The analysis rely on the decomposition of the systems dynamics in automata, in order to derive necessary or sufficient conditions for a reachability property of the form $s \rightarrow^* s'$.

The core objects are the *local paths* within two local states a_i, a_j of a same automaton a. We call $a_i \rightsquigarrow a_j$ an *objective* and define $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$ the set of the acyclic paths of local transitions between a_i and a_j. Definition 4 gives the formalization of $\mathsf{local\text{-}paths}$ where we use the following notations: for a local transition $t = a_i \xrightarrow{\ell} a_j \in T$, $\mathsf{orig}(t) \stackrel{\Delta}{=} a_i$, $\mathsf{dest}(t) \stackrel{\Delta}{=} a_j$, $\mathsf{enab}(t) \stackrel{\Delta}{=} \ell$; ε denotes the empty sequence, and $|\eta|$ is the length of sequence η.

Definition 4 ($\mathsf{local\text{-}paths}$)**.** *Given an objective $a_i \rightsquigarrow a_j$,*

- *if $i = j$, $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_i) \stackrel{\Delta}{=} \{\varepsilon\}$;*
- *if $i \neq j$, a sequence η of transitions in $T(a)$ is in $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$ if and only if $\mathsf{orig}(\eta^1) = a_i$, $\mathsf{dest}(\eta^{|\eta|}) = a_j$, $\forall n, 1 \leq n < |\eta|$, $\mathsf{dest}(\eta^n) = \mathsf{orig}(\eta^{n+1})$, and $\forall n, m, |\eta| \geq n > m \geq 1, \mathsf{dest}(\eta^n) \neq \mathsf{orig}(\eta^m)$.*

We write $t \in \eta \overset{\Delta}{\Leftrightarrow} \exists n, 1 \leq n \leq |\eta| : \eta_n = t$. Given a local path η, $\widetilde{\eta}$ denotes the union of the conditions of all the local transitions composing it:

$$\widetilde{\eta} \overset{\Delta}{=} \bigcup_{n=1}^{|\eta|} \mathsf{enab}(\eta_n)$$

In the AN of figure 1, $\mathsf{local\text{-}paths}(a_0 \rightsquigarrow a_2) = \{a_0 \xrightarrow{b_0, c_0} a_2\}$; $\mathsf{local\text{-}paths}(c_2 \rightsquigarrow c_1) = \emptyset$.

Focusing on the reachability of a single local state g_1 from a state s where $s(g) = 0$, the analyzes essentially start with the local paths in $\mathsf{local\text{-}paths}(g_0 \rightsquigarrow g_1)$: if g_1 is reachable, then at least of the local path η has to be realizable, meaning that all the local states of its conditions ($\widetilde{\eta}$) should be reachable. This lead to a recursive reasoning by repeating the procedure with the objectives from s to the local states in $\widetilde{\eta}$.

The dependence relationships between the local paths of the different automata can be represented as a graph, where the nodes are all the local states, all the possible objectives, and all their local paths. Such a graph is called a *Local Causality Graph* (LCG), and abstracts all the executions of the AN.

From a complexity point of view, local paths are computed for each pair of local states within every automata; the length of a local path being at most the number of local states within the automaton, the number of local paths is at most polynomial in the number of local transitions and exponential in the size of the single automaton. In practice, the automata are small, typically between 2 and 4 states for biological models. Therefore, LCGs turn out to be very small compared to the reachable state space of biological networks. They have been successfully applied for analyzing dynamics of ANs with hundreds or thousands of automata, which were intractable with standard model-checking approaches [29,28].

The over-approximation and under-approximation reduce to finding subgraphs of LCGs that satisfy some particular structural properties, which have been proven to be necessary or sufficient for the reachability property, respectively. Whereas the over-approximation can be verified in a time linear with the LCG [29], the under-approximation we consider in this paper requires to enumerate over many possible sub-LCGs, but checking if a sub-LCG satisfies the sufficient condition is linear in its size.

Note that further refinements of $\mathsf{local\text{-}paths}$ have been considered for the mentioned approximations, but for the sake of simplicity, we stick to this coarse-grained presentation in the scope of this paper.

Appendix A gives examples of LCGs satisfying necessary or sufficient conditions for reachability properties in the AN of figure 1.

5 Identification of bifurcations using ASP

Among the states reachable from s_0, we want to find a state s_b from which (1) the goal is reachable and (2) there exists a transition to a state from which the goal is not reachable. Putting aside the complexity of reachabilities checking, the

enumeration of candidate states s_b is a clear bottleneck for the identification of bifurcations in an AN.

Our approach combines the formal approximations and (optionally) unfoldings introduced in the previous section with a constraint programming approach to efficiently identify bifurcations. As discussed in the previous section, checking the over-/under-approximations from candidate states and sub-LCGs is easy. For the case of unfolding, checking if a state s belongs to the state space represented by a complete finite prefix is NP-complete [14]. Therefore, a declarative approach such as Answer-Set Programming (ASP) [3] is very well suited for specifying admissible s_b and t_b, and obtaining efficient enumerations of solutions by a solver.

We first present the general scheme of our method, and then given details on its implementation with ASP.

