Proving Unreachability in Automata Networks by Mixing Static Analysis and Bounded Model Checking

Samuel Buchet, Morgan Magnin, Olivier Roux

Proving Unreachability in Automata Networks by Mixing Static Analysis and Bounded Model Checking

Samuel Buchet, Morgan Magnin, Olivier Roux
LS2N, UMR CNRS 6004, École Centrale de Nantes,
1 rue de la Noë, 44321 Nantes, France
samuel.buchet@ec-nantes.fr

Abstract

Discrete models have been widely used in the field of bioinformatics, more especially in Systems Biology. Automata Network is a powerful modeling framework which can represent different dynamics on complex systems. The study of its dynamical properties has multiple purposes. Among these properties, reachability is of crucial interest for many real life applications (e.g. in biological regulatory networks). For example, it can give interesting insights about the behavior of the biological system. Additionally, it can be used to validate a model regarding the biological literature. In the last decade, some approaches have been shown of interest to address instances with hundreds of components. They are based on approximations that however lead to non-conclusiveness on some examples that may appear in practical biological examples. In this work, we present an approach combining static analysis with Bounded Model Checking to decide about the unreachable case. The originality of our method lies in the computation of a sufficient bound on the length of execution sequences, derived from previous works using static analysis.

Keywords— Static Analysis; Reachability; Bounded Model Checking; Discrete models; Automata Networks; Systems Biology;

1 Introduction

Biological systems such as genetic regulatory networks have been widely studied through discrete dynamical models. Indeed, representing biological species as discrete variables can give interesting insights in the dynamics of the system. Over the years, several discrete modeling frameworks have been proposed to capture and understand the behaviors of these networks. One benefit of studying these systems through discrete models is the possibility to perform formal analysis on models, since its state space is discrete and can usually be abstracted as a finite system.

In this paper, we focus on a discrete dynamical framework which can be seen as a generalization of Boolean Networks: Asynchronous Automata Networks (AAN) ([Folschette et al., 2015, Paulevé, 2018]). This qualitative approach is based on three key concepts: (1) Biological components (e.g. genes) are abstracted in the form of what we have called *automata*. The different local states (that are not restricted to Boolean values) of each automaton correspond to the different discrete qualitative levels of the components represented by the automaton. (2) Interaction between biological components is modeled in an atomic way by local transitions on the automata, conditioned by a set of required local states in different automata. (3) In the modeling task, such a representation makes it possible to build the largest possible dynamics, and then to proceed by successive refinements to restrict the possible behaviors. For this latter point, the asynchronous semantics seems suitable due to the large scope of behaviors that are covered. Although there are biological justifications for both the asynchronous and the synchronous semantics, the asynchronous semantics can be justified from the fact that the probability that several biological phenomena occur simultaneously is low. The flexibility of Asynchronous Automata Networks makes them useful to

represent various systems, including biological regulatory networks (which can also be modeled as Boolean Networks ([Taou et al., 2018])). The formal study of the dynamics of biological systems, especially the reachability problem, represent a main challenge because of its difficulty. In addition, reachability analysis is strongly linked to the computation of attractors, which is an important property of biological systems. Attractors are long-term structures in which any dynamics eventually falls into, thus capturing either the long-term behavior of a component or being interpreted as cell types ([Shen-Orr et al., 2010]). This kind of properties is also crucial when refining a model of a living system, that is one way to remove inconsistencies or to predict missing information in biological models consisting in comparing the reachabilities/unreachabilities of the model with the experimentally observed behaviors.

In recent years, Bounded Model Checking and especially SAT encoding have been shown as an efficient way to verify temporal properties on dynamical models ([Biere et al., 2009]). In this work, we investigate an approach based on static analysis which, combined with Bounded Model Checking, can be used to tackle reachability analysis in Asynchronous Automata Networks. More especially, this approach can be useful to prove the unreachability cases. To do so, we first introduce the Asynchronous Automata Networks - abbreviated hereafter as Automata Networks (AN) -. Second we describe the associated reachability problems and existing methods to tackle them. Third, we present in more details existing static analysis tools used for these problems. Finally, we introduce our main contribution, which is a new approach based on the computation of a completeness threshold (denominated bound) to prove unreachability. This bound represents a sufficient length for the sequences that solve reachability instances, and can be used as the length of encoded execution sequences to make unreachability proofs with a Bounded Model Checking approach. The computation of the bound relies on the existing static analysis approach, and we show that our new result strengthen the link between the existing properties based on static analysis and the set of solutions to a reachability problem, thus opening the door to further extensions. We also present the current limitations of our approach on specific instances, and we provide additional perspectives to its generalization.

Notations

Integer ranges are noted $[m; n] = \{m, m+1, ..., n\}$. $|A|$ represents the cardinality of a set A. A sequence of elements x is noted $x = (x^1 :: x^2 :: .. :: x^{|x|-1} :: x^{|x|})$ where $|x|$ denotes the length of the sequence. The empty sequence is denoted as ε, thus $|\varepsilon| = 0$.

2 Automata Networks and related works

In this section, the Asynchronous Automata Networks and their associated reachability problems are introduced. Then, multiple approaches and related works about the reachability problem are covered.

2.1 Asynchronous Automata Networks and reachability analysis

Automata Networks are discrete dynamical models which can be used to represent various real world systems. Their use has been focused on the study of biological networks, for example through the static analysis tool *Pint* ([Paulevé, 2017]), from the *Colomoto* framework ([Naldi et al., 2018]). Automata Networks are composed of several automata, linked together through their transitions. Each automaton has a limited number of local states and a global state of the model is a vector composed of local states for each automaton. Practically speaking when local states represent the discrete qualitative levels of a biological component, the number of such states is assumed to be relatively limited (often ≤ 3). Indeed, the benefits of discrete modeling is to make abstractions of the system so that its behavior is easier to analyse and interpret. A limited number of local states leads to simpler models that can be exhaustively analyzed.

Definition 1 *Automata Network (AN)*
An Automata Network is a tuple $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ *where:*

- $\Sigma = \{a, b, ..\}$ *is the finite set of automata*
- $\forall x \in \Sigma, \mathcal{S}_x = \{x_0, x_1, ...\}$ *is the finite set of local states of automaton* x
- $\mathcal{S} = \prod_{x \in \Sigma} \mathcal{S}_x$ *is the set of global states of* $\mathcal{M}$
- $\mathcal{T}_x \subset \{(x_i, x_j, l) \mid x_i, x_j \in \mathcal{S}_x, x_i \neq x_j, l \subset \bigcup_{y \in \Sigma \setminus \{x\}} \mathcal{S}_y \text{ and } |l \cap \mathcal{S}_y| \leq 1 \quad \forall y \in \Sigma \setminus \{x\}\}$ *is the finite set of transitions of automaton* $x \in \Sigma$
- $\mathcal{T} = \bigcup_{x \in \Sigma} \mathcal{T}_x$ *is the set of all transitions of the network*

For a global state $s \in \mathcal{S}$, the corresponding local state of an automaton x is given by $s(x)$. If $s(x) = x_i$, then x_i is said *active* for the state s and the states $\{x_j : x_j \neq x_i\}$ are said *inactive* for s. A transition $\tau \in \mathcal{T}_x$ is a tuple $(x_i, x_j, \{y_k, z_l, ...\})$ with x_i the origin state of the transition (denoted $orig(\tau)$), x_j the destination state of the transition (denoted $dest(\tau)$) and a set of local states $\{y_k, z_l, ...\}$ which are necessary to play the transition (denoted $enab(\tau)$). The notation $x_i \xrightarrow{y_k, z_l, ...} x_j$ is used for the transitions. In this work, self loop transitions (i.e. transitions of the form $x_i \rightarrow x_i$) are not considered. Indeed, transitions in Automata Networks aim at representing a modification in the dynamics of a biological system.

Example 1 *Fig. 1 shows an Automata Network with three automata. In this Automata Network,* $\Sigma = \{a, b, c\}$ *and* $\mathcal{S} = \{(a_i, b_j, c_k) \mid i, j, k \in [0; 2]\}$. *For the automaton* a, $\mathcal{S}_a = \{a_0, a_1, a_2\}$ *and* $\mathcal{T}_a = \{a_0 \xrightarrow{b_2} a_1, a_1 \xrightarrow{b_1, c_2} a_2\}$. *For the transition* $\tau = a_1 \xrightarrow{b_1, c_2} a_2$, $orig(\tau) = a_1$, $dest(\tau) = a_2$ *and* $enab(\tau) = \{b_1, c_2\}$.

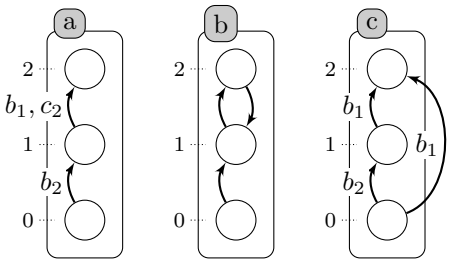

Figure 1: An Automata Network with 3 automata.

To describe the dynamics of Automata Networks, it is also necessary to specify the semantics. The semantics indicates which transitions can be played simultaneously from any global state. In this work, we focus on the asynchronous semantics, which means that, from one global state, only one transition can be played at a moment, among all the playable transitions. The asynchronous semantics has been preferably used for qualitative modeling in systems biology ([Assmann and Albert, 2009]). Indeed, qualitative models are focused on the representation of the transitions from different discrete states in the species of a biological system ([Chatain et al., 2018]). In the asynchronous update mode, it is assumed that the transitions in the local states of two different species will not happen at the same time.

The asynchronous semantics makes the model non-deterministic in the sense that each transition that is playable from one state leads to a different global state. Note that since transitions between the same local state (i.e. $x_i \xrightarrow{c} x_i$) are not considered here, self loops will not appear in the global reachability graph of the Automata Network.

Definition 2 *Transition playability (asynchronous semantics)*
Given an Asynchronous Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$*:*

- *A transition* $\tau \in \mathcal{T}_x$ *is playable from a global state* $s \in \mathcal{S}$ *if* $s(x) = orig(\tau)$ *and* $\forall y_i \in enab(\tau), y_i = s(y)$*. We define the playable predicate such that* $playable(s, \tau_x) = \top$ *if and only if* τ *is playable from* s*.*
- *The playability operator, denoted* $\cdot$*, is defined as* $s \cdot \tau = s'$ *where* $s'(y) = s(y) \quad \forall y \in \Sigma \setminus \{x\}$ *and* $s'(x) = dest(\tau)$ *if* $playable(s, \tau) = \top$*. If* $playable(s, \tau) = \bot$*,* $s \cdot \tau$ *is not defined.*
- *A sequence* π *of transitions of size* n *is denoted as* $\pi = (\pi^1 :: \pi^2 :: .. :: \pi^n)$ *with* $\pi^i \in \mathcal{T} \quad \forall i$ *and playing* π *(when possible) is denoted as* $s \cdot \pi = s \cdot \pi^1 \cdot \pi^2 \cdot .. \cdot \pi^n$*, with* $s \in \mathcal{S}$*.*

Example 2 *In the Automata Network of Fig. 1, from the local state* $s = (a_0, b_2, c_0)$*, the transitions* $\tau_1 = a_0 \xrightarrow{b_2} a_1$ *and* $\tau_2 = c_0 \xrightarrow{b_2} c_1$ *are playable and the transition* $\tau_3 = b_1 \rightarrow b_2$ *is not playable. In this model,* $s \cdot \tau_1 = (a_1, b_2, c_0)$*. For the sequence* $\pi = (c_0 \xrightarrow{b_2} c_1 :: b_2 \rightarrow b_1)$*,* $s \cdot \pi = s \cdot \pi^1 \cdot \pi^2 = (a_0, b_1, c_1)$*.*

From these definitions, the reachability problem is defined as questioning the existence of a transition sequence that makes the model evolve from a global initial state to some state which verifies the presence of certain local states. This problem can also be reduced to the reachability of only one local state, by using an additional automaton in the model, as explained in [Folschette et al., 2015]. For this reason, we will only consider the reachability of a single local state in this work. Regarding the complexity, this problem can be proved PSPACE-Complete through a transformation to bounded Petri nets ([Cheng et al., 1995, Paulevé, 2018]). Indeed, the number of (global) states of an Automata Network is finite and exponential in the number of automata and the reachability problem can be seen as finding a path between two states in the state transitions graph of the model.

Definition 3 *Reachability in Asynchronous Automata Networks*

- *Input: An Automata Network* $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$*, a reachability tuple* $\mathcal{R} = (s_0, x_i)$ *with* $s_0 \in \mathcal{S}$ *an initial global state and* $x_i \in \mathcal{S}_x$ *a goal local state*
- *Output:* $\top$ *or* $\bot$
- *Issue: Is there a sequence of transitions* $\pi = (\pi^1 :: \pi^2 :: .. :: \pi^n)$ *such that* $s_0 \cdot \pi = s'$ *with* $s'(x) = x_i$ *?*

From this latter definition, solutions are formalized by sequences of transitions. However, when a reachability instance admits a solution, this instance may admit an infinite number of solution sequences. Indeed, it can be possible for example to insert arbitrarily long loops inside a solution. The corresponding notion of solution in [Paulevé, 2018] has been refined (as sub-traces and minimal traces) in order to represent only canonical solutions. For this reason, we introduce in this work the equivalent notions of sub-solutions and minimal solutions. A sub-solution corresponds to a sub-sequence of a solution (containing only transitions of the original sequence, with a conservation of their order) and that is also a solution to the reachability instance. A minimal solution is a solution which does not admit any sub-solution. It is also worth noting that solutions and minimal solutions may contain multiple occurrences of the same transition.

