SAT-Based Method for Finding Attractors in Asynchronous Multi-Valued Networks

Takehide Soh, Morgan Magnin, Daniel Le Berre, Mutsunori Banbara, Naoyuki Tamura

SAT-based Method for Finding Attractors in Asynchronous Multi-valued Networks

Takehide Soh[1][a], Morgan Magnin[2][b], Daniel Le Berre[3][c], Mutsunori Banbara[4][d], and Naoyuki Tamura[1][e].

[1]*Kobe University, Information Infrastructure and Digital Transformation Initiatives Headquarters, 1-1, Rokko-dai, Nada, Kobe, Hyogo 657-8501 Japan*

[2]*Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France*

[3]*Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France*

[4]*Nagoya University, Graduate School of Informatics, Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan*

soh@lion.kobe-u.ac.jp, morgan.magnin@ec-nantes.fr, leberre@cril-lab.fr, banbara@nagoya-u.jp, tamura@kobe-u.ac.jp

Keywords: Automata Network, Attractor, Constraint Programming, SAT.

Abstract: In this paper, we propose a SAT-based method for finding attractors of bounded size in asynchronous automata networks. The automata network is a multi-valued mathematical model which has been studied for the qualitative modeling of biological regulatory networks. An attractor is a minimal set of states in automata networks that cannot be escaped and thus loops indefinitely. Attractors are crucial to validate the initial design of a biological model and predict possible asymptotic behaviors, e.g., how cells may result through maturation in differentiated cell types. Developing an efficient computational method to find attractors is thus an important research topic. Our contribution is a translation of the problem of finding attractors of automata networks into a sequence of propositional satisfiability (SAT) problems. We also propose to add two optional constraints to improve the computation time of attractors. Experiments are carried out using 30 automata networks, 8 coming from real biological case studies and 22 crafted ones with controlled attractor size. The experimental results show that our method scales better than the state-of-the-art ASP method when the size of the attractors increases.

1 INTRODUCTION

Background. Understanding the mechanisms involved in biological regulation is a fundamental issue in analyzing living systems. The formal study of the dynamics of biological systems raises many problems, e.g., identification of attractors, bifurcations, and reachability, that are combinatorial by essence. Making these issues scalable requires designing efficient methods that rely on efficient programming frameworks. Considering this viewpoint, the long-term behavior of a regulatory network's dynamics is of specific interest (Wuensche, 1998). Such outline has been interpreted as distinct responses of the organism, such as differentiating into distinct cell types in multicellular organisms (Huang et al., 2005). Moreover, when refining a model of a living system, one way to remove inconsistencies or to predict missing information in biological models consists in comparing the attractors of the model with the experimentally observed long-term behavior. This explains why our property of interest in this paper is the computation of attractors, an attractor being a minimal set of states that cannot be escaped and thus loops indefinitely.

Automata Network. Various kinds of mathematical models have been proposed for the qualitative modeling of biological regulatory networks (BRNs). These models include neural networks, differential equations, Petri Nets, Boolean Networks (BN), probabilistic Boolean networks, and other multi-valued models such as synchronous/asynchronous Automata Networks (AN).

In this paper, we focus on a subclass of automata networks called Asynchronous Automata Networks (AAN) (Folschette et al., 2015; Paulevé, 2016a),

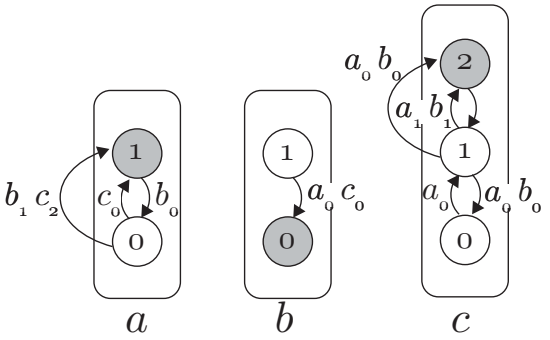
[a] https://orcid.org/0000-0001-5897-9192
[b] https://orcid.org/0000-0001-5443-0506
[c] https://orcid.org/0000-0003-3221-9923
[d] https://orcid.org/0000-0002-5388-727X
[e] https://orcid.org/0000-0002-5466-1010