5.1 General scheme

A sound and complete characterization of the local transitions $t_b \in T$ triggering a bifurcation from state s_0 to the goal g_1 would be the following: t_b is a bifurcation transition if and only if there exists a state $s_b \in S$ such that

$$\text{(C1)}\ \ s_u \not\rightarrow^* g_1 \qquad \text{(C2)}\ \ s_b \rightarrow^* g_1 \qquad \text{(C3)}\ \ s_0 \rightarrow^* s_b$$

where $s_u = s_b \cdot t_b$, $s \not\rightarrow^* g_1 \overset{\Delta}{\Leftrightarrow} \forall s' \in S, s \rightarrow^* s' \Rightarrow s'(g) \neq g_1$ and $s \rightarrow^* g_1 \overset{\Delta}{\Leftrightarrow} \exists s_g \in S : s_g(g) = g_1 \wedge s \rightarrow^* s_g$.

However, in an enumeration scheme for s_b candidates, checking reachability and non-reachability of the goal from each s_b candidate ((C1) and (C2)) is prohibitive. Instead, we relax the above constraints as follows:

$$\text{(I1}^{\#}\text{)}\ \ \neg\mathsf{OA}(s_u \rightarrow^* g_1) \qquad \text{(I2}^{\#}\text{)}\ \ \mathsf{UA}(s_b \rightarrow^* g_1) \qquad \begin{array}{l} \text{(I3)}\ \ s_b \in \mathsf{unf\text{-}prefix}(s_0) \\ \text{(I3}^{\#}\text{)}\ \ \mathsf{UA}(s_0 \rightarrow^* s_b) \end{array}$$

where $\mathsf{unf\text{-}prefix}(s_0)$ is the set of all reachable states from s_0 represented as the prefix of the unfolding of the AN which has to be pre-computed (section 4.1). Either (I3) or (I3$^{\#}$) can be used, at discretion. From OA and UA properties (section 4.2), we directly obtain:

$$\text{(I1}^{\#}\text{)} \Rightarrow \text{(C1)} \qquad \text{(I2}^{\#}\text{)} \Rightarrow \text{(C2)} \qquad \begin{array}{l} \text{(I3)} \Leftrightarrow \text{(C3)} \\ \text{(I3}^{\#}\text{)} \Rightarrow \text{(C3)} \end{array}$$

Therefore, our characterization is sound (no false positive) but incomplete: some t_b might be missed (false negatives). Using (I3) instead of (I3$^{\#}$) potentially reduces the false negatives, at the condition that the prefix of the unfolding is tractable. When facing a model too large for the unfolding approach, we should rely on (I3$^{\#}$) which is much more scalable but may lead to more false negatives.

Relying on the unfolding from s_b ($\mathsf{unf\text{-}prefix}(s_b)$) is not considered here, as it would require to compute a prefix from each s_b candidate, whereas $\mathsf{unf\text{-}prefix}(s_0)$ is computed only once before the bifurcation identification.

5.2 ASP syntax and semantics

We give a very brief overview of ASP syntax and semantics that we use in the next section. Please refer to [18,3,16] for an in-depth introduction to ASP.

An ASP program consists in a set of predicates, and logic rules of the form:

```
1 a0 ← a1, ..., an, not an+1, ..., not an+k.
```

where a_i are atoms, terms or predicates in first order logic.

Essentially, such a logical rules states that when all $a_1, \ldots, a_n$ are true and all $a_{n+1}, \ldots, a_{n+k}$ are false, then a_0 has to be true as well. a_0 can be $\perp$ (or simply omitted): in such a case, the rule is satisfied only if the right hand side of the rule is false (at least one of $a_1, \ldots, a_n$ is false or at least one of $a_{n+1}, \ldots, a_{n+k}$ is true). A solution consists in a set of true atoms/terms/predicates with which all the logical rules are satisfied.

ASP allows to use variables (starting with an uppercase) instead of terms/predicates: these *pattern* declarations will be expanded to the corresponding propositional logic rules prior to the solving. For instance, the following ASP program has as unique (minimal) solution `b(1) b(2) c(1) c(2).`

```
2 c(X) ← b(X).
3 b(1).
4 b(2).
```

In the following, we also use the notation n `{ a(X) : b(X) }` m which is true when at least n and at most m `a(X)` are true where `X` ranges over the true `b(X)`.

5.3 ASP implementation

We present here the main rules for implementing the identification of bifurcation transitions with ASP. A significant part of ASP declarations used by (I1$^{\#}$), (I2$^{\#}$), (I3), and (I3$^{\#}$) are generated from the prior computation of local-paths and, in the case of (I3), of the prefix of the unfolding. Applied on figure 1, our implementation correctly uncovers t_8 as a bifurcation for a_2.

Declaration of local states, transitions, and states Every local state $a_i \in S(a)$ of each automaton $a \in \Sigma$ are declared with the predicate `ls(`a`,`i`)`. We declare the local transitions of the AN and their associated conditions by the predicates `tr(`id`,`a`,`i`,`j`)` and `trcond(`id`,`b`,`k`)`, which correspond to the local transition $a_i \xrightarrow{\{b_k\} \cup \ell} a_j \in T$. States are declared with the predicate `s(ID,A,I)` where `ID` is the state identifier, and `A`, `I`, the automaton and local state present in that state. Finally, the goal g_1 is declared with `goal(`g`,1)`.