Definition 4 *Solutions, sub-solutions and minimal solutions for reachability*
Let $\mathcal{R} = (s_0, x_i)$ *be a reachability instance.*
- A sequence π_1 *of transitions is a solution to the reachability instance* $\mathcal{R}$ *if* $[s_0 \cdot \pi_1](x) = x_i$*.*
- A sequence π_2 *of transitions is a sub-solution of a solution* π_1 *if* π_2 *is a solution and if there exists an injection* $\phi : [1; |\pi_2|] \rightarrow [1; |\pi_1|]$ *such that* $\forall i \in [1; |\pi_2|], \pi_1^{\phi(i)} = \pi_2^i$ *and* $\forall j \in [1; |\pi_2| - 1], \phi(j) < \phi(j+1)$ *(*π_2 *contains a subset of transitions from* π_1*, in the same order).*
- A solution π *is a minimal solution to* $\mathcal{R}$ *if there exists no sub-solution* π' *of* π*.*

Example 3 *In the Automata Network of Fig. 1, the answer of the reachability instance* $\mathcal{R} = ((a_0, b_0, c_0), a_2)$ *is* $\top$*.*
$\pi_1 = (b_0 \rightarrow b_1 :: b_1 \rightarrow b_2 :: b_2 \rightarrow b_1 :: b_1 \rightarrow b_2 :: a_0 \xrightarrow{b_2} a_1 :: b_2 \rightarrow b_1 :: c_0 \xrightarrow{b_1} c_2 :: a_1 \xrightarrow{b_1, c_2} a_2)$ *is a solution (i.e.* $[(a_0, b_0, c_0) \cdot \pi_1](a) = a_2$*).*
The solution $\pi_2 = (b_0 \rightarrow b_1 :: b_1 \rightarrow b_2 :: a_0 \xrightarrow{b_2} a_1 :: b_2 \rightarrow b_1 :: c_0 \xrightarrow{b_1} c_2 :: a_1 \xrightarrow{b_1, c_2} a_2)$ *is a sub-solution of* π_1 *and it is a minimal solution.*

2.2 Related works

Several approaches have been used to solve the reachability problem in Automata Networks. For example, one can use static analysis techniques, which consist in verifying some properties without any execution of the system ([Wichmann et al., 1995]). A method based on static analysis has been employed in [Folschette et al., 2015]. This method is based on necessary and sufficient conditions on reachability. The verification of these conditions is usually computationally efficient and, depending on the result, it can be possible to conclude if the reachability is true or not. However, there exists reachability instances for which the result on the verification of the conditions cannot be used (when the necessary conditions are verified and the sufficient conditions are not verified). Nonetheless, it has been observed in the study that, on some biological models, this method is mostly conclusive.

The reachability problem also falls into the category of Model Checking. Formally, Model Checking consists in checking a temporal property on a mathematical model ([Clarke, 2008]). Several techniques have been created to verify exhaustively large scale systems. This field has applications in various domains such as software verification, parallel systems and systems biology. For instance, model checking has been employed in [Češka et al., 2014] for the verification of biochemical networks, ([Heath et al., 2008]) for the analysis of biological pathways.

Explicit approaches have been applied in Model Checking to biological systems ([Klarner et al., 2012a, Klarner et al., 2012b, Gallet et al., 2014]). The use of symbolic methods such as Binary Decision Diagrams (BDD) was also originally popular in Model Checking ([Burch et al., 1990]). Several general symbolic model checking tools such as NuSMV [1] or ITS-tools [2] can be used to encode the dynamics of Automata Networks. However some of these symbolic methods have not been proven efficient on large scale models ([Bryant, 1986]) and some instances can be difficult to verify with state of the art approaches, as we can see in the benchmarks from [Paulevé, 2017].

The Bounded Model Checking framework became widely used in Model Checking to study dynamical properties like reachability ([Biere et al., 2009]). This framework consists in encoding execution sequences of a dynamical system using a logic programming language. Then, a dedicated solver is used to exhaustively search among all the sequences of fixed length and look for specific temporal properties. This approach has been used to solve various problems in biological networks ([Dubrova and Teslenko, 2011, Nabli et al., 2016]). For example, solvers for SAT problems have been used to solve reachability problems in Petri nets ([Ogata et al., 2004]), and to compute attractors in asynchronous boolean networks ([Van Giang and Hiraishi, 2020], where the methodology slightly diverges from classical Bounded Model Checking). An encoding have been proposed for reachability analysis in synchronous Gene Regulatory Networks with SMT solving in ([Giacobbe et al., 2017]). An Answer Set Programming encoding has also been developed for the computation of attractors in Automata Networks ([Ben Abdallah et al., 2017]).

Although Bounded Model Checking has demonstrated its efficiency to verify properties on large scale systems, proving unreachability still remains difficult as it requires a sufficient length for the encoding of execution sequences. For a finite state model, it is for instance possible to take the number of states of the system but this number is usually exponential in the size of the model. More generally, Bounded Model Checking also offers techniques to determine a *completeness threshold* of the system (i.e. a bound on the sufficient length of reachability sequences) through the encoding of inductive formula (k-induction techniques, [Biere et al., 2009]). For example, it is possible to encode the computation of the longest shortest path that reaches the goal state. However, this type of bound remains very general and its computation can be particularly difficult, as it is explained in appendix B.

In this paper, a bound dedicated to Automata Networks is derived from the existing static analysis approaches. This work aims at combining the bound with a Bounded Model Checking approach to tackle unreachability cases. Indeed, an exhaustive search on execution sequences of the model, combined with a reasonable bound on the reachability instance, can be used to efficiently make unreachability proofs.

[1] `http://nusmv.fbk.eu/NuSMV/`

[2] `https://lip6.github.io/ITSTools-web/index.html`

3 Static analysis and Local Causality Graph

As mentioned previously, our approach to solve the reachability problem in Automata Networks relies on existing work using static analysis, that is introduced in [Folschette et al., 2015]. This approach is based on the consideration of local reachability problems (i.e. a reachability problem within an automata). In this section, we introduce notions from this work that will be used thereafter. The definitions given here are not exactly identical with the definitions in the previous works. However, they are equivalent and the known results still hold.

3.1 Local Causality Graph

The main element presented in this section is a graph called the Local Causality Graph. This graph represents local reachabilities (automaton reachabilities), which are defined through objectives and local solutions. Indeed, solving a reachability problem on the whole Asynchronous Automata Network can be seen equivalently as solving a local reachability in one automaton (from the goal local state), which leads to sub reachabilities on the other automata (depending on the conditions of transitions). An objective represents a local reachability problem (i.e. in one automaton) and a local solution is an acyclic sequence of transitions from this automaton, that locally solves the corresponding objective (all the local states are reached at most once).

Definition 5 *Objective and local solution*
Given an Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$*:*

- *A local reachability problem of an automaton* $x \in \Sigma$ *is represented by tuple* $(x_i, x_j) \in \mathcal{S}_x \times \mathcal{S}_x$ *called objective and denoted* $x_i \rightsquigarrow x_j$
- *A local solution to an objective* $x_i \rightsquigarrow x_j$ *is a sequence of transitions* $\eta = (\eta^1 :: .. :: \eta^n)$ *(with* $\eta^i \in \mathcal{T}_x \quad \forall i \in [1;n]$*) such that* $orig(\eta^1) = x_i$*,* $dest(\eta^n) = x_j$ *and* $dest(\eta^k) = orig(\eta^{k+1}) \quad \forall k \in [1; n-1]$ *and* $\forall i, j \in [1;n], j > i \implies orig(\eta^i) \neq dest(\eta^j)$
- *The finite set of all local solutions associated with* $x_i \rightsquigarrow x_j$ *is denoted as* $lpaths(x_i \rightsquigarrow x_j)$
- $lpaths(x_i \rightsquigarrow x_i) = \{\varepsilon\} \quad \forall x_i \in \mathcal{S}_x, \forall x \in \Sigma$

Example 4 *In the Automata Network of Fig. 1, the objective* $c_0 \rightsquigarrow c_2$ *has two solutions. There is one solution* $\eta_1 = (c_0 \xrightarrow{b_2} c_1 :: c_1 \xrightarrow{b_1} c_2)$ *and a second solution* $\eta_2 = (c_0 \xrightarrow{b_1} c_2)$*. The objective* $b_0 \rightsquigarrow b_0$ *admits only a trivial solution* $\eta_3 = \varepsilon$ *since the local reachability is already satisfied.*

In practice, the objectives are solved through an exhaustive search. The number of local states for each automaton is in fact relatively small, as mentioned in subsection 2.1. Note that each transition does not appear more than once in a local solution, which makes the set of local solutions finite. Although some solutions of a reachability instance may contain duplicates of transitions, it will be shown in the following that decomposing a minimal solution into local solutions is always possible. Then, local solutions can be used as additional information to solve a reachability problem. Indeed, the local solutions of an objective may require some states to be realized, depending on the conditions of the transitions. These local states lead to intermediary objectives that needs to be solved, as the conditions of the transitions are not necessarily satisfied in the initial state. This recursive definition results in the construction of a graph, which is called the Local Causality Graph.

Note that the definition of the Local Causality Graph presented here slightly differs from the definition in [Folschette et al., 2015]. Indeed, the transitions of the automata were not exhibited in the original Local Causality Graphs. In this paper, we show that adding the transitions reveals interesting information about the reachability problem. This alternative Local Causality Graph still contains the information from the original graph and a translation between the two definitions is possible.

Definition 6 *Local Causality Graph (LCG)*
Given an Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ and a reachability instance $\mathcal{R} = (s_0, x_i)$, a Local Causality Graph associated with $\mathcal{M}$ and $\mathcal{R}$ is a digraph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ with $\mathcal{V}_\mathcal{G} = \mathcal{L} \cup \mathcal{O} \cup \mathcal{Q} \cup \mathcal{T}'$ such that :

- $\mathcal{L} \subset \bigcup_{y \in \Sigma} \mathcal{S}_y$ *(local states)*
- $\mathcal{O} \subset \bigcup_{y \in \Sigma} \mathcal{S}_y \times \mathcal{S}_y$ *(objectives)*
- $\mathcal{Q} = \bigcup_{y_i \rightsquigarrow y_j \in \mathcal{O}} lpaths(y_i \rightsquigarrow y_j)$ *(local solutions)*
- $\mathcal{T}' = \bigcup_{o \in \mathcal{O}} \{\eta^i \mid i \in [1; |\eta|], \eta \in lpaths(o)\}$ *(transitions)*
- $\mathcal{E}_\mathcal{G} \subset \mathcal{V}_\mathcal{G} \times \mathcal{V}_\mathcal{G}$ *(directed edges)*

And which satisfies the following conditions:

1. $x_i \in \mathcal{L}$ *(overall objective)*
2. $(y_i, s_0(y) \rightsquigarrow y_i) \in \mathcal{E}_\mathcal{G} \quad \forall y_i \in \mathcal{L}$ *(objectives from initial state)*
3. $(y_i \rightsquigarrow y_j, \eta) \in \mathcal{E}_\mathcal{G} \quad \forall \eta \in lpaths(y_i \rightsquigarrow y_j), \forall y_i \rightsquigarrow y_j \in \mathcal{O}$ *(all local solutions)*
4. $(y_j, y_i \rightsquigarrow y_j) \in \mathcal{E}_\mathcal{G} \quad \forall y_i, y_j \in \mathcal{L}$ *(saturation)*
5. $\forall \eta \in \mathcal{Q}, (\eta, \eta^i) \in \mathcal{E}_\mathcal{G} \quad \forall i \in [1; |\eta|]$ *(transitions of the local solutions)*
6. $\forall \tau \in \mathcal{T}', \forall y_i \in enab(\tau), (\tau, y_i) \in \mathcal{E}_\mathcal{G}$ *(required local states)*

The construction of the Local Causality Graph starts from the goal local state (from the reachability problem) and one associated objective, from the initial state to this local state (conditions 1 and 2). Then, the different conditions of the definition are applied to the graph, and vertices and edges are added until all the conditions are satisfied. Each time a local state appears in the graph, the objective from the initial state of this automaton to this local state has to be added (condition 2). However, it is not easy to predict which objective would be used for the reachability of a local state, since it depends on the previous local reachabilities that have been solved. That is why all the objectives from any local state of the graph from the same automaton to this local state are added to the graph (condition 4). When an objective is added, all its local solutions are automatically added to the graph (condition 3) with their corresponding transitions (condition 5) and the associated local states required to play these transitions (condition 6).

It is useful to mention that the local solutions vertices "○" in a Local Causality Graph correspond to subsets of transitions, to which they are connected according to condition 4. The presence of a local solution vertex is justified by the fact that it is interesting to distinguish between the different local solutions of an objective in order to solve a reachability. Besides, all vertices appear only once in the graph. For example, if a local state $x_i \in \mathcal{S}_x$ is in the condition of more than one transition, there will be only one vertex $x_i \in \mathcal{V}_\mathcal{G}$ in the graph. All these transitions will be then connected to this single vertex. Thanks to this, the size of the graph is polynomial in the size of the Automata Network (assuming the number of local solutions for each objective is small). On the other side, this potentially induces the presence of cycles, which will be discussed later.