which is convenient to model BRNs. In Automata Networks, biological components (e.g., genes) are represented as abstracted in the form of *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. Interaction between biological components is modeled in an atomic way by local transitions on the automaton, where each one is conditioned by a set of required local states in different automata and can modify the local state of a unique automaton. In other words, AANs allow to have multiple requirements for a local transition to occur, but *not* to synchronize several local transitions in different automata. In this sense, they are considered *Asynchronous* Automata Networks. Synchronous semantics could also be defined on such a model, but this is out of the scope of the current paper. While building a model, the choice of an appropriate semantics is of crucial importance with regard to dynamical properties. This discussion is a matter of research in itself, with many papers balancing the merits of each semantics. For some applications, like the biological ones, asynchronous semantics is said to capture more realistic behaviors in the sense that, at a given time, a single gene can change its expression level. For a more comprehensive discussion on this aspect, (Garg et al., 2008) considers the differences and respective merits of synchronous and asynchronous semantics to model biological networks and identify attractors. Note also that, depending on the chosen semantics (asynchronous or synchronous), AANs encompass the well-known Boolean frameworks of René Thomas (Thomas, 1973) and Stuart Kauffman (Kauffman, 1969).

Studies on Automata Networks. Automata networks are a very concise and convenient framework to model BRNs. Their connection with other formal models used in the Systems Biology community (like logical networks, which they generalize) has been formally proven in the past (Paulevé et al., 2012; Folschette et al., 2015). In a more practical perspective, respective translations between model frameworks have previously been implemented and integrated altogether in the CoLoMoTo initiative (Naldi et al., 2018) through the tool implementing (Paulevé, 2017). As such, the existing literature defining logical models of gene regulatory networks or signaling networks, e.g., (Grieco et al., 2013; Abou-Jaoudé et al., 2015; Sahin et al., 2009) (among many others) can benefit from results obtained on automata networks (Levy et al., 2018). Many approaches have thus been developed to analyze dynamical properties on automata networks (Paulevé et al., 2013; Paulevé, 2016b; Fitime et al., 2017; Chai et al., 2020). In (Rougny et al., 2021), the authors offer a representative example of the benefits of automata networks to capture the behavior of complex biological systems like the circadian clock while recognizing that such works would greatly benefit the analysis of long-term behaviors, i.e. attractors, untractable when too many nodes and interactions are involved.

Computing Attractors. In recent years, the design of efficient approaches to identify attractors in biological regulator networks mainly relied on the ASP framework (Gebser et al., 2012). In (Ben Abdallah et al., 2017), the authors proposed a first ASP encoding to enumerate attractors up to a given length. Their approach is iterative and without any assumption on the topology of the interaction graph. Contrastingly, the authors of (Khaled and Benhamou, 2020) enumerate attractors in asynchronous Boolean networks that have the form of a circuit using ASP by introducing a new resolution semantics that does not use the usual negation by failure. Instead, they rely on a weak version of the negation by failure that allows them to provide an enumeration approach preventing the simulation of the underlying networks. This approach targets networks with a given structure (all nodes must have an incoming edge and some results have been obtained only for cyclic interaction graphs). In this paper, we do not assume any specific structure for the networks.

Using SAT instead of ASP. If ASP solvers became very popular thanks to their versatile modeling capability, SAT solvers became in two decades de facto NP-complete oracles for solving a wide range of problems, either decision or optimization ones, on a wide range of complexity (from NP-complete to PSPACE-complete)(Biere et al., 2021) thanks to the availability of numerous, robust and increasingly more efficient solvers. The use of SAT solvers requires translating the original problem into a set of clauses, with Boolean variables. Such translation is most often ad-hoc, but can also use an intermediate modeling language for which a generic translation to SAT exists. In this paper, we use constraints with integer variables as an intermediate modeling language. We propose a constraints-based method for finding attractors of bounded size in asynchronous automata networks implemented by translation into a sequence of SAT problems. Our contribution is the translation of the problem of finding attractors of automata networks into the enumeration of the solutions of a set of constraints. Notably, we redefine attractors by using two predicates about local playability and global playability. Moreover, we propose to improve the computation time of our approach thanks