For instance, the following instructions declare the automaton a of figure 1 with its local transitions, the state $s_0 = \langle a_0, b_0, c_0 \rangle$, and the goal being a_2:

```
1 ls(a,0). ls(a,1). ls(a,2).
2 tr(1,a,1,0).
3 tr(2,a,0,1). trcond(2,b,0).
4 tr(3,a,0,2). trcond(3,b,0). trcond(3,c,0).
5 s(0,a,0). s(0,b,0). s(0,c,0).        goal(a,2).
```

Declaration of s_b, t_b, and s_u The bifurcation transition t_b, declared as `btr(b)`, is selected among the declared transitions identifiers (line 6). If $a_i \xrightarrow{\ell} a_j$ is the selected transition, the global state s_u (recall that $s_u = s_b \cdot t_b$) should satisfy $s_u(a) = a_j$ (line 7) and, $\forall b_k \in \ell$, $s_u(b) = b_k$ (line 8). The state s_b should then match s_u, except for the automaton a, as $s_b(a) = a_i$ (lines 9 and 10).

```
6  1 { btr(ID) : tr(ID,_,_,_) } 1.
7  s(u,A,J) ← btr(ID),tr(ID,A,_,J).
8  s(u,B,K) ← btr(ID),trcond(ID,B,K).
9  s(b,A,I) ← btr(ID),tr(ID,A,I,_).
10 s(b,B,K) ← s(u,B,K),btr(ID),tr(ID,A,_,_),B!=A.
```

(I1#) declaration of $\neg$OA($s_u \rightarrow^* g_1$) This part aims at finding the state s_u from which g_1 is not reachable. For that, we designed an ASP implementation of the reachability over-approximation presented in section 4.2. It consists in building a Local Causality Graph (LCG) from pre-computed local-paths. A predicate `oa_valid(G,ls(A,I))` is then defined upon G to be true when the local state a_i is reachable from the initial state s_G. The full implementation is given in appendix B.1.

We instantiate a LCG u with the initial state s_u by declaring that the goal is not reachable (`oa_valid`) (line 11).

```
11 ← oa_valid(u,ls(G,I)),goal(G,I).
```

(I2#) declaration of UA($s_b \rightarrow^* g_1$) This part aims at finding the state s_b from which g_1 is reachable. Our designed ASP implementation of the reachability under-approximation (section 4.2) consists in finding a sub-LCG G with the satisfying properties for proving the sufficient condition. The edges of this sub-LCG are declared with the predicate `ua_lcg(G,Parent,Child)`. The graph is parameterized by (1) a *context* which specifies a set of possible initial states and (2) an edge from the node `root` to the local state(s) for which the simultaneous reachability has to be decided from the supplied context. The full implementation is given in appendix B.2.

We instantiate the under-approximation for building a state s_b from which the goal g_1 is reachable: g_1 is a child of the `root` node of graph b (line 12). The context is subject to the same constraints as s_b from s_u (lines 13 and 14 reflect lines 9 and 10). Then, s_b defines one local state per automaton among the context from which the reachability of g_1 is ensured (line 15), and according to lines 9 and 10.

```
12 ua_lcg(b,root,ls(G,I)) ← goal(G,I).
13 ctx(b,A,I) ← btr(ID),tr(ID,A,I,_).
14 ctx(b,B,K) ← s(u,B,K),btr(ID),tr(ID,A,_,_),B!=A.
15 1 { s(b,A,I) : ctx(b,A,I) } 1 ← ctx(b,A,_).
```

(I3) declaration of $s_b \in$ unf-prefix(s_0) Given a prefix of an unfolding from s_0 (section 4.1), checking if s_b is reachable from s_0 is an NP-complete problem [14] which can be efficiently encoded in SAT [22] (and hence in ASP). A synthetic description of the ASP implementation of reachability in unfoldings is given in appendix C. The interested reader should refer to [13]. Our encoding provides a predicate `reach(`a`,`i`)` which is true if a reachable state contains a_i. Declaring s_b reachable from s_0 is done simply as follows:

```
16 reach(A,I) ← s(b,A,I).
```

(I3$^{\#}$) declaration of UA$(s_0 \rightarrow^* s_b)$ An alternative to (I3) which does not require to compute a complete prefix of the unfolding is to rely on the under-approximation of reachability similarly to (I2$^{\#}$). The under-approximation is instantiated for the reachability of s_b from s_0 with the following statements:

```
17 ua_lcg(0,root,ls(A,I)) ← s(b,A,I).
18 ctx(0,A,I) ← s(0,A,I).
```

6 Experiments

We evaluate our method for the computation of bifurcation transitions in models of actual biological networks showing differentiation capabilities. We have selected three networks showing at least two attractors reachable from a same initial state. In these cases, by supplying an adequate goal state representing one of the attractor, we expect to identify transitions causing a bifurcation from this attractor. We start by introducing our three case studies.

Immunity control in bacteriophage lambda (Lambda phage) . A number of bacterial and viral genes take part in the decision between lysis and lysogenization in temperate bacteriophages. In the lambda case, at least five viral genes (refered to as cI, cro, cII, N and cIII) and several bacterial genes are involved. We apply our method on an automata network equivalent to the model introduced in [41] and with two different goals, corresponding to lysis or lysogenization.

Epidermal Growth Factor & Tumor Necrosis Factor$_\alpha$ (EGF/TNF) is a model that combines two important mammalian signaling pathways induced by the Epidermal Growth Factor (EGF) and Tumor Necrosis Factor alpha (TNF$_\alpha$) [24,7]. EGF and TNF$_\alpha$ ligands stimulate ERK, JNK and p38 MAPK cascades, the PI3K/AKT pathways, and the NFkB cascade. This network encompasses cross-talks between these pathways, as well as two negative feedback loops.