Example 5 *The graph in Fig. 2 is a Local Causality Graph associated with the reachability problem $\mathcal{R} = ((a_0, b_0, c_0), a_2)$ for the Automata Network of Fig. 1. The numbers labeling some of the vertices are explained later. In this graph, the objectives of the form $x_i \rightsquigarrow x_i$ have not been added since they do not bring additional information for the solution. The local state b_1 has two predecessors, because it appears in the conditions of two transitions and as explained above, the vertices must appear only once in the graph. Finally, the sufficient conditions mentioned in subsection 3.2 are satisfied in this graph, which implies that reachability of a_2 is possible from (a_0, b_0, c_0).*

3.2 Necessary and sufficient conditions for reachability

As already mentioned, the Local Causality Graph has been created and used to solve reachability problems in a static way. *Necessary conditions* for reachability relying on the Local Causality Graph are introduced in [Paulevé, 2018]. These conditions are defined from an incomplete Local Causality Graph, which contains only objectives starting from the local states of the initial state. The necessary conditions state that

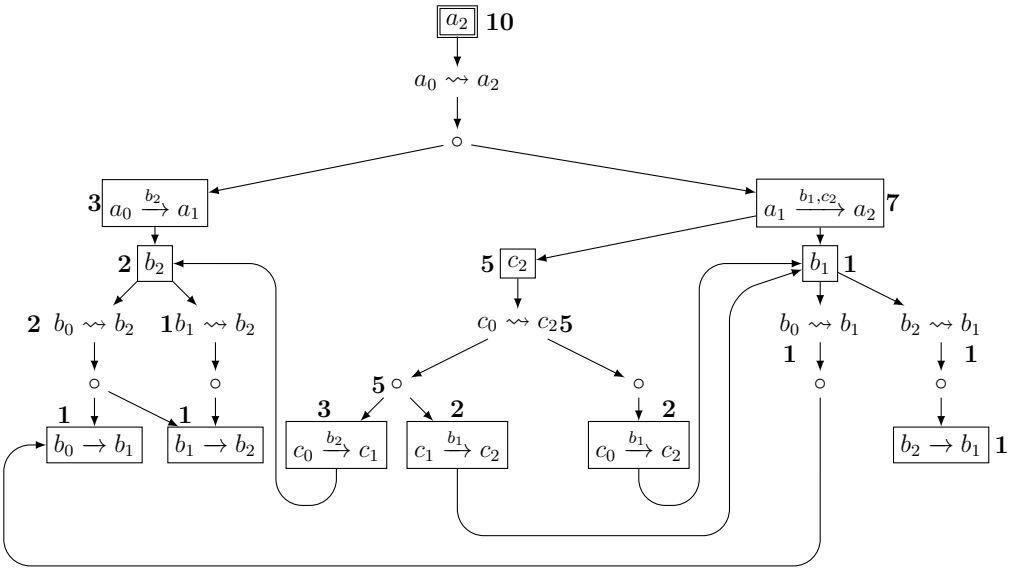

Figure 2: A Local Causality Graph associated with the reachability problem of Example 3. In this representation, boxed vertices correspond to local states and local transitions. Objectives are represented with the same notation introduced previously and ∘ vertices represent local solutions (i.e. sequences of local transitions solving an objective).

given an initial state, if there is no local solution to an objective starting from a local state in the initial global state, then its reachability is not possible (i.e. there is no local path to solve the objective within the automaton). This property can be propagated to the successors of the local solution in the graph to prove that some reachabilities are not possible. The verification can be done with an algorithm that is polynomial in the size of the Automata Network.

These conditions have been used in the study to reduce the size of a Local Causality Graph by proving that some objectives (if not all) are impossible. In addition, it can be shown that if a reachability is true, then the transitions of the minimal solutions for this reachability all belong to the Local Causality Graph. In other words, the transitions of the Local Causality Graph are sufficient to solve the reachability, and other transitions may be discarded for this specific instance. This property has been used in the study to reduce the computational cost of solving reachability with Model Checking tools. This property can also be used to remove unreachable local states and automata from the model in some cases.

Sufficient conditions for reachability are introduced in [Folschette et al., 2015]. These conditions can be used to prove, in some cases, that the reachability is true. The idea behind these conditions is that if every objective of some local state in the graph has at least one solution, its reachability must be true. Indeed, if all objectives admit some solution, it means that the local state can be reached from any other local state of the same automaton in the graph, which slackens the possibilities of the Automata Network (the local states can be reached in any order). This condition can be propagated to the successors of the local state in the Local Causality Graph. It results as a verification that is again polynomial in the size of the Automata Network. Moreover, when the conditions are verified, a solution sequence can be computed. Although these conditions have been successfully applied on large biological models, the sufficient conditions represent an under-estimation of the solutions to a reachability instance. Thus, there are still simple examples for which the verification fails, as it is explained in Example 6.

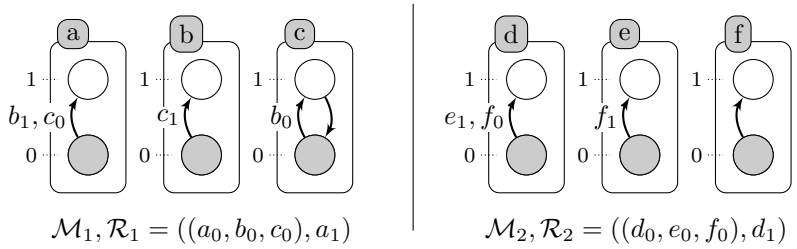

Figure 3: Two Automata Networks illustrating where the necessary conditions are verified and where the sufficient conditions are not verified.

Example 6 *The two Automata Networks $\mathcal{M}_1$ and $\mathcal{M}_2$ in Fig. 3 illustrate simple cases where the verification of the reachability by the necessary and the sufficient conditions fails.*
- In the network $\mathcal{M}_1$ (on the left), the reachability $\mathcal{R}_1 = ((a_0, b_0, c_0), a_1)$ is possible and the following transition sequence is a solution: $(c_0 \xrightarrow{b_0} c_1 :: b_0 \xrightarrow{c_1} b_1 :: c_1 \to c_0 :: a_0 \xrightarrow{b_1, c_0} a_1)$.
However sufficient conditions verification fails at proving that the reachability is true because, in the corresponding Local Causality Graph, the objective $b_1 \rightsquigarrow b_0$ has no local solutions (see appendix D.1). The sufficient conditions ensure that the local states from the Local Causality Graph are reachable in any order, and this is not verified in this example because b_1 cannot be reached from b_0 (no local solutions for $b_1 \rightsquigarrow b_0$). This objective is practically useless for the reachability of a_1 since b_0 does not need to be reached when b_1 is activated. But it appears in the Local Causality Graph because b_1 and b_0 are both present.
- Regarding the network $\mathcal{M}_2$, the reachability instance $\mathcal{R}_2 = ((d_0, e_0, f_0), d_1)$ is not possible. Indeed, e_1 and f_0 are required to play the transition from d_0 to d_1, but f_1 needs to be activated in order to activate e_1 and f_0 cannot be activated afterward. In this example, the necessary conditions are yet verified since every local state of the associated Local Causality Graph is still locally reachable from its local state in the initial state (appendix D.2). Indeed, the necessary conditions consist in verifying that the all objectives starting from the local states in the initial state have at least one solution. Intuitively, for an objective, the absence of local solutions from the local state in the initial state implies the absence of local paths solving the local state in a global solution.

4 Proofs of unreachability and computation of a bound

This paper focuses on the unreachability case in Asynchronous Automata Networks. From a biological point of view, proving unreachability is much more important than proving that reachability is true. Indeed, the guarantee that a state is accessible remains dummy since unobserved conditions related to the environment will always possibly prevent actually reaching this state. Unfortunately, as explained before, the necessary conditions presented in section 3.2 are not able to tackle all unreachability cases. Our approach to prove unreachability consists in taking advantage on the ability of Bounded Model Checkers to perform exhaustive search over execution sequences of large scale models. However, a *completeness threshold* (i.e. a bound on the length of reachability sequences) remains necessary to formally prove the unreachability case.

In this work, we propose a method to compute a dedicated bound for Automata Networks. The originality of our approach lies in the use of the Local Causality Graph. This section introduces our bound, named the *Local Causality Bound*. The Local Causality Bound aims to be specifically adapted to the Asynchronous Automata Networks and we expect this bound to be smaller than more generic *completeness thresholds*, as it is developed in appendix B. We assume that a smaller bound will help the Model Checkers to give more efficiently an answer. In this part, the first subsection introduces the main definition and property about the Local Causality Bound. Then, the outline of its proof is detailed in subsection 4.2. Complete proofs of intermediary results can be found in appendix C. Some specific cases of the computation of the bound are investigated in subsection 4.3. Subsection 4.4 shows some practical

examples of the Local Causality Bound. Finally, the general application of the bound is discussed in subsection 4.5.

4.1 Local Causality Bound

Local Causality Graphs aim to over-approximate the solutions of a reachability problem, by linking multiple local reachability problems. Here we investigate how this over-approximation could be exploited to derive a sufficient length on the number of transitions necessary to solve the reachability, and its use to formally prove unreachability. Indeed, the Local Causality Graph can also be computed when the associated reachability instance has no solution. Unfortunately, not all solutions can be represented by a Local Causality Graph since some of them may be arbitrarily long, or they may contain useless transitions, that are not captured by the graph. For this reason, it will be shown that the minimal solutions introduced in Definition 4 can be used to characterize all solutions that are captured by a Local Causality Graph.

Locally, the number of transition necessary to solve an objective can be over-estimated by the length of the longest local solution for that objective. This number may be used to build a bound, but it would not cover the transitions from other automata that are necessary to form a complete solution. Nevertheless, the structure of the Local Causality Graph can be used since the transitions of a local solution are connected to the local states of its conditions. If the bounds of sub-objectives associated with the local states are already known, one can recursively compute a bound for the initial objective.

Since the computation of the bound for one vertex of the Local Causality Graph would depend on its successors in the graph, the next definition proposes a possible computation of the bound as a function of any vertex of the graph. The function is recursive because the bound of one vertex depends on the bounds of the vertices that are connected to it. However, this definition does not directly allow the computation of the bound in a cyclic Local Causality Graph, as it will be discussed afterwards.

Definition 7 *Bound function*
Let $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ be an Automata Network, $\mathcal{R} = (s_0, x_i)$ a reachability problem, and $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ a Local Causality Graph associated with $\mathcal{M}$ and $\mathcal{R}$. Following definition 6, $\mathcal{V}_\mathcal{G}$ is defined as $\mathcal{V}_\mathcal{G} = \mathcal{L} \cup \mathcal{O} \cup \mathcal{Q} \cup \mathcal{T}'$ with $\mathcal{L} \subset \bigcup_{y \in \Sigma} \Sigma_y, \mathcal{O} \subset \bigcup_{y \in \Sigma} \Sigma_y \times \Sigma_y, \mathcal{Q} = \bigcup_{o \in \mathcal{O}} lpaths(o)$ and $\mathcal{T}' = \bigcup_{o \in \mathcal{O}} \{\eta^i \mid i \in [1; |\eta|], \eta \in lpaths(o)\}$.
The bound function associated with $\mathcal{G}$ is a function $B : \mathcal{V}_\mathcal{G} \to \mathbb{N}$ with:

$$
\begin{aligned}
-B(\tau) &= 1 + \sum_{y_i \in enab(\tau)} B(y_i) && \forall \tau \in \mathcal{T}' \\
-B(\eta) &= \sum_{k \in [1;|\eta|]} B(\eta^k) && \forall \eta \in \mathcal{Q} \\
-B(o) &= \max_{\eta \in lpaths(o)} B(\eta) && \forall o \in \mathcal{O} \\
-B(y_j) &= \max_{y_i \rightsquigarrow y_j \in \mathcal{O}} B(y_i \rightsquigarrow y_j) && \forall y_j \in \mathcal{L}
\end{aligned}
$$

Intuitively, the bound function relies on both necessary and the sufficient aspects of the Local Causality Graph. The number of transition associated with a local state is sufficiently approximated by the maximum bound of the corresponding objectives of the graph, and the number of transitions of an objective is sufficiently approximated by the maximum bound of all the local solutions. Indeed, in both cases, it is not possible to decide which objective/local solution would be used in a minimal solution to the reachability. On the contrary, since all transitions are necessary in a local solution and since all their conditions need to be satisfied to play the transitions, their associated bounds take into account all the vertices that are successors in the Local Causality Graph. As it will be proved in subsection 4.2, the bound function, applied to the goal local state of the reachability instance indeed gives a *completeness threshold* for the reachability. This property is formalized in the following proposition:

Proposition 1 *Local Causality Bound for a reachability instance*
Let $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ be an Automata Network, $\mathcal{R} = (s_0, x_i)$ a reachability instance, $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ a Local Causality Graph associated with $\mathcal{M}$ and $\mathcal{R}$, and B the bound function associated with $\mathcal{G}$.
– For all transition sequences π such that π is a minimal solution of $\mathcal{R}$, $|\pi| \leq B(x_i)$

The next example illustrates how the bound can be computed in practice, with an acyclic Local Causality Graph.