$\Sigma = \{a,b,c\}$
$S = \{a_0,a_1\} \times \{b_0,b_1\} \times \{c_0,c_1,c_2\}$
$T_a = \{(a_0,a_1,\{c_0\}),(a_0,a_1,\{b_1,c_2\}),(a_1,a_0,\{b_0\})\}$
$T_b = \{(b_1,b_0,\{a_0,c_0\})\}$
$T_c = \{(c_0,c_1,\{a_0\}),(c_1,c_0,\{a_0,b_0\}),(c_1,c_2,\{a_1,b_1\}),$
$(c_1,c_2,\{a_0,b_0\}),(c_2,c_1,\{\})\}$

Figure 1: Automata Network Example: grey is used to represent one possible state of the model, that is (a_1,b_0,c_2).

to the addition of two optional constraints. The non-Boolean constraints are finally translated to a Boolean domain to complete the translation into a SAT problem, which allows using a wide variety of very efficient SAT solvers. Experiments are carried out using 30 automata networks. The SAT approach is compared favorably to the existing ASP approach on instances containing larger-sized attractors.

2 PRELIMINARIES

2.1 Automata Networks

Definition 1. An automata network is a triple (Σ,S,T) where Σ, S and T are defined as follows:

- Σ is a set of automata. We use a symbol α to denote an automaton $\alpha \in \Sigma$.
- $S = \prod_{\alpha\in\Sigma} S_\alpha$ is the finite set of **global states**, where $S_\alpha = \{\alpha_0,\alpha_1,\ldots\}$ is a finite set of local states of an automaton $\alpha \in \Sigma$ and α_v denotes an automaton α is in the state v.
- T is a finite set of transitions. A transition can change the state of only one automaton α and is represented in the form of (α_u,α_v,C). It changes the state of the automaton α from u to v using a condition $C \subset \bigcup_{\alpha\in\Sigma} S_\alpha$. In addition, we define the transitions changing the state of an automaton α as $T_\alpha \subseteq \{(\alpha_u,\alpha_v,C) \mid \alpha_u,\alpha_v \in S_\alpha, u \neq v, C \subset \bigcup_{\alpha\in\Sigma} S_\alpha\}$.

Figure 1 shows an example of an automata network. It contains three automata named a,b,c. We focus on the automaton a to explain the example. The automaton a can be in one of the two states 0 or 1. The automaton a has three transitions $T_a = \{(a_0,a_1,\{c_0\}),(a_0,a_1,\{b_1,c_2\}),(a_1,a_0,\{b_0\})\}$. The transition $(a_0,a_1,\{b_1,c_2\})$ means that the transition changes the state of the automaton a from 0 to 1 when the two conditions b_1 – automaton b is in state 1 – and c_2 – automaton c is in state 2 – hold.

2.2 Playability in Asynchronous Automata Network

This section explains how a transition $t \in$T on a single automaton is used to define the transitions among global states $g \in S$ in an automata network (Σ,S,T).

Asynchronous Update. The state of an automaton can be updated either synchronously or asynchronously. The synchronous update is the simplest method and all automata are updated simultaneously. The asynchronous update is a more realistic model and all automata are not necessarily updated. A difference between the two update methods appears in their state transition graphs (STGs). The synchronous update is deterministic and thus the outdegree of each node in STG is at most one. The asynchronous update is nondeterministic and thus the outdegree of each node in STG is not limited. In this paper, we focus on the asynchronous update method. In particular, we allow only one automaton to update, called *generalized asynchronous*, which is known to contain many other update schemes as its special case (Gershenson, 2002).

In the following, we treat a global state g as a set rather than a tuple if it is clear from the context.

Definition 2. (Local Playability of Transition). A transition $(\alpha_u,\alpha_v,C) \in T$ is *locally playable* in a global state $g \in S$ if $\{\alpha_u\} \cup C \subseteq g$ holds.

Let g_i,g_j be two global states in S such that $g_i \neq g_j$ and $t = (\alpha_u,\alpha_v,C) \in T_\alpha$ is a transition changing the state of $\alpha \in \Sigma$.