T-helper cell plasticity (Th) has been studied in [1] in order to investigate switches between attractors subsequent to changes of input conditions. It is a cellular network regulating the differentiation of T-helper (Th) cells, which orchestrate many physiological and pathological immune responses. T-helper (CD4+) lymphocytes play a key role in the regulation of the immune response. By APC activation, native CD4 T cells (Th0) differentiate into specific Th subtypes producing different cytokines which influence the activity of immune effector cell

Automata Network	Goal	(I3)			(I3$^{\#}$)		M-C	
		$\lvert\mathsf{pfx}\rvert$	$\lvert t_b\rvert$	Time	$\lvert t_b\rvert$	Time	$\lvert t_b\rvert$	Time
Lambda phage	CI_2	45	6	$0.1s$	0	$0.2s$	10	$0.1s$
$\lvert\Sigma\rvert = 4 \quad \lvert T\rvert = 11$	Cro_2		3	$0.1s$	2	$0.3s$	3	$0.1s$
EGF/TNF	$NFkB_0$	52	4	$0.1s$	2	$0.1s$	5	$0.2s$
$\lvert\Sigma\rvert = 28 \quad \lvert T\rvert = 55$	IKB_1		3	$0.1s$	2	$0.1s$	5	$0.2s$
Th_th1	$BCL6_1$	444	6	$16s$	5	$23s$	8	$13s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$TBET_1$		5	$10s$	4	$24s$	11	$14s$
Th_th2	$GATA3_0$	3264	7	$24s$	4	$24s$	8	$60s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$BCL6_1$		5	$25s$	4	$25s$	7	$600s$
Th_th17	$RORGT_1$	2860	9	$23s$	8	$26s$	18	$48s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$BCL6_1$		5	$23s$	4	$24s$	7	$26s$
Th_HTG	$BCL6_1$	OT			6	$61s$	OT	
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$GATA3_1$				7	$34s$		

Table 1. Experimental results for the identification of bifurcations depending if (I3) or (I3$^{\#}$) is used, compared to a exact model-checking (M-C) using NuSMV[9]. Models Th_th1, Th_th2, Th_th17, Th_HTG are the same automata network but have different initial state. For each model, two different goals have been tested. $|\Sigma|$ is the number of automata, and $|T|$ the number of transitions; $|\mathsf{pfx}|$ is the size (number of events in the partial order structure) of the prefix of the unfolding from the initial state of the model; $|t_b|$ is the number of identified bifurcation transitions. Computation times have been obtained on an Intel® Core™ i7-4770 3.40GHz CPU with 16GiB of RAM. OT indicates an out-of-time execution (more than one hour).

types. Several subtypes (Th1, Th2, Th17, Treg, Tfh, Th9, and Th22) have been well established. We report in this paper four (Pro Th1, Pro Th2, Pro Th17, Pro HTG(Th1 + Th17)) different initial state from which different attractors are reachable. The network is composed of 101 nodes and 221 interactions.

Method. For each selected model with initial state and goal, we performed the bifurcation identification following either (I1$^{\#}$), (I2$^{\#}$), (I3) (unfolding from s_0); or (I1$^{\#}$), (I2$^{\#}$), (I3$^{\#}$) (reachability under-approximation). We use clingo 4.5.3 [17] as ASP solver, and Mole [38] for the computation of the unfolding ((I3)). The computation times correspond to the total toolchain duration, and includes the local-paths computation, unfolding, ASP program generation, ASP program loading and grounding, and solving. Note that the local-paths computation (and ASP program generation) is almost instantaneous for each case. Source code and models are provided in the supplementary material file.

Results. Table 1 summarizes the results of the experiments. The last experiment shows the limit of the exact analysis of the reachable state space: the

computation of the prefix is not tractable on this model. However, the alternative approach (I3$^{\#}$) allows to identify bifurcation transitions in this large model. Following section 5.1, (I3$^{\#}$) always results in less bifurcations transitions than (I3) with our models. It can be explained with the additional approximation for the reachability of s_b from s_0, using the notations of section 3. One can finally remark that when (I3) is tractable, (I3$^{\#}$) shows a slightly slower solving time, albeit of the same order of magnitude. This suggest that checking if a state belongs to an unfolding is more efficient than checking its under-approximation. Finally, because there exists to our knowledge no other method for identifying bifurcations, we cannot compare our results with different methods, and in particular with an exact method in order to appreciate the false negative rate obtained by the (I1$^{\#}$)-(I2$^{\#}$)-(I3) scheme.

7 Conclusion

This paper presents an original combination of computational techniques to identify transitions of a dynamical system that can remove its capability to reach a (set of) states of interest. Our methodology combines static analysis of Automata Networks (ANs) dynamics, partial order representations of the state space, and constraint programming to efficiently enumerate those bifurcations. To our knowledge, this is the first intregated approach for deriving bifurcation transitions from concurrent models, and ANs in particular.

Bifurcations are key features of biological networks, as they model decisive transitions which control the *differentiation* of the cell: the bifurcations decide the portions of the state space (no longer) reachable in the long-run dynamics. Providing automatic methods for capturing those differentiations steps is of great interest for biological challenges such as cell reprogramming [12,1], as they suggest targets for modulating undergoing cellular processes.

Given an initial state of the AN and a goal state, our method first computes static abstractions of the AN dynamics and (optionnaly) a symbolic representation of the reachable state space with so-called unfoldings. From those prior computations, a set of constraints are issued to identify bifurcation transitions. We used Answer-Set Programming to declare the admissible solutions and the solver clingo to obtain their efficient enumerations. For large models, the unfolding may be intractable: in such a case, the methods relies only on reachability over- and under-approximations. By relying on those relaxations which can be efficiently encoded in ASP, our approach avoids costly exact checking, and is tractable on large models, as supported by the experiments.