Example 7 *Fig. 2 shows the bounds computed for the Local Causality Graph of the reachability instance $((a_0, b_0, c_0), a_2)$ of the Automata Network in Fig. 1. For clarity, some of the bounds have been discarded. We can see that the bound $B(a_2) = (2 + 1) + (5 + 1 + 1) = 3 + 7 = 10$ correctly overestimates the length of the solution ($= 6$) given in Example 3.*

As mentioned above, the computation of a bound is useful as soon as one is able to exhaustively check the sequences of a fixed length, with the use of a Bounded Model Checking approach for example. Although this type of encoding have already been proposed for various logical models, a SAT encoding specifically dedicated to Asynchronous Automata Network has been implemented in this work. The SAT encoding is detailed in appendix A. An implementation of the bound computation and the SAT encoding are also available online[3].

4.2 Proof of the Local Causality Bound

To formally prove that the Local Causality Bound is able to over-estimate the length of all minimal solutions to a reachability instance, we propose an association between each minimal solution and the Local Causality Graph, that shows precisely how the graph is actually able to over-approximate the minimal solutions. This association consists in decomposing the minimal solution into multiple objectives. These objectives are identified from the conditions of the transitions in the solution and they are associated to sub (local) transition sequences forming local solutions. Since the same objective may be solved several times in a solution, the identification is based on the index of the transition having the condition. We also show that the identified objectives can be associated between each other to form a structure that is similar to the Local Causality Graph. From this association, it is possible to prove by induction on this structure that the Local Causality Bound correctly overestimates the number of transitions of any minimal solution. This subsection shows all the steps that are necessary to build this association. The last part of the proof consists in two lemmas that are demonstrated in appendix C.

In this part, the following notations are used: $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ denotes an Asynchronous Automata Network. $\mathcal{R} = (s_0, x_i)$ represents a reachability instance, assuming $s_0(x) \neq x_i$ and π is a minimal solution to that instance ($s_0 \cdot \pi(x) = x_i$). $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ is a Local Causality Graph associated with $\mathcal{R}$, with $\mathcal{V}_\mathcal{G} = \mathcal{L} \cup \mathcal{O} \cup \mathcal{Q} \cup \mathcal{T}'$, and B is the bound function previously defined.

The following definition introduces sequences of ordered transition indexes from the minimal solution π, for each automaton of the model. These sequences intend to catch all transitions of a local solution having a condition on this specific automaton. The corresponding local state (from the condition) will be associated with an objective afterwards.

Definition 8 *Transitions conditioned on y*
Let $\pi = (\pi^1 :: .. :: \pi^n)$ be a minimal solution for the reachability instance $\mathcal{R} = (s_0, x_i)$. $\forall y \in \Sigma, \mathcal{I}_{\pi,y}$ is the sequence of ordered indexes of all transitions having a condition on y:
- $enab(\pi^{\mathcal{I}^j_{\pi,y}}) \cap \mathcal{S}_y \neq \varnothing \quad \forall j \in [1; |\mathcal{I}_{\pi,y}|]$ (all referred transitions have a condition on y)
- $\mathcal{I}^j_{\pi,y} < \mathcal{I}^{j+1}_{\pi,y} \quad \forall j \in [1; |\mathcal{I}_{\pi,y}| - 1]$ (the indexes are ordered)
- $enab(\pi^k) \cap \mathcal{S}_y = \varnothing \quad \forall k \notin \mathcal{I}_{\pi,y}$ (there is no other transition having a condition on y in the solution)

The next definition, relying on these sequences of indices, aims to identify sequences of objectives locally for each automaton. These objectives correspond to objectives that are solved through local solutions in the minimal solution π. They also correspond to objectives of the Local Causality Graph. If an automaton does not have any identified condition in the minimal solution, the corresponding sequence of objectives would be empty. For the goal local state x_i of the reachability instance, one additional objective is identified. Indeed, this local state does not appear in the conditions of the transitions.

[3]The implementation can be found at `https://github.com/smbct/AAN-reach`

Definition 9 *Objectives associated with a minimal solution*
$\mathcal{O}_{\pi,\_}$ *are the sequences of objectives of each automaton that are exactly solved by a minimal solution in* π. *i.e. the local states of these objectives correspond to the conditions for these automata on the other transitions. For all* $z \in \Sigma \setminus \{x\}$ *such that* $\mathcal{I}_{\pi,z} = \varepsilon$, *we have:*
- $\mathcal{O}_{\pi,z} = \varepsilon$
For all $y \in \Sigma \setminus \{x\}$ *such that* $\mathcal{I}_{\pi,y} \neq \varepsilon$, *we have* $|\mathcal{O}_{\pi,y}| = |\mathcal{I}_{\pi,y}|$ *and* $|\mathcal{O}_{\pi,x}| = |\mathcal{I}_{\pi,x}| + 1$
For $y \in \Sigma$ *such that* $\mathcal{I}_{\pi,y} \neq \varepsilon$ *(including* x*), we have:*
- $\mathcal{O}^1_{\pi,y} = s_0(y) \rightsquigarrow y_j$ *with* $y_j \in enab(\pi^{\mathcal{I}^1_{\pi,y}}) \cap \mathcal{S}_y$
- $\mathcal{O}^i_{\pi,y} = y_j \rightsquigarrow y_k$ *with* $y_j \in enab(\pi^{\mathcal{I}^{i-1}_{\pi,y}}) \cap \mathcal{S}_y$ *and* $y_k \in enab(\pi^{\mathcal{I}^i_{\pi,y}}) \cap \mathcal{S}_y$ *if* $1 < i \leq |\mathcal{I}_{\pi,y}|$
For x, *the automaton of the goal local state, we have also:*
- $\mathcal{O}^{|\mathcal{I}_{\pi,x}|+1}_{\pi,x} = \begin{cases} s_0(x) \rightsquigarrow x_i \text{ if } \mathcal{I}_{\pi,x} = \varepsilon \\ x_k \rightsquigarrow x_i, x_k \in enab(\pi^{\mathcal{I}^{|\mathcal{I}_{\pi,x}|}_{\pi,x}}) \cap \mathcal{S}_x \text{ otherwise} \end{cases}$

The next definition creates an association between the objectives previously identified and the transitions of the minimal solution. It is possible to show that these transitions also correspond to minimal solutions of the associated Local Causality Graph.

Definition 10 *Transitions associated with the objectives*
Let $\mathcal{O}_{\pi,y}$ *be the sequence of objectives associated with* y *in the minimal solution* π, *with* $y \in \Sigma$ *(including* x*). When* $\mathcal{I}_{\pi,y} \neq \varepsilon$, *the transitions associated with each objective* $\mathcal{O}^i_{\pi,y}, i \in [1; |\mathcal{I}_{\pi,y}|]$ *of* $\mathcal{O}_{\pi,y}$ *are the (multi)sets defined as:*
$tr(\mathcal{O}^i_{\pi,y}) = \begin{cases} \{\pi^j \mid \pi^j \in \mathcal{T}_y, j < \mathcal{I}^1_{\pi,y}\} \text{ if } i = 1 \\ \{\pi^j \mid \pi^j \in \mathcal{T}_y, \mathcal{I}^{i-1}_{\pi,y} < j < \mathcal{I}^i_{\pi,y}\} \text{ if } i \in [2; |\mathcal{I}_{\pi,y}|] \end{cases}$ *and set of transitions associated with* $\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x}$
is defined as:
$tr(\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x}) = \begin{cases} \{\pi^j \mid \pi^j \in \mathcal{T}_x, j > \mathcal{I}^{|\mathcal{I}_{\pi,x}|}_{\pi,x}\} & \text{if } \mathcal{I}_{\pi,x} \neq \varepsilon \\ \{\pi^j \mid \pi^j \in \mathcal{T}_x\} & \text{otherwise} \end{cases}$

Finally, the next definition introduces the objective tree that aims to connect the identified objectives with each other to reveal their dependencies, and form a structure similar to the Local Causality Graph. The main property of the bound can then be proved on this recursive structure through structural induction.

Definition 11 *Objective tree associated with a minimal solution*
- The objective tree of a minimal solution π *is a digraph* $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$ *with* $\mathcal{V}_\mathcal{H} \subset \{\mathcal{O}^i_{\pi,y} \mid i \in [1; |\mathcal{O}_{\pi,y}|], y \in \Sigma\}$ *and* $\mathcal{E}_\mathcal{H} \subset \mathcal{V}^2_\mathcal{H}$, *verifying* $(\mathcal{O}^i_{\pi,y}, \mathcal{O}^j_{\pi,z}) \in \mathcal{E}_\mathcal{H} \iff \pi^{\mathcal{I}^j_{\pi,z}} \in tr(\mathcal{O}^i_{\pi,y})$ *(i.e. the transition having the condition on* z *finishing the objective is a transition associated with the parent objective)*

It will be shown that $\mathcal{H}$ corresponds to a tree (it has no cycles and it is connected). Indeed, $\mathcal{H}$ is finite since it contains the finite set of objectives identified from the minimal solution. Moreover, each objective can only be connected to an objective that was solved earlier in the solution, which does not allow the formation of cycles. Last, but not least, each objective has one predecessor except $\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x}$ which can be identified as the root of $\mathcal{H}$. We also introduce the notation $\prec_\mathcal{H}$ defined as: $o' \prec_\mathcal{H} o$ if there exists a path from o to o' in $\mathcal{H}$, i.e. if o' is a successor of o (direct or not direct).

The following example shows step by step the construction of the objective tree previously defined, from a minimal solution of a reachability instance.

Example 8
In this example, the previous definitions are applied to the rechability instance $\mathcal{R} = ((a_0, b_0, c_0), a_2)$ *on the Automata Network from Fig. 1, with its associated Local Causality Graph from Fig. 2 and with the minimal solution* $\pi_2 = (b_0 \to b_1 :: b_1 \to b_2 :: a_0 \xrightarrow{b_2} a_1 :: b_2 \to b_1 :: c_0 \xrightarrow{b_1} c_2 :: a_1 \xrightarrow{b_1,c_2} a_2)$ *from Example. 3.*
The sequences of indices for each automaton are:
- $\mathcal{I}_{\pi_2,a} = \varepsilon$; $\mathcal{I}_{\pi_2,b} = (3 :: 5 :: 6)$; $\mathcal{I}_{\pi_2,c} = (6)$
The associated sequences of objectives are:
- $\mathcal{O}_{\pi_2,b} = (b_0 \rightsquigarrow b_2 :: b_2 \rightsquigarrow b_1 :: b_1 \rightsquigarrow b_1)$; $\mathcal{O}_{\pi_2,c} = (c_0 \rightsquigarrow c_2)$; $\mathcal{O}_{\pi_2,a} = (a_0 \rightsquigarrow a_2)$
Their corresponding transitions are:
- $tr(\mathcal{O}^1_{\pi_2,a}) = \{a_0 \xrightarrow{b_2} a_1, a_1 \xrightarrow{b_1,c_2} a_2\}$

- $tr(\mathcal{O}^1_{\pi_2,b}) = \{b_0 \to b_1, b_1 \to b_2\}$; $tr(\mathcal{O}^2_{\pi_2,b}) = \{b_2 \to b_1\}$; $tr(\mathcal{O}^3_{\pi_2,b}) = \varepsilon$
- $tr(\mathcal{O}^1_{\pi_2,c}) = \{c_0 \xrightarrow{b_1} c_2\}$

Which leads to the construction of the following objective tree $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$*:*
- $\mathcal{V}_\mathcal{H} = \{\mathcal{O}^1_{\pi_2,a}, \mathcal{O}^1_{\pi_2,b}, \mathcal{O}^2_{\pi_2,b}, \mathcal{O}^3_{\pi_2,b}, \mathcal{O}^1_{\pi_2,c}\}$
- $\mathcal{E}_\mathcal{H} = \{(\mathcal{O}^1_{\pi_2,a}, \mathcal{O}^1_{\pi_2,b}), (\mathcal{O}^1_{\pi_2,a}, \mathcal{O}^3_{\pi_2,b}),$
  $(\mathcal{O}^1_{\pi_2,a}, \mathcal{O}^1_{\pi_2,c}), (\mathcal{O}^1_{\pi_2,c}, \mathcal{O}^2_{\pi_2,b})\}$.

For the following, $\mathcal{I}_{\pi,y}$ and $\mathcal{O}_{\pi,y}$ ($y \in \Sigma$) denote the lists of indexes and objectives that are constructed from the minimal solution π defined earlier. $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$ denotes an objective tree associated with π. The next lemma gives some properties from the previous definitions that are useful for the proof. The proof of these properties can be found in appendix C.