Further work will consider the complete identification of bifurcation transitions, by allowing false positives (but no false negatives). In combination with the under-approximation of the bifurcations presented in this paper, it will provide an efficient way to delineate *all* the transitions that control the reachability of the goal attractor. Future work will also focus on exploiting the identified bifurcations for driving estimations of the probability of reaching the goal at steady state, in the scope of hybrid models of biological networks [39,40].

References

1. W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaudon, V. Soumelis, C. Chaouiya, and D. Thieffry. Model checking to assess t-helper cell plasticity. *Frontiers in Bioengineering and Biotechnology*, 2(86), 2015.
2. V. Acuna, F. Chierichetti, V. Lacroix, A. Marchetti-Spaccamela, M.-F. Sagot, and L. Stougie. Modes and cuts in metabolic networks: Complexity and algorithms. *Biosystems*, 95(1):51–60, 2009.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
4. G. Bernot, F. Cassez, J.-P. Comet, F. Delaplace, C. Müller, and O. Roux. Semantics of biological regulatory networks. *Electronic Notes in Theoretical Computer Science*, 180(3):3 – 14, 2007.
5. L. Calzone, E. Barillot, and A. Zinovyev. Predicting genetic interactions from boolean models of biological networks. *Integr. Biol.*, 7(8):921–929, 2015.
6. C. G. Cassandras. *Discrete event systems: modeling and performance analysis*. CRC, 1993.
7. C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, M. P. van Iersel, N. Rodriguez, A. Dräger, F. Büchel, T. Cokelaer, B. Kowal, B. Wicks, E. Gonçalves, J. Dorier, M. Page, P. T. Monteiro, A. von Kamp, I. Xenarios, H. de Jong, M. Hucka, S. Klamt, D. Thieffry, N. Le Novère, J. Saez-Rodriguez, and T. Helikar. SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Systems Biology*, 7(1):1–15, 2013.
8. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.
9. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin / Heidelberg, 2002.
10. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252, New York, NY, USA, 1977. ACM.
12. I. Crespo and A. del Sol. A general strategy for cellular reprogramming: The importance of transcription factor cross-repression. *STEM CELLS*, 31(10):2127–2135, Oct 2013.
13. J. Esparza and K. Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. Springer, 2008.
14. J. Esparza and C. Schröter. Unfolding based algorithms for the reachability problem. *Fund. Inf.*, 47(3-4):231–245, 2001.
15. M. Folschette, L. Paulevé, M. Magnin, and O. Roux. Sufficient conditions for reachability in automata networks with priorities. *Theoretical Computer Science*, 608, Part 1:66 – 83, 2015. From Computer Science to Biology and Back.
16. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

17. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Preliminary report. In M. Leuschel and T. Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, volume arXiv:1405.3694v1, 2014. Theory and Practice of Logic Programming, Online Supplement.
18. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.
19. A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *IFAC Analysis and Design of Hybrid Systems, Alghero, Italy*, 2006.
20. R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel. *Hybrid systems*. Springer-Verlag, 1993.
21. A. Hamez, Y. Thierry-Mieg, and F. Kordon. Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. *Fundam. Inf.*, 94(3-4):413–437, 2009.
22. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-Safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.
23. S. Klamt and E. D. Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.
24. A. MacNamara, C. Terfve, D. Henriques, B. P. Bernabé, and J. Saez-Rodriguez. State–time spectrum of signal transduction logic models. *Physical Biology*, 9(4):045003, 2012.
25. I. Moon. Automatic verification of discrete chemical process control systems. 1992.
26. I. Moon and S. Macchietto. Formal verification of batch processing control procedures. In *Proc. PSE'94*, page 469. 1994.
27. I. Moon, G. J. Powers, J. R. Burch, and E. M. Clarke. Automatic verification of sequential control systems using temporal logic. *AIChE Journal*, 38(1):67–75, 1992.
28. L. Paulevé, G. Andrieux, and H. Koeppl. Under-approximating cut sets for reachability in large scale automata networks. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, 2013.
29. L. Paulevé, M. Magnin, and O. Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. *Mathematical Structures in Computer Science*, 22(04):651–685, 2012.
30. L. Paulevé and A. Richard. Static analysis of boolean networks based on interaction graphs: a survey. *Electronic Notes in Theoretical Computer Science*, 284:93 – 104, 2011. Proceedings of The Second International Workshop on Static Analysis and Systems Biology (SASB 2011).
31. E. Plathe, T. Mestl, and S. Omholt. Feedback loops, stability and multistationarity in dynamical systems. *Journal of Biological Systems*, 3:569–577, 1995.
32. S. Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.
33. S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.
34. S. Prajna, A. Jadbabaie, and G. J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *Automatic Control, IEEE Transactions on*, 52(8):1415–1428, 2007.
35. A. Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378–392, 2010.

36. O. Sahin, H. Frohlich, C. Lobke, U. Korf, S. Burmester, M. Majety, J. Mattern, I. Schupp, C. Chaouiya, D. Thieffry, A. Poustka, S. Wiemann, T. Beissbarth, and D. Arlt. Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance. *BMC Systems Biology*, 3(1), 2009.
37. R. Samaga, A. V. Kamp, and S. Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. *Journal of Computational Biology*, 17(1):39–53, 2010.
38. S. Schwoon. Mole. `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`.
39. I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, feb 2002.
40. G. Stoll, E. Viara, E. Barillot, and L. Calzone. Continuous time boolean modeling for biological signaling: application of gillespie algorithm. *BMC Syst Biol*, 6(1):116, 2012.
41. D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks–ii. immunity control in bacteriophage lambda. *Bulletin of mathematical biology*, 57:277–97, 1995 Mar 1995.
42. R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973.
43. R. Thomas and R. d'Ari. *Biological Feedback*. CRC Press, 1990.

A Examples of Local Causality Graphs

Figure 4 gives examples of Local Causality Graphs (section 4.2) for approximation reachability of a_2 in the AN of figure 1. The left LCG does not satisfy the necessary condition (no local paths from c_2 to c_0), hence a_2 is not reachable from the given initial state$\langle a_1, b_0, c_2\rangle$. The middle LCG does satisfy the necessary condition. And the right LCG is a valid sub-LCG for the sufficient condition for a_2 reachability.

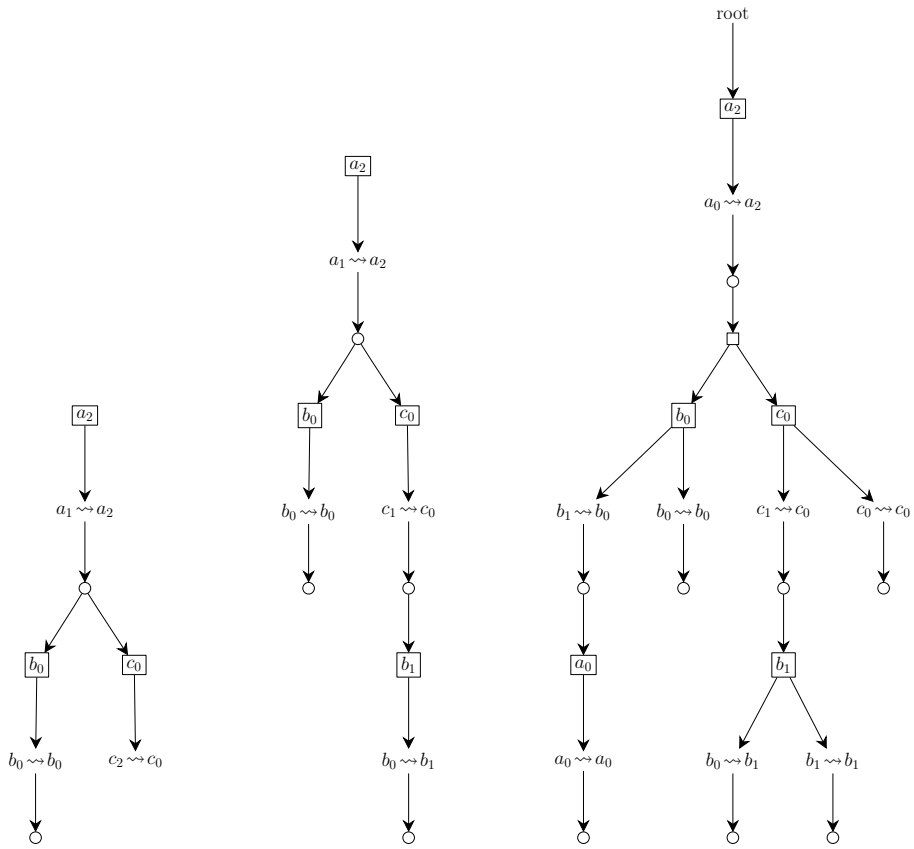

Figure 4. Exemples of Local Causality Graphs for (left) over-approximation of a_2 reachability from $\langle a_1, b_0, c_2\rangle$ (middle) over-approximation of a_2 reachability from $\langle a_1, b_0, c_1\rangle$ (right) under-approximation of a_2 reachability from $\langle a_0, b_1, c_1\rangle$. The small circles represent the local paths.

B ASP implementation of Over- and Under-approximation of Reachability

B.1 OA($s \rightarrow^* s'$): necessary condition for reachability

We propose here a possible encoding of the necessary condition for reachability in ANs outlined in section 4.2 and introduced in [29]. Starting from $s_u(g) = g_0$, the analysis starts with the *local paths* of the *objective* $g_0 \rightsquigarrow g_1$: g_1 is reachable only if all the conditions of the transitions of at least one local path $\eta \in \mathsf{local\text{-}paths}(g_0 \rightsquigarrow g_1)$ are reachable. This recursive reasoning can be modelled with a graph relating dependencies between objectives, local paths, and local states.

The local paths computed *a priori* are used to generate the template declaration of the directed edges of the LCG `oa_lcg(G,Parent,Child)` from each possible objective $a_i \rightsquigarrow a_j$. If $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \emptyset$, the objective $a_i \rightsquigarrow a_j$ is linked to a node `bottom`:

```
1 oa_lcg(G,obj(a,i,j),bottom) ← oa_lcg(G,_,obj(a,i,j)).
```

otherwise, for $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \{\eta^1, \ldots, \eta^n\}$, we declare a node `lpath` for each different local path $m \in \{1, \ldots, n\}$ as a child of $a_i \rightsquigarrow a_j$:

```
2 oa_lcg(G,obj(a,i,j),lpath(obj(a,i,j),m)) ← oa_lcg(G,_,obj(a,i,j)).
```

then, for each different local state $b_k \in \widetilde{\eta^m}$ in the conditions of the local transitions of η^m, we add an edge from the `lpath` node to `ls(b,k)`:

```
3 oa_lcg(G,lpath(obj(a,i,j),m),ls(b,k)) ← oa_lcg(G,_,obj(a,i,j)).
```

In the case when the local path requires no condition ($\widetilde{\eta^m} = \emptyset$, this can happen when the objective is trivial, i.e., $a_i \rightsquigarrow a_i$, or when the local transitions do not dependent on the other automata), we link the `lpath` to a node `top`:

```
4 oa_lcg(G,lpath(obj(a,i,j),m),top) ← oa_lcg(G,_,obj(a,i,j)).
```