Lemma 1

1. $\forall y \in \Sigma, \forall i \in [1; |\mathcal{O}_{\pi,y}|], \exists \eta \in lpaths(\mathcal{O}^i_{\pi,y})$ *such that*
   $\{\eta^j \mid j \in [1; |\eta|]\} = tr(\mathcal{O}^i_{\pi,y})$ *($tr(o)$ corresponds to a local solution of o)*

2. $\bigcup_{o \in \mathcal{V}_\mathcal{H}} tr(o) = \{\pi^i \mid i \in [1, |\pi|]\}$ *(tr forms a partition of π)*

3. $\forall y \in \Sigma, \forall i \in [1; |\mathcal{O}_{\pi,y}|], \mathcal{O}^i_{\pi,y} \in \mathcal{V}_\mathcal{G}$ *(all the objectives identified belongs to the Local Causality Graph)*

4. $\mathcal{H}$ *is a tree (acyclic and connected)*

5. $\forall o \in \mathcal{V}_\mathcal{H}, \exists \eta \in lpaths(o)$ *such that* $\forall y_* \rightsquigarrow y_j : (o, y_* \rightsquigarrow y_j) \in \mathcal{E}_\mathcal{H}, \exists k \in [1; |\eta|]$ *such that* $(\eta^k, y_j) \in \mathcal{E}_\mathcal{G}$
   (all successors in the objective tree are (indirect) successors in the Local Causality Graph)

The next lemma is used to associate the bound function with the objectives identified in the minimal solution. This property is recursive (the bound of an objective is characterized with respect to the bounds of sub-objectives in the tree) and its application to the root of the objective tree gives an overestimate for all the transitions of the minimal solution. Its proof can also be found in appendix C.

Lemma 2
- $\forall o \in \mathcal{V}_\mathcal{H}, B(o) \geq |tr(o)| + \sum_{o' \prec_\mathcal{H} o} |tr(o')|$

The proof of lemma 2 can be done by structural induction on $\mathcal{H}$. Its base case corresponds to objectives that do not have any successor in $\mathcal{H}$. The transitions associated with these objectives simply correspond to local solutions. The inductive case occurs when an objective has some successors. In this case, it can be shown that the direct successors are also linked to the objectives in the Local Causality Graph, through the transitions of its local solutions (lemma 1.5). Assuming that the property is recursively true, it is possible to show that the bound of the objective takes into account the transitions associated to all its direct and not direct successors in the tree.

This lemma, when applied to the root of the tree (which is $\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x}$), proves that the bound function applied to x_i in the Local Causality Graph is able to overestimate the length of the minimal solution π. Indeed, lemma 1.2 implies that $\sum_{o \in \mathcal{V}_\mathcal{G}} |tr(o)| = |\pi|$. Also, since all objectives identified from the minimal solution belong to the Local Causality Graph, the root of $\mathcal{H}$ is a successor of x_i in $\mathcal{G}$. Thus, $B(x_i) \geq B(\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x})$ is able to overestimate $|\pi|$.

4.3 Bound computation in cyclic Local Causality Graphs

Proposition 1 can be applied to any Local Causality Graph, but the bound function is recursive: the bound of a vertex of the graph depends on the bounds of its direct successors. Thus, in a cyclic graph, this definition does not lead to a concrete bound since the vertices of the cycles are mutually dependent and no base case can be applied.

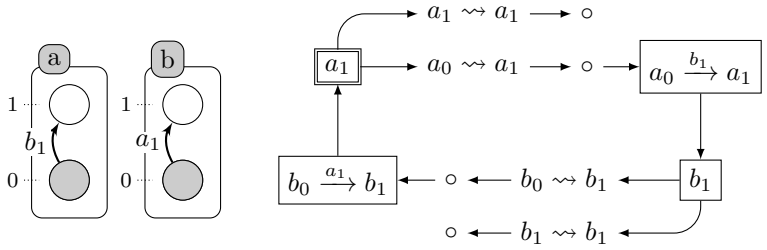

Figure 4: An Automata Network illustrating a simple cycle.

Intuitively, one would think that a cycle implies that the reachability is not possible. Indeed, a cycle in the Local Causality Graph means that a local state needs itself in order to be reached. The following example shows how intuitively such simple cycles may be removed in order to compute the Local Causality Bound.

Example 9 *Let $\mathcal{M}$ be the Automata Network of Fig. 4, $\mathcal{R} = ((a_0, b_0), a_1)$ be a reachability instance and $\mathcal{G}$ the Local Causality Graph associated with $\mathcal{R}$, also represented in Fig. 4.*
In this example, there is no solution to $\mathcal{R}$. More precisely, a_1 cannot be obtained from a_0 since the only local transition realizing this change requires b_1, which requires a_1 (to be reached from b_0). This mutual dependence creates the following cycle in the associated local causality graph: $(a_1, a_0 \rightsquigarrow a_1, \circ_{a_0 \rightsquigarrow a_1}, a_0 \xrightarrow{b_1} a_1, b_1, b_0 \rightsquigarrow b_1, \circ_{b_0 \rightsquigarrow b_1}, b_0 \xrightarrow{a_1} b_1)$. Thus, the local causality bounds of all these vertices are not computable, including the reachability goal a_1. However, the absence of solution in this case allows to remove some vertices of the cycle. Since $a_0 \rightsquigarrow a_1$ and $b_0 \rightsquigarrow b_1$ are both in the cycle, it is possible to assume that a_1 cannot be reached from (a_0, b_0), and similarly b_1 cannot be reached from (a_0, b_0). Thus, we can safely remove the local solution $\circ_{b_0 \rightsquigarrow b_1}$ regarding $\mathcal{R}$ for this Local Causality Graph. This leads to a sub graph $\mathcal{G}'$ of $\mathcal{G}$ where the local causality bound of a_1 can be computed, and equals 1. Since no solution exists for $\mathcal{R}$, 1 is indeed a correct overestimation. Besides, it is important to notice that the bound computed from the sub graph $\mathcal{G}'$ is only valid for $\mathcal{R}$. Indeed, in $\mathcal{G}'$, the bound of b_1 equals 0 but b_1 can be reached in one transition from (a_1, b_0). Since the objective $a_1 \rightsquigarrow a_1$ is present in $\mathcal{G}$, $\mathcal{G}$ is presumably valid for this alternative reachability instance but $\mathcal{G}'$ is not.

The previous example shows that it is possible in some situations to safely remove vertices from a Local Causality Graph while having the guarantee that the Local Causality Bound remains correct for a given reachability instance. Formalizing these cases is possible by identifying properties on the objective trees made from the minimal solutions of a reachability instance. For example, it is not possible to obtain an objective tree where an objective $x_0 \rightsquigarrow x_1$ has itself as a descendent, and where at the same time there is no objective of the form $x_* \rightsquigarrow x_0$. With these properties, it would be possible to identify sub parts of a Local Causality Graph that cannot be associated to any objective tree, proving that no minimal solutions exist for some objective.

However in some cases, removing any of the vertices inside a cycle automatically breaks the proposition 1 for the main reachability goal. Indeed, although simple cycles cannot form a minimal solution, in some more complex examples cycles may appear because of the combinatorial order of reachabilities between several automata. We illustrate this in the next example, where the automata b, c and d are mutually dependant in such a way that the only minimal solution to reach a_1 from (a_0, b_0, c_0, d_0) uses all the vertices in the cycle of the associated Local Causality Graph, and where some of the vertices (local transitions) from the cycle are used twice.

Example 10 *Let $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ be the Automata Network of Fig. 5 and $\mathcal{R} = (s_0, a_1)$ with $s_0 = (a_0, b_0, c_0, d_0)$. Let $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ be the Local Causality Graph associated with $\mathcal{M}$ and $\mathcal{R}$ represented in Fig. 6. The solution $\pi = (\mathbf{c_0} \xrightarrow{\mathbf{d_0}} \mathbf{c_1} :: \mathbf{d_0} \xrightarrow{\mathbf{c_1}} \mathbf{d_1} :: c_1 \rightarrow c_0 :: b_0 \xrightarrow{c_0, d_1} b_1 :: d_1 \xrightarrow{b_1} d_0 :: \mathbf{c_0} \xrightarrow{\mathbf{d_0}} \mathbf{c_1} :: \mathbf{d_0} \xrightarrow{\mathbf{c_1}} \mathbf{d_1} :: a_0 \xrightarrow{b_1, c_1, d_1} a_1)$ is a minimal solution to $\mathcal{R}$.*
This solution contains two duplicated transitions: $c_0 \xrightarrow{d_0} c_1$ and $d_0 \xrightarrow{c_1} d_1$. These transitions are also part

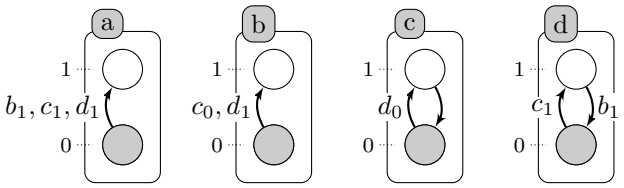

Figure 5: An Automata Network associated with cyclic Local Causality Graphs.

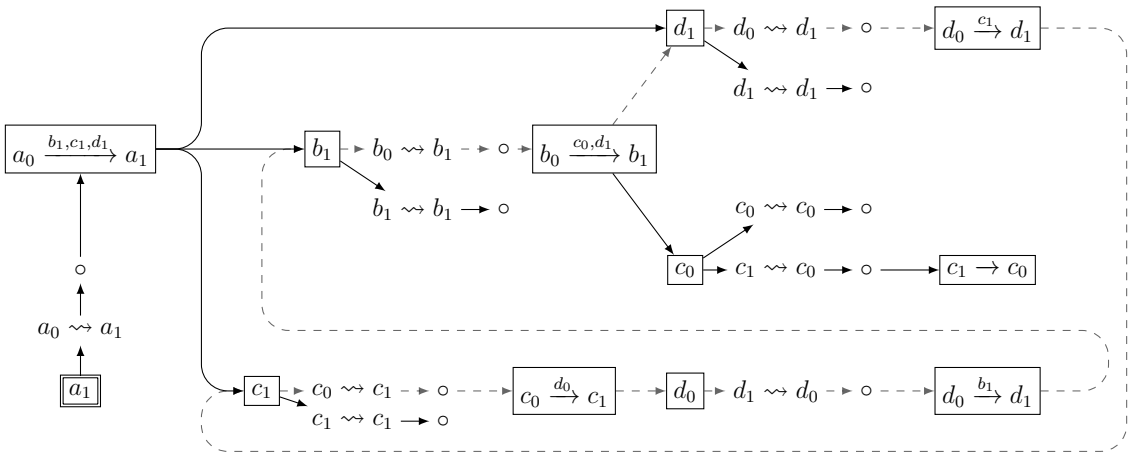

Figure 6: A Local Causality Graph with a cycle.

of a cycle in $\mathcal{G}$.
The association between π and $\mathcal{G}$ described in subsection 4.2 can be achieved through the following sequences of indices and objectives:
$\mathcal{I}_{\pi,a} = \varepsilon; \mathcal{I}_{\pi,b} = (5 :: 8); \mathcal{I}_{\pi,c} = (2 :: 4 :: 7 :: 8); \mathcal{I}_{\pi,d} = (1 :: 4 :: 6 :: 8)$
$\mathcal{O}_{\pi,a} = (a_0 \rightsquigarrow a_1); \mathcal{O}_{\pi,b} = (b_0 \rightsquigarrow b_1 :: b_1 \rightsquigarrow b_1); \mathcal{O}_{\pi,c} = (c_0 \rightsquigarrow c_1 :: c_1 \rightsquigarrow c_0 :: c_0 \rightsquigarrow c_1 :: c_1 \rightsquigarrow c_1); \mathcal{O}_{\pi,d} = (d_0 \rightsquigarrow d_0 :: d_0 \rightsquigarrow d_1 :: d_1 \rightsquigarrow d_0 :: d_0 \rightsquigarrow d_1)$
Which leads to the objective tree $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$, with:
$\mathcal{E}_\mathcal{H} = \{(\mathcal{O}^1_{\pi,a}, \mathcal{O}^2_{\pi,b}), (\mathcal{O}^1_{\pi,a}, \mathcal{O}^4_{\pi,c}), (\mathcal{O}^1_{\pi,a}, \mathcal{O}^4_{\pi,d}), (\mathcal{O}^4_{\pi,d}, \mathcal{O}^3_{\pi,c}),$
$(\mathcal{O}^3_{\pi,c}, \mathcal{O}^3_{\pi,d}), (\mathcal{O}^3_{\pi,d}, \mathcal{O}^1_{\pi,b}), (\mathcal{O}^1_{\pi,b}, \mathcal{O}^2_{\pi,c}), (\mathcal{O}^1_{\pi,b}, \mathcal{O}^2_{\pi,d}),$
$(\mathcal{O}^2_{\pi,d}, \mathcal{O}^1_{\pi,c}), (\mathcal{O}^1_{\pi,c}, \mathcal{O}^1_{\pi,d})\}$.
This objective tree is represented in Fig. 7. Now let us focus on the bound of c_1. Assuming it is possible to cut the cycle in $\mathcal{G}$, one would remove the edge $(d_0 \xrightarrow{c_1} d_1, c_1)$, which would lead to a bound of 0 for d_1 (only the trivial solution remains valid). Then, the bound of c_1 would be equal to 4. However, according to the Local Causality Bound, we should have $B(c_0 \rightsquigarrow c_1) \geq |tr(c_0 \rightsquigarrow c_1)| + \sum_{o \prec_\mathcal{H} c_0 \rightsquigarrow c_1} |tr(o)|$, i.e.
$B(c_0 \rightsquigarrow c_1) \geq |tr(c_0 \rightsquigarrow c_1)| + |tr(d_1 \rightsquigarrow d_0)| + |tr(b_0 \rightsquigarrow b_1)| + |tr(c_1 \rightsquigarrow c_0)| + |tr(d_0 \rightsquigarrow d_1)| + |tr(c_0 \rightsquigarrow c_1)| + |tr(d_0 \rightsquigarrow d_0)| \implies B(c_0 \rightsquigarrow c_1) \geq 6$.
This means that the association between the Local Causality Graph and the minimal solution is not possible if the cycle is cut.

This last example shows that in some instances where several automata are linked in a complex way trough their transitions, cycles appear in the Local Causality Graph, and at the same time duplicated transitions occur in the minimal solutions. This can also be seen on the objective tree made from the solution π, in Fig. 10. Indeed, while the objective tree remains acyclic, the objectives $d_0 \rightsquigarrow d_1$ and $c_0 \rightsquigarrow c_1$ are duplicated, indicating the multiple use of the vertices from the cycle of the Local Causality Graph. Thus cycles in the Local Causality Graphs may not be directly safely removed in these instances.

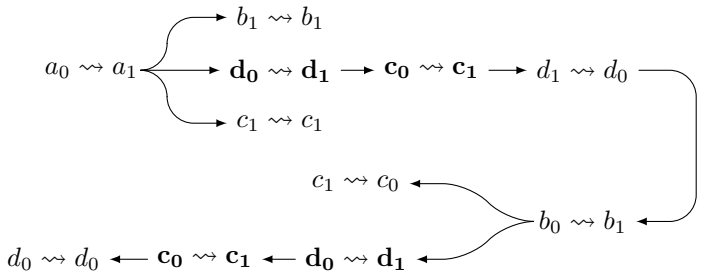

Figure 7: The objective tree for the solution π for $\mathcal{R} = ((a_0, b_0, c_0, d_0), a_1)$ in the example 10

However, we suggest that it is possible to refine (unfold) the Local Causality Graph by duplicating some vertices. The duplicates would allow to refine the objectives by partially re-ordering the reachability of the sub-goals, thus revealing simpler cases where the cycles can be removed. Besides, it is important to note that this particularity occurs only when the Local Causality Graphs have cycles and when the minimal solutions contain duplicates at the same time. The instances with duplicated transitions in the minimal solution and with an acyclic Local Causality Graph can still be directly treated with the bound.

4.4 Practical examples

To illustrate a practical application of proposition 1, we show here some examples that can be tested with the implementation of the bound computation and the SAT encoding mentioned in section 4.1 (see text file *readme.md*). The Automata Network models used in these examples can be found in the implementation. Each example shows a reachability instance with its answer, and the Local Causality Bound is given when it can be computed. It also shows the answer of the static analysis tool *Pint* [4]

The first reachability holds and the second reachability is false. For both of these examples, the computation of the Local Causality Bound is possible and *Pint* is not able to provide an answer.

Example 1:

- Model: *example_1.an*
- Initial state: (a_0, b_0, c_0, d_0)
- Goal state: a_3
- Reachability: Yes
- Local Causality Bound: 29
- Sufficient length: 11 (shortest sequence)
- *Pint*'s answer: Inconclusive

Example 2:

- Model: *example_3.an*
- Initial state: (a_0, b_0, c_0)
- Goal state: a_2
- Reachability: No
- Local Causality Bound: 8
- *Pint*'s answer: Inconclusive

[4] An implementation of *Pint* is available at `https://loicpauleve.name/pint/`. *Pint* is indeed able to provide a conclusion for unreachability without the use of a bound and it has already been compared to several Model Checking tools on biological models ([Paulevé, 2017]).

In the third example, the corresponding Local Causality Graph has cycles, thus the Local Causality Bound cannot be computed. Yet, the reachability is true and a solution can be found with the SAT encoding, by manually choosing the length of the sequence encoded.

Example 3:

- Model: *example_2.an*
- Initial state: $(a_0, b_0, c_0, d_0, e_0)$
- Goal state: a_1
- Reachability: Yes
- Local Causality Bound: Not computable (cyclic Local Causality Graph)
- Sufficient length: 8 (shortest sequence)
- *Pint*'s answer: Inconclusive

4.5 Discussion

As previously seen, the computation of the Local Causality Bound is very fast. Indeed, the construction of a Local Causality Graph can be done with a polynomial algorithm in the number of automata in the model, and each vertex of the graph needs to be processed only once in order to compute the bound. However, the examples introduced in subsection 4.3 show that the approximation induced by the Local Causality Graph may be too rough for the computation of the bound, especially when the graph contains cycles.

On the first hand, example 9 shows that some cycles can be treated by removing vertices when certain conditions are satisfied. Such modifications lead to an alternative Local Causality Graph, in which the Local Causality Bound remains correct. We suggest that the conditions allowing to remove some vertices can be elaborated by reasoning on the shape of the objective trees (i.e. not all objective trees can be associated to minimal solutions). For instance, in example 9 we could see that an objective $a_0 \rightsquigarrow a_1$ cannot be an undirect successor of itself, unless there is at least one other objective $a_1 \rightsquigarrow a_*$.

On the second hand, example 10 shows a more difficult graph where removing any vertex from the cycle does not lead to a correct bound. We saw that in this example, the minimal solution contains multiple occurences of several transitions, which explains why all the vertices of the Local Causality Graph are mandatory to create the association with the objective tree (thus preventing the possibility to break the cycle). In this situation, refinements can still be considered by duplicating some vertices, in order to reveal simpler cycles that can be removed from the graph. Besides, we may also assume that these latter cases would not be frequent in practice. Indeed, as we saw in example 10, a very particular combination of several automata is necessary to obtain this configuration. Although such a configuration may be theoretically possible with a larger number of automata, this requires a strong connectivity between them, complicating the reachability at the same time. Moreover, we may assume that Automata Networks from biological applications remain sparse, which can be justified by the fact that the necessary and sufficient conditions previously proposed were already successful on concrete examples [Folschette et al., 2015]. For these reason, we do not expect a duplication strategy to significantly increase the size of a Local Causality Graph in practice.

5 Conclusion

To conclude, we have presented in this work a new approach combining static analysis and Bounded Model Checking to prove unreachability in Automata Networks. We have introduced our main result which is a property on a sufficient length of solution sequences of the reachability problem. This property can be used to compute a completeness threshold in acyclic Local Causality Graphs. We have also suggested how to put this property into practice through a SAT encoding presented in appendix A. Thanks to this property and to the encoding, the bound can be used to prove unreachability. We also provide an implementation online of the bound computation from the Local Causality Graph and of the SAT encoding. This implementation has been illustrated through simple examples. Some specific cases with cyclic Local Causality Graphs have also been presented. These examples show the current limitations of our method

and bring a new understanding about the reachability instances and the Local Causality Graphs in Asynchronous Automata Networks. Indeed, additionally to the elaboration of the bound, our property helps to strengthen the link between the static analysis techniques already developed and the solutions of the reachability instances, thus opening the door to various extensions.

Further improvements will now require to refine the Local Causality Graph by deriving additional properties on its structure. This will imply some trade-off between the computational efficiency of LCG-based methods and the coverage of all possible instances of Automata Networks.

Competing interests: The authors declare none

References

[Assmann and Albert, 2009] Assmann, S. M. and Albert, R. (2009). Discrete Dynamic Modeling with Asynchronous Update, or How to Model Complex Systems in the Absence of Quantitative Information. In Belostotsky, D. A., editor, *Plant Systems Biology*, volume 553, pages 207–225. Humana Press, Totowa, NJ.

[Balint et al., 2015] Balint, A., Belov, A., Järvisalo, M., and Sinz, C. (2015). Overview and analysis of the SAT challenge 2012 solver competition. *Artificial Intelligence*, 223:120 – 155.

[Ben Abdallah et al., 2017] Ben Abdallah, E., Folschette, M., Roux, O., and Magnin, M. (2017). ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. *Algorithms for Molecular Biology*, 12(1):20.

[Biere et al., 2009] Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands.

[Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[Burch et al., 1990] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. (1990). Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science (LICS 1990)*, pages 428–439. IEEE Computer Society Press.

[Chatain et al., 2018] Chatain, T., Haar, S., and Paulevé, L. (2018). Boolean Networks: Beyond Generalized Asynchronicity. In Baetens, J. M. and Kutrib, M., editors, *Cellular Automata and Discrete Complex Systems*, volume 10875, pages 29–42. Springer International Publishing, Cham.

[Cheng et al., 1995] Cheng, A., Esparza, J., and Palsberg, J. (1995). Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1):117 – 136.

[Clarke, 2008] Clarke, E. M. (2008). *The Birth of Model Checking*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Dubrova and Teslenko, 2011] Dubrova, E. and Teslenko, M. (2011). A sat-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1393–1399.

[Folschette et al., 2015] Folschette, M., Paulevé, L., Magnin, M., and Roux, O. (2015). Sufficient conditions for reachability in automata networks with priorities. *Theoretical Computer Science*, 608:66 – 83. From Computer Science to Biology and Back.

[Gallet et al., 2014] Gallet, E., Manceny, M., Le Gall, P., and Ballarini, P. (2014). An LTL Model Checking Approach for Biological Parameter Inference. In Merz, S. and Pang, J., editors, *Formal Methods and Software Engineering*, volume 8829, pages 155–170. Springer International Publishing, Cham.

[Giacobbe et al., 2017] Giacobbe, M., Guet, C. C., Gupta, A., Henzinger, T. A., Paixão, T., and Petrov, T. (2017). Model checking the evolution of gene regulatory networks. *Acta Informatica*, 54(8):765–787.

[Heath et al., 2008] Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. (2008). Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257.

[Klarner et al., 2012a] Klarner, H., Siebert, H., and Bockmayr, A. (2012a). Time Series Dependent Analysis of Unparametrized Thomas Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(5):1338–1351.

[Klarner et al., 2012b] Klarner, H., Streck, A., Šafránek, D., Kolčák, J., and Siebert, H. (2012b). Parameter Identification and Model Ranking of Thomas Networks. In Gilbert, D. and Heiner, M., editors, *Computational Methods in Systems Biology*, volume 7605, pages 207–226. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Nabli et al., 2016] Nabli, F., Martinez, T., Fages, F., and Soliman, S. (2016). On enumerating minimal siphons in petri nets using clp and sat solvers: theoretical and practical complexity. *Constraints*, 21(2):251–276.

[Naldi et al., 2018] Naldi, A., Hernandez, C., Levy, N., Stoll, G., Monteiro, P. T., Chaouiya, C., Helikar, T., Zinovyev, A., Calzone, L., Cohen-Boulakia, S., Thieffry, D., and Paulevé, L. (2018). The CoLoMoTo Interactive Notebook: Accessible and Reproducible Computational Analyses for Qualitative Biological Networks. *Frontiers in Physiology*, 9.

[Ogata et al., 2004] Ogata, S., Tsuchiya, T., and Kikuno, T. (2004). SAT-based verification of safe Petri nets. In Wang, F., editor, *Automated Technology for Verification and Analysis*, pages 79–92, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Paulevé, 2017] Paulevé, L. (2017). Pint: a static analyzer for transient dynamics of qualitative networks with IPython interface. In *CMSB 2017 - 15th conference on Computational Methods for Systems Biology*, volume 10545 of *Lecture Notes in Computer Science*, pages 309–316. Springer International Publishing.

[Paulevé, 2018] Paulevé, L. (2018). Reduction of Qualitative Models of Biological Networks for Transient Dynamics Analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(4):1167–1179.

[Plaisted and Greenbaum, 1986] Plaisted, D. A. and Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293 – 304.

[Shen-Orr et al., 2010] Shen-Orr, S. S., Tibshirani, R., Khatri, P., Bodian, D. L., Staedtler, F., Perry, N. M., Hastie, T., Sarwal, M. M., Davis, M. M., and Butte, A. J. (2010). Cell type–specific gene expression differences in complex tissues. *Nature methods*, 7(4):287–289.

[Taou et al., 2018] Taou, N. S., Corne, D. W., and Lones, M. A. (2018). Investigating the use of boolean networks for the control of gene regulatory networks. *Journal of Computational Science*, 26:147 – 156.

[Van Giang and Hiraishi, 2020] Van Giang, T. and Hiraishi, K. (2020). An efficient method for approximating attractors in large-scale asynchronous boolean models. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1820–1826. IEEE.

[Wichmann et al., 1995] Wichmann, B., Canning, A., Marsh, D., Clutterbuck, D., Winsborrow, L., and Ward, N. (1995). Industrial perspective on static analysis. *Software Engineering Journal*, 10(2):69.

[Češka et al., 2014] Češka, M., Šafránek, D., Dražan, S., and Brim, L. (2014). Robustness analysis of stochastic biochemical systems. *PLOS ONE*, 9(4):1–23.

A SAT encoding for Automata Networks

Bounded Model Checking can be used to verify temporal logic properties on transition systems. In Bounded Model Checking, properties are verified through the execution of the model up to some k transitions. This approach has been used successfully on large scale systems ([Biere et al., 2009]) and has overcome the use of Binary Decision Diagrams. In this section, an encoding of the dynamics of Asynchronous Automata Networks into a propositional logic formula is presented.