A LCG G for over-approximation is parameterized with a state s_G: if a local path has a local state a_j in its transition conditions, the node `ls(a,j)` is linked, in G, to the node for the objective $a_i \rightsquigarrow a_j$ (line 5), with $a_i = s_G(a)$. It is therefore required that state s_G defines a (single) local state for each automaton referenced in G (line 6).

```
5 oa_lcg(G,ls(A,I),obj(A,J,I)) ← oa_lcg(G,_,ls(A,I)), s(G,A,J).
6 1 { s(G,A,J) : ls(A,J) } 1 ← oa_lcg(G, _, ls(A, _)).
```

The necessary condition for reachability is then declared using the predicate `oa_valid(G,N)` which is true if the node `N` satisfies the following condition: it is not `bottom` (line 7); and, in the case of a local state or objective node, one of its children is `oa_valid` (lines 8 and 9; or in the case of a local path, either `top` is its child, or all its children (local states) are `oa_valid` (lines 10 and 11).

```
7 ← oa_valid(G,bottom).
8 oa_valid(G,ls(A,I)) ← oa_lcg(G,ls(A,I),X),oa_valid(G,X).
9 oa_valid(G,obj(A,I,J)) ← oa_lcg(G,obj(A,I,J),X),oa_valid(G,X).
10 oa_valid(G,N) ← oa_lcg(G,N,top).
11 oa_valid(G,lpath(obj(a,i,j),m)) ← ⋀_{b_k ∈ η̃^m} oa_valid(G,ls(b,k)).
```

B.2 UA($s \to^* s'$): sufficient condition for reachability

We give here a declarative implementation of the sufficient condition for reachability in ANs outlined in section 4.2 and introduced in [15]. The under-approximation consists in building a graph relating objectives, local paths, and local states which satisfies several constraints. If such a graph exists, then the related reachability property is true. Similarly to (I1$^{\#}$), we give template declarations for the edges with the predicate `ua_lcg(G,Parent,Child)`. We assume that the reachability property is specified by adding an edge from `root` to `ls(`a`,`i`)` for each local state to reach.

The graph `ua_lcg` is parameterized with a *context* which is a set of local states, declared with the predicate `ctx(G,A,J)`. Every local states a_i of the graph that are not part of the reachability specification belong to that context (line 12); and are linked to the objective $a_j \rightsquigarrow a_i$ for each a_j in the context (line 13).

```
12 ctx(G,A,I) ← ua_lcg(G,N,ls(A,I)), N != root.
13 ua_lcg(G,ls(A,I),obj(A,J,I)) ← ua_lcg(G,_,ls(A,I)), ctx(G,A,J).
```

A first constraint is that each objective in the graph is linked to one and only one of its local path. Therefore, objectives without local paths ($\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \emptyset$) cannot be included (line 14), for the others, a choice has to be made among $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \{\eta^1, \ldots, \eta^n\}$ (line 15).

```
14 ← ua_lcg(G,_,obj(a,i,j)).
15 1 { ua_lcg(G,obj(a,i,j),lpath(obj(a,i,j),1..n)) } 1 ← ua_lcg(G,_,obj(a,i,j)).
```

As for `oa_lcg`, each local path is linked to all the local states composing its transition conditions: for each $m \in \{1, \ldots, n\}$, for each $b_k \in \widetilde{\eta^m}$,

```
16 ua_lcg(G,lpath(obj(a,i,j),m),ls(b,k)) ← ua_lcg(G,_,obj(a,i,j)).
```

The graph has to be acyclic. This is declared using a predicate `conn(G,X,Y)` which is true if the node `X` is connected (there is a directed path) to `Y` (line 17). A graph is cyclic when `conn(G,X,X)` (line 18).

```
17 conn(G,X,Y) ← ua_lcg(G,X,Y).  conn(G,X,Y) ← ua_lcg(G,X,Z), conn(G,Z,Y).
18 ← conn(G,X,X).
```

Then, if the node for an objective $a_i \rightsquigarrow a_j$ is connected to a local state a_k, the under-approximation requires $a_i \rightsquigarrow a_j$ to be connected with $a_k \rightsquigarrow a_j$ (assuming that a has at least 3 local states, definition not shown):

```
19 ua_lcg(G,obj(A,I,J),obj(A,K,J)) ← not boolean(A), conn(G,obj(A,I,J),ls(A,K)).
```

When a local transition is conditioned by at least two other automata (for instance $c_\circ \xrightarrow{a_i, b_j} c_\bullet$), the under-approximation requests that reaching b_j does not involve other local states from a others that a_i. This is stated by the `indep(G,Y,`a`,`i`,ls(`b`,`j`))` which cannot be true if b_j is connected to a local state a_k with $k \neq i$ line 20. Then, the under-approximation requires that at most one `indep` predicate is false, for a given LCG G and a given local path Y (line 21). Such an independence should also hold between the local states of the reachability specification (line 22).

```
20 indepfailure(Y,ls(A,I)) ← indep(G,Y,A,I,N), conn(G,N,ls(A,K)), K!=I.
21 ← indepfailure(Y,N),indepfailure(Y,M),M!=N.
22 indep(G,root,A,I,ls(B,J)) ← ua_lcg(G,root,ls(A,I)),ua_lcg(G,root,ls(B,J)),B != A.
```

For $\eta^m \in \mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$, for each local transition $a_\circ \xrightarrow{\ell} a_\bullet \in \eta^m$, for each couple of different local states in its condition $b_k, c_l \in \ell, b_k \neq c_l$:

```
23 indep(G,lpath(obj(a,i,j),m),b,k,ls(c,l)) ← ua_lcg(G,_,lpath(obj(a,i,j),m)).
```

C ASP implementation of reachability in unfoldings

A (prefix of an) unfolding is an acyclic bipartite digraph where nodes are either *events* (application of a transition) or *conditions* (change of local state) [13]. We use the predicate `post(X,Y)` to denote an edge from X to Y; and `h(C,ls(A,I))` to denote that the condition C corresponds to the local state a_i. Figure 5 shows an example of unfolding.