The goal is to use solvers for the SAT problems, which consists in deciding if there is an assignment which makes such a logic formula satisfiable. Indeed, the growing efficiency of these solvers makes them useful to tackle all type of combinatorial problems and they have been widely used for Bounded Model Checking ([Balint et al., 2015]).

A.1 Global states and execution sequence

To encode the execution of an Automata Network, logic variables should be defined to represent the global states. In this encoding, local states of an automaton are represented with one variable per local state. If a local state is active, the corresponding variable will be equal to *true* and if it is not, the variable will be equal to *false*.

Definition 12 *Global state encoding*
Given an Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$, a local state of an automaton $x \in \Sigma$ is represented as a tuple $v_x = (v_{x_1}, v_{x_2}, .., v_{x_n})$ with:
$v_{x_i} = \begin{cases} \top \text{ if } x_i \text{ is active} \\ \bot \text{ otherwise} \end{cases}$
and verifying $(v_{x_i} = \top) \implies (v_{x_j} = \bot) \quad \forall j \neq i$
A global state $s \in \mathcal{S}$ is represented as a tuple $v = (v_x, v_y, .., v_z)$ with one group of variables for each automaton.

To use Bounded Model Checking on Automata Networks, it is necessary to represent a sequence of global states of the model. To do so, the variables which represent the global states are duplicated and ordered. The sequence $(s_0, s_1, .., s_n)$ is then represented by the sequence of variables: $(v_1 = (v_{x,1}, .., v_{z,1}), v_2 = (...), .., v_n)$.

A.2 Transition constraints

In order to perform Bounded Model Checking with SAT, the dynamics of Asynchronous Automata Networks has to be encoded as a propositional logic formula. To do so, constraints are used to link the global states of the sequence between each other. These constraints force the logic variables to take only valid values, according to the dynamics of Automata Networks.

Every change between two global states means that one (and only one) transition has been played. Then, if an automaton is modified, the expression must verify that a transition has been played. To do so, the following verifications are necessary:

- there is a transition corresponding to the change
- all the conditions of this transition are verified on the origin global state
- only one local state variable per automaton is *true* in the next global state
- only one automaton is modified between the two states (asynchronicity)

In this encoding, the playability of a transition is checked for every local state of every automaton. This verification is encoded through an implication: if a local state is activated, then a corresponding transition has been played (or this local state was already activated). The formulation used in this encoding is based on a weak constraint, which means that every change must be verified but the absence of change between two consecutive states of a sequence is also valid. However, it does not complicate the search of reachability sequences since a valid sequence can be obtained from the encoded sequence.

Definition 13 *Transition constraint for local state x_j*
Given an Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$, an encoding variable sequence v and a transition $\tau = x_i \xrightarrow{c} x_j \in \mathcal{T} \in \mathcal{M}$, the validity of the transition τ played to activate x_j is encoded by the logic expression:
- $\phi_{\tau,k} := v_{x_i,k} \wedge \left(\bigwedge_{y_i \in enab(\tau)} v_{y_i,k} \right) \wedge \left(\bigwedge_{x_l \neq x_j} \neg v_{x_l,k+1} \right)$
To ensure that there is no change in an automaton, the following expression is used:
- $\psi_{z,k} := \bigwedge_{z_i \in \mathcal{S}_z} (v_{z_i,k} \rightarrow v_{z_i,k+1})$
The encoding of the playability of any transition to activate x_j is made by:
- $\phi_{x_j,k} := v_{x_j,k+1} \rightarrow \left(\left(\left(\bigvee_{\tau : dest(\tau) = x_j} \phi_{\tau,k} \right) \wedge \psi_{x,k} \right) \vee v_{x_j,k} \right)$

The expression $\phi_{\tau,k}$ is used to check that a transition τ can be played between the two global states s_k and s_{k+1}. The expression $\psi_{z,k}$ is additionally used to check that the local state of an automaton y has not changed between the global states s_k and s_{k+1}, which is necessary for the asynchronicity. Finally, the overall constraint $\phi_{x_j,k}$ check that if a local state x_j is activated at step $k+1$, then either a transition τ has been played (and the other automata has not changed) or this local state was already active at step k. The overall constraint between two global states combine these constraints for every local states, for every automata.

Definition 14 *Constraint between s_k and s_{k+1}*
Given an Automata Network $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ and an encoding variable sequence v, the global constraint between s_k and s_{k+1} is:
- $\Phi_k := \left(\bigwedge_{x_j \in \mathcal{S}_x, x \in \Sigma} \phi_{x_j,k} \right)$

In practice, one can notice that there is a lot of duplication of the $\psi_{z,k}$ constraints. Indeed, with these definitions, the asynchronicity constraint of an automaton z will be duplicated in every local state of every automaton $x \neq z$. Instead, it is possible to encode this constraint in a variable and then make a reference to this variable wherever this constraint is necessary. This makes the logic expression shorter.

A.3 Reachability analysis

In order to verify the reachability we need a final constraint. If we want to check the activation of a local state x_i in the last state of the sequence, we only need to check if the corresponding variable is *true*. This can be done by a simple conjunction on the formula: $\Phi \wedge v_{x_i,n}$. Thanks to this procedure, it is possible to check the reachability not only at step n of the sequence but also for the other global states. Indeed, the constraints let the formula be satisfiable even if there is no change between two consecutive global states. So, if the tested reachability has only solution sequences of size $k < n$, then some global states would be identical at least $n - k$ times in the SAT solution.

In general, SAT solvers take as input a Conjunctive Normal Form expression. To transform the logic expression into this form, we proceeded as in [Plaisted and Greenbaum, 1986]. This transformation changes the size of the formula by adding variables. However, the size of the new formula remains polynomial in the size of the original one.

B Bounded Model Checking and completeness threshold

In Bounded Model Checking, properties are usually verified for fixed length execution sequences of the models. However, extensions have been introduced in order to provide completeness. These extensions are presented in [Biere et al., 2009]. Some completeness bounds can be directly derived from a Bounded Model Checking encoding. For example, the *k-induction* bound can be computed by adding few modifications to the encoding proposed above. In this section, we discuss about the relevance of the Local Causality Bound regarding the *k-induction* bound.

B.1 *k-induction* in Bounded Model Checking

As seen in appendix A, the encoding can be used to check properties in fixed length execution sequences of the model. By creating logic variables v_i for each state i of the execution sequence and by linking them

with the logic expressions ϕ_k, one is able to represent the evolution of an Automata Network.

Some additional constraints can also be used to find a completeness threshold. To do so, it is possible to remove the constraint of the initial state. The resulting sequence is able to check the existence of a path from any state of the model to the state verifying the reachability. If the SAT solver returns *UNSAT* for this sequence, it proves that no sequence of that length, or longer, can reach the goal state. Indeed, the sequence will remain unsatisfiable after adding another state, since the new sequence contains the same clauses as the first one. In particular, a sequence starting from the initial state to the goal state cannot be satisfiable, which proves the unreachability of the initial problem. Thus, the computation of the bound can be done by increasing the length of that sequence until it is not satisfiable. Unfortunately, execution sequences can contain infinite loops. An alternative is the verification of the shortest longest sequence to the goal state. The shortest sequence means a sequence with no loops, which can be done by checking that all the states in the sequence are different. The definition 15 shows how to compute such a constraint. This constraint makes sure that at least one automaton is different between two global states.

Definition 15 *State inequality ($s_i \neq s_j$)*
To check the inequality of two states, the following constraint is used:
- $\psi_{i,j} := \bigvee\limits_{x \in \Sigma} \left(\bigwedge\limits_{x_l \in \mathcal{S}_x} (v_{x_l,i} \rightarrow \neg v_{x_l,j}) \right)$

This additional constraint increases the length of the sequence, which complicates the search of a satisfiable assignment. Another constraint can be added to facilitate the search. Indeed, the goal state should not be activated along the sequence, since its activation is the goal of the reachability. Then, it is possible to check it all along the sequence by adding constraints like: $\neg v_{x_i,j} \forall j$.

B.2 Comparison with the Local Causality Bound

The computation of the Local Causality Graph and the Local Causality Bound is essentially driven by the initial state of the reachability instance. Indeed, the first objectives added in the Local Causality Graph start from the initial state of the automaton. On the contrary, the initial state is not taken into account for the computation of the *k-induction* bound since the initial state constraint is removed from the sequence, which makes the solver search for every sequence that reaches the goal. By the definition of the Automata Networks, we can expect the Local Causality Bound to be shorter than the *k-induction* bound because the initial state is a strong constraint. If this constraint is removed, additional paths that reach the goal state may be found.

To briefly illustrate this difference, we have generated few random Automata Networks of limited size. To obtain a Local Causality Bound, reachability instances associated to acyclic Local Causality Graphs have also been computed thanks to random generations. These instances can be found in the file *README.md* from the implementation mentioned in section 4.1. In these examples, we can see that the *k-induction* bound is at least twice the length of the Local Causality Bound. The difference also seems to become bigger as the size of the Automata Network increases. However, random instances may not totally reflect the difference on actual biological models. Thus, more detailed comparisons with additional experiments are necessary.

C Proof of the Local Causality Bound

This section shows the detailed proofs of lemma 1 and lemma 2 for the Local Causality Bound. These results rely on the following property, already proved in [Paulevé, 2018]. This property states that the transitions of a minimal solution necessarily belong to any Local Causality Graph associated with the reachability instance. In other words, transitions of the Local Causality Graph are sufficient to any minimal solution.

Proposition 2 *Sufficient transitions for reachability*
Let $\mathcal{M} = (\Sigma, \mathcal{S}, \mathcal{T})$ be an Automata Network, $\mathcal{R} = (s_0, x_i)$ a reachability instance, $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ a Local Causality Graph associated with $\mathcal{M}$, with $\mathcal{V}_\mathcal{G} = \mathcal{L} \cup \mathcal{O} \cup \mathcal{Q} \cup \mathcal{T}'$ and $\mathcal{R}$ and let π be a minimal solution to $\mathcal{R}$.
$- \forall k \in [1; |\pi|], \pi^k \in \mathcal{V}_\mathcal{G}$, i.e. all the transitions of a minimal solution belong to the Local Causality Graph.

The proof of proposition 2 from [Paulevé, 2018] does not correspond completely to Asynchronous Automata Networks. Indeed, their framework considers the generalized semantics: several transitions can be played at the same time in an execution sequence of the model. Moreover, the model authorizes synchronized transitions, that make a change in more than one automaton. Thus, the definition of the Local Causality Graph is slightly different, in order to take into account local states of transitions that do not correspond to the bounds of an objective. However, the proof remains the same. It consists in showing that, from any solution sequence to a reachability instance, it is always possible to remove the transitions that do not belong to the Local Causality Graph. This can be done by identifying loops around these transitions. The proposition is used in the next proof, which contains all the sub-proofs of lemma 1.

Proof 1 *Proof of Lemma 1*

1. *Let $y \in \Sigma, i \in [1; |\mathcal{O}_{\pi,y}|]$. From definition 10, the transitions of $tr(\mathcal{O}_{\pi,y})$ form a local path in y. Let assume that $tr(\mathcal{O}_{\pi,y})$ does not form a local solution, this implies from definition 5 that there exists $\pi^j, \pi^l \in tr(\mathcal{O}_{\pi,y}), j \neq l$, such that $dest(\pi^j) = dest(\pi^l)$. Then by removing the loop between j and l from π, the remaining solution is still playable since $\forall k \in [j+1; l-1], enab(\pi^k) \cap \mathcal{T}_y = \varnothing$. This would imply that π is not minimal, which is a contradiction.*

2. *From definition 10, $\left(\bigcup_{o \in \mathcal{V}_\mathcal{H}} tr(o) \right) \subset \{\pi^i \mid i \in [1; |\pi|]\}$.*
   Now let us show that $\{\pi^i \mid i \in [1; |\pi|]\} \subset \left(\bigcup_{o \in \mathcal{V}_\mathcal{H}} tr(o) \right)$. Let us assume that there exists $i \in [1; |\pi|]$ with $\pi^i \in \mathcal{T}_y$, such that $\forall k \in [1; |\mathcal{O}_{\pi,y}|], \pi^i \notin tr(\mathcal{O}^k_{\pi,y})$. Since π is minimal, this implies that $\exists j \in [i+1; |\pi|]$ such that $enab(\pi^j) \cap \mathcal{T}_y \neq \varnothing$ (otherwise, it would be possible to remove π^i and keep a playable sequence). Let us assume that j is the smallest index greater than i such that $enab(\pi^j) \cap \mathcal{T}_y \neq \varnothing$. $enab(\pi^j) \cap \mathcal{T}_y \neq \varnothing$ implies that $\exists k \in [1; |\mathcal{I}_{\pi,y}|]$ such that $\mathcal{I}^k_{\pi,y} = j$. From definition 10, this implies that $\pi^i \in tr(\mathcal{O}^k_{\pi,y})$, which is a contradiction.

3. *From proposition 2, $\forall i \in [1; |\pi|], \pi^i \in \mathcal{V}_\mathcal{G}$. This implies from definition 6 that $\forall y_j \in enab(\pi^i), y_j \in \mathcal{V}_\mathcal{G}$. Then from definition 9, $\forall y \in \Sigma, \forall k \in [2; |\mathcal{I}_{\pi,y}|], \mathcal{O}^k_{\pi,y} \in \mathcal{V}_\mathcal{G}$. Moreover, since $y_j \in \mathcal{V}_\mathcal{G} \implies s_0(y) \rightsquigarrow y_j \in \mathcal{V}_\mathcal{G}$, then $\mathcal{O}^1_{\pi,y} \in \mathcal{V}_\mathcal{G}$. Lastly, $x_i \in \mathcal{V}_\mathcal{G} \implies \mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x} \in \mathcal{V}_\mathcal{G}$.*

4. *From definition 11, and since any transition of π belongs to exactly one identified objective, all objectives except $\mathcal{O}^{|\mathcal{O}_{\pi,y}|}_{\pi,x}$ have exactly one predecessor. Thus, $|\mathcal{E}_\mathcal{H}| = |\mathcal{V}_\mathcal{H}| - 1$.*
   Let us assume that $\mathcal{H}$ contains a cycle, and let $(\mathcal{O}^i_{\pi,x}, \mathcal{O}^{i'}_{\pi,y}), (\mathcal{O}^{i'}_{\pi,y}, \mathcal{O}^{i''}_{\pi,z}) \in \mathcal{E}_\mathcal{H}$ be two consecutive edges of the cycle, implying that $\mathcal{O}^i_{\pi,x} \prec_\mathcal{H} \mathcal{O}^{i''}_{\pi,z}$. From definition 10, $k > \mathcal{I}^{i'}_{\pi,y} \quad \forall \pi^k \in tr(\mathcal{O}^{i'}_{\pi,y})$. In particular, from definition 11, $\mathcal{I}^{i''}_{\pi,z} > \mathcal{I}^{i'}_{\pi,y}$.
   Similarly, $(\mathcal{O}^{i'}_{\pi,y}, \mathcal{O}^{i''}_{\pi,z}) \in \mathcal{E}_\mathcal{H} \implies k > \mathcal{I}^{i''}_{\pi,z} \quad \forall \pi^k \in tr(\mathcal{O}^{i''}_{\pi,z})$. By reasoning on consecutive objectives in the cycle, it can be shown by induction that $\mathcal{I}^{i''}_{\pi,z} < k \quad \forall \pi^k \in tr(o)$ if $o \prec_\mathcal{H} \mathcal{O}^{i''}_{\pi,z}$. In particular, $\mathcal{O}^i_{\pi,x} \prec_\mathcal{H} \mathcal{O}^{i''}_{\pi,z} \implies \mathcal{I}^{i''}_{\pi,z} < \mathcal{I}^{i'}_{\pi,y}$. Thus, we have $\mathcal{I}^{i'}_{\pi,y} < \mathcal{I}^{i''}_{\pi,z}$ and $\mathcal{I}^{i''}_{\pi,z} < \mathcal{I}^{i'}_{\pi,y}$, which is a contradiction. So $\mathcal{H}$ has no cycle, and since $|\mathcal{E}_\mathcal{H}| = |\mathcal{V}_\mathcal{H}| - 1$, it is then a tree with $\mathcal{O}^{|\mathcal{O}_{\pi,x}|}_{\pi,x}$ being its root.

5. *Let $o \in \mathcal{V}_\mathcal{H}$ an objective of $\mathcal{H}$. By using lemma 1, let $\eta_o \in lpaths(o)$ be the local solution corresponding to $tr(o)$. From definition 6, $(o, \eta_o) \in \mathcal{E}_\mathcal{G}$ and $(\eta, \eta^i) \in \mathcal{E}_\mathcal{G} \quad \forall i \in [1; |\eta|]$. Moreover, from definition 11, for each objective $y_* \rightsquigarrow y_j \in \mathcal{V}_\mathcal{H}$ such that $(o, y_* \rightsquigarrow y_j) \in \mathcal{E}_\mathcal{H}, \exists k$ such that $\pi^k \in tr(o)$ and $y_j \in enab(\pi^k)$. Then $(\eta, \pi^k) \in \mathcal{E}_\mathcal{G}$ which implies $(\pi^k, y_j) \in \mathcal{E}_\mathcal{G}$.*

These results can be used for the proof of lemma 2. The following proof is performed by structural induction on the objective tree (lemma 1.4), i.e. we show that given $o \in \mathcal{V}_\mathcal{H}$, if $\forall o' : o' \prec_\mathcal{H} o, B(o') \geq |tr(o')| + \sum_{o'' \prec_\mathcal{H} o'} |tr(o'')|$, then $B(o) \geq |tr(o)| + \sum_{o' \prec_\mathcal{H} o} |tr(o')|$.

Proof 2 *Proof of Lemma 2*

- Let $P(o)$ the property: $B(o) \geq |tr(o)| + \sum_{o' \prec_\mathcal{H} o} |tr(o')|$, with $o \in \mathcal{V}_\mathcal{H}$.

- Base case: let $o \in \mathcal{V}_\mathcal{H}$ such that $\nexists (o, o') \in \mathcal{E}_\mathcal{H}$. From lemma 1.3, $o \in \mathcal{V}_\mathcal{G}$. By using lemma 1.1,

let $\eta_o \in lpaths(o)$ be the local solution corresponding to $tr(o)$. From definition 7, we have $B(\eta_o) \geq \sum_{k\in[1;|\eta_o|]} \left(1 + \sum_{y_j \in enab(\eta_o^k)} B(y_j)\right) \geq |\eta_o|$, which implies that $B(o) \geq |\eta_o|$. Thus, since o has no successors, $P(o)$ holds.

- Inductive case: let $o \in \mathcal{V}_\mathcal{H}$ and let assume that $\forall o' : o' \prec_\mathcal{H} o, P(o')$ holds.
Similarly to the base case, lemma 1.3 implies that $o \in \mathcal{V}_\mathcal{G}$. Let $\eta_o \in lpaths(o)$ be the local solution corresponding to $tr(o)$. From definition 7,
$B(\eta_o) = \sum_{k\in[1;|\eta_o|]} \left(\sum_{y_j \in enab(\eta_o^k)} B(y_j) + 1\right) = |\eta_o| + \sum_{y_j \in enab(\eta_o^k), k\in[1;|\eta_o|]} B(y_j)$
According to lemma 1.5, $\forall (o, y_* \rightsquigarrow y_j) \in \mathcal{V}_\mathcal{H}, \exists k \in [1; |\eta_o|]$ such that $(\eta_o^k, y_j) \in \mathcal{E}_\mathcal{G}$. This implies $B(\eta_o) \geq |\eta_o| + \sum_{y_j:(o,y_*\rightsquigarrow y_j)\in\mathcal{E}_\mathcal{H}} B(y_j)$
Moreover, from definition 7:
- $B(y_j) \geq B(y_* \rightsquigarrow y_j) \quad \forall y_* \rightsquigarrow y_j \in \mathcal{V}_\mathcal{G}$, and
- $B(o) \geq B(\eta) \quad \forall \eta \in lpath(o), \forall o \in \mathcal{V}_\mathcal{G}$.
This leads to $B(o) \geq |\eta_o| + \sum_{o':(o,o')\in\mathcal{E}_\mathcal{H}} B(o')$. Then P can be applied inductively to all objectives descendant of o in $\mathcal{H}$. This leads to:
$B(o) \geq |\eta_o| + \sum_{o':(o,o')\in\mathcal{E}_\mathcal{H}} \left(|tr(o')| + \sum_{o''\prec_\mathcal{H} o'} |tr(o'')|\right)$.
$\mathcal{H}$ being a tree, we have:
$\{o' \mid o' \prec_\mathcal{H} o\} = \bigcup_{o':(o,o')\in\mathcal{E}_\mathcal{G}} \left(\{o'\} \cup \{o'' : o'' \prec_\mathcal{H} o'\}\right)$, *which implies $B(o) \geq |\eta_o| + \sum_{o':o'\prec_\mathcal{H} o} |tr(o')|$. Since $|\eta_o| = |tr(o)|$, this concludes that $P(o)$ is also verified.*

Lemma 2 can then be used to prove proposition 1 when applied to the root of the objective tree, i.e. $\mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}$. Indeed, from lemma 1.2, the objectives of $\mathcal{H}$ are able to capture all the transitions of π through their corresponding local solutions. Moreover, since $\mathcal{H}$ is a tree we have $\{\mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}\} \cup \{o' \mid o' \prec_\mathcal{H} \mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}\} = \mathcal{V}_\mathcal{H}$. Then, applying the inductive property P to $\mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}$ gives: $B(\mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}) \geq \sum_{o\in\mathcal{V}_\mathcal{H}} |tr(o)|$, which, from lemma 1.2, is equivalent to $B(\mathcal{O}_{\pi,x}^{|\mathcal{O}_{\pi,x}|}) \geq |\pi|$. Finally, since from definition 7, $B(x_i) \geq B(x_* \rightsquigarrow x_i) \quad \forall x_* \rightsquigarrow x_i \in \mathcal{V}_\mathcal{G}$, this leads to $B(x_i) \geq |\pi|$, concluding the proof of proposition 1.

D Examples with necessary and sufficient conditions

D.1 Sufficient conditions

Sufficient conditions allow to prove that the reachability is possible when additional constraints are verified on the set of solutions to a reachability instance. More especially, the conditions pass if the sub-goals required to solve the instance can be reached in any order, which is ensured when all objectives have at least one local solution. When an objective does not have any local solution, the corresponding parent local state is invalidated and the information is propagated to the parents vertices. Fig. 9 shows the Local Causality graph for the instance $\mathcal{R}_1$ of the example 6. In this Local Causality Graph, we can see that the objective $b_1 \rightsquigarrow b_0$ does not have any local solution. This invalidates all the local states, including the goal a_1. The absence of local solution for $b_1 \rightsquigarrow b_0$ means that b_0 cannot be reached from b_1. However, in the example 6, solving this objective is not necessary to reach a_1 since c_1 is reached before b_1, justifying that the reachability is actually possible.

D.2 Necessary conditions

The necessary conditions aims at verifying that at least, there exists local paths in each automata allowing to reach all the sub-goals (local states in the transitions). Thus, this verification relies on a sub Local Causality Graph that contains only objectives starting from the local states of the initial state. When no local path exists for an objective, the objective is considered as unsolvable and this information can be propagated to the parents vertices in the graph. Fig. 9 shows the sub Local Causality Graph for the example 6. In this graph, it can be noticed that every objective has a solution, meaning that the necessary

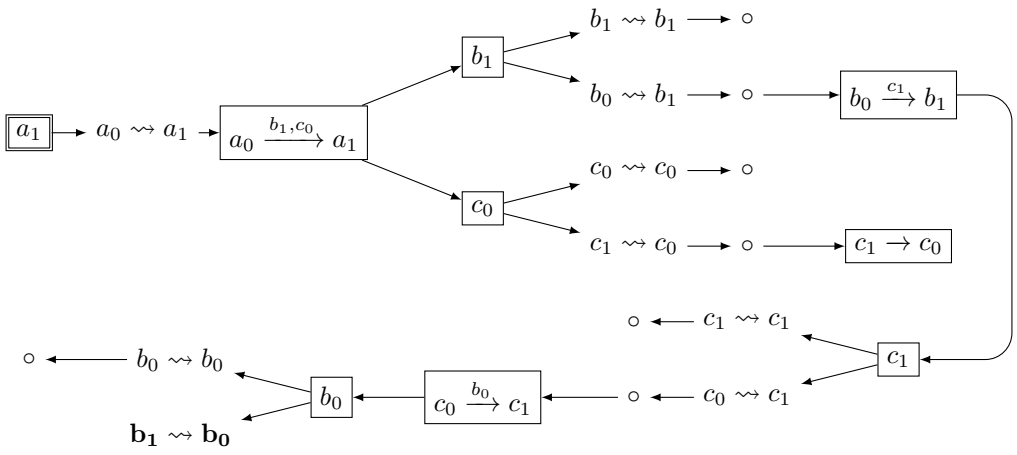

Figure 8: The Local Causality Graph for the verification of the sufficient conditions for the example 6.

conditions are verified. However, no solution exists for the reachability of d_1 since f_0 needs to be reached from f_1 after activating e_1, no transition allows to perform this change.

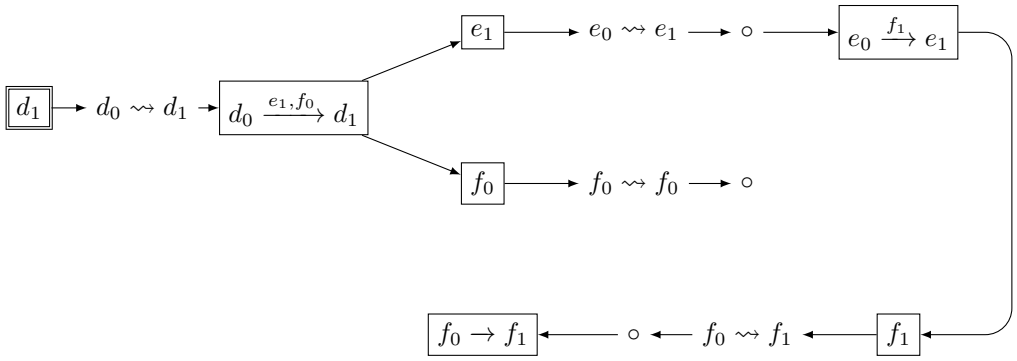

Figure 9: The Local Causality Graph made for the verification of the necessary conditions for the example 6.