A state s belongs to the prefix if it is possible to build a *configuration* such that all the local states in s have a unique corresponding condition on the *cut* of the configuration (line 1).

A configuration is a set of events, and we use `e(E)` to denote that the event E belongs to the configuration. By definition, if E is in a configuration, all its parent events are in the configuration (line 2). There should be no *conflicts* between two events of a configuration: two events are in conflict if they share a common parent condition (line 3).

A condition is on the cut if its parent event is in the configuration (line 4), and none of its children event is in the configuration (line 5).

```
1 1 { cut(C)  : h(C,ls(A,I)) } 1 ← reach(A,I).
2 e(F) ← post(F,C),post(C,E),e(E).
3 ← post(C,E),post(C,F),e(E),e(F),E != F.
4 e(E) ← cut(C),post(E,C).
5 ← cut(C),post(C,E),e(E).
```

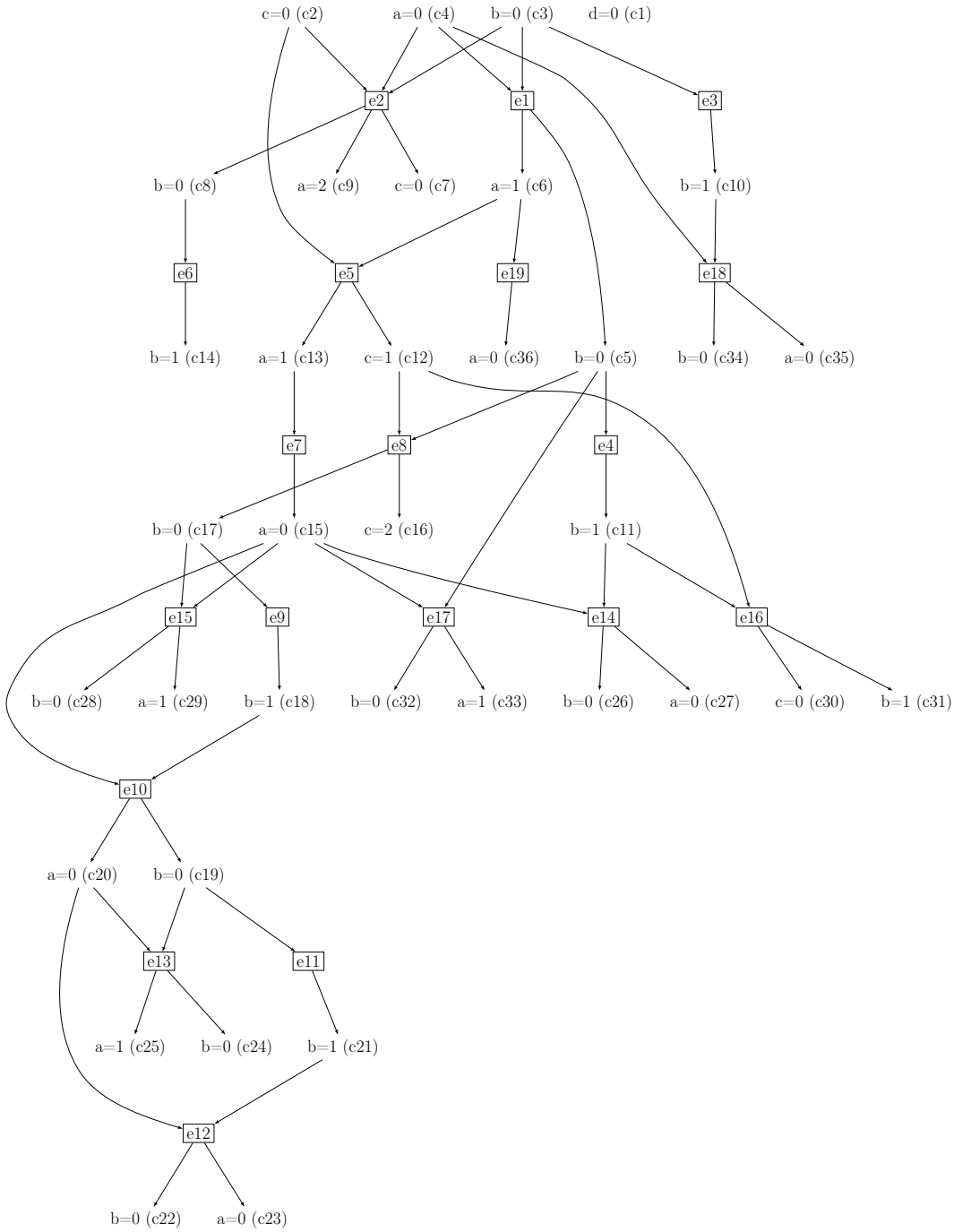

Figure 5. Unfolding of the AN of figure 1. Events are boxed nodes, conditions have no borders and indicate both the automata local state and the condition identifier.