Definition 3. (Global Playability of Transition). A transition $t = (\alpha_u,\alpha_v,C) \in T$ is *globally playable* from a global state $g_i \in S$ to a global state $g_j \in S$ such that the transition t is locally playable in the global state g_i, $\alpha_v \in g_j$, and $|g_i \setminus g_j| = 1$.

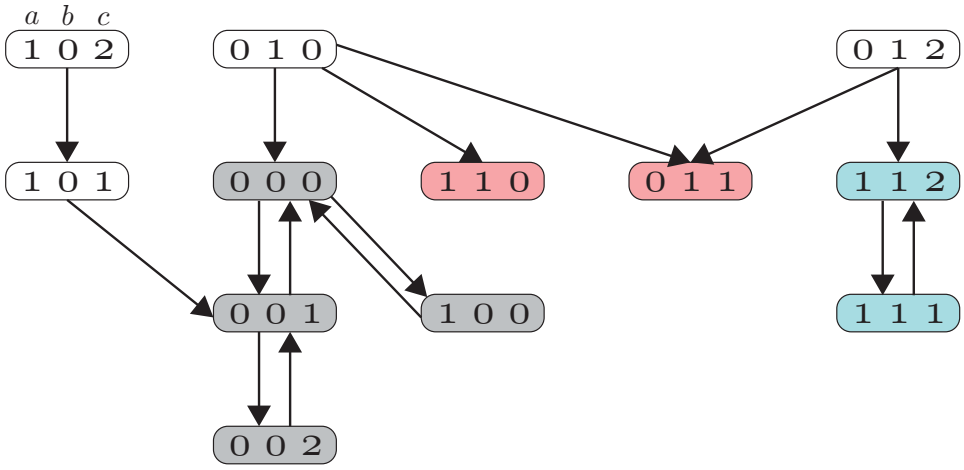


Figure 2: State Transition Graph Drawn from the Example Automata Network: red, blue, and grey are used to represent attractors of size 1, 2 and 4, respectively.

Note that the condition $|g_i \setminus g_j| = 1$ guarantees the difference between g_i and g_j is only one automaton, i.e., the automaton α whose state is changed from u to v according to the generalized asynchronous update.

2.3 Attractors

We define two predicates about the playability of a transition t with respect to global states which facilitate the understanding of the definition of trap domains and our proposed constraint model described in Section 3.2.

$$P(t, g_i) \quad \stackrel{def}{\Longleftrightarrow} \quad \{\alpha_u\} \cup C \subseteq g_i$$

$$P(t, g_i, g_j) \quad \stackrel{def}{\Longleftrightarrow} \quad P(t, g_i) \wedge \alpha_v \in g_j \wedge |g_i \setminus g_j| = 1$$

Then, trap domains and attractors are defined as follows.

Definition 4. (Trap Domain). $G \subseteq S$ is a trap domain if and only if $\forall t \in T. \forall g_i \in G. (P(t, g_i) \rightarrow \exists g_j \in G. (P(t, g_i, g_j))$

Definition 5. (Attractor (or Minimal Trap Domain)). $G \subseteq S$ is an attractor if and only if there is no $G' \subset G$ such that G' is a trap domain.

That is, attractors are trap domains that are inclusion-minimal. The *size* of an attractor G is the cardinality of G denoted as $|G|$.

Figure 2 shows the state transition graph of the automata network of Figure 1. All attractors in this network are:

- Size 1: $\{(a_0, b_1, c_1)\}, \{(a_1, b_1, c_0)\}$
- Size 2: $\{(a_1, b_1, c_2), (a_1, b_1, c_1)\}$
- Size 4: $\{(a_0, b_0, c_0), (a_0, b_0, c_1), (a_0, b_0, c_2), (a_1, b_0, c_0)\}$

In the state transition graph, attractors can be seen as strongly connected components (SCCs).

Definition 6. Bounded Attractor Enumeration Problem (BAE). BAE is the problem of finding all attractors sized less than or equal to a given size.

- **Input**: An automata network (Σ, S, T) and a bound k.
- **Output**: All attractors of the given automata network whose sizes are less than or equal to k.

Given a BAE whose inputs are the automata network of Figure 1 and $k = 1$, the output is 2 attractors. In the case of $k = 2$, the output is 3 attractors. In the case of $k = 4$, the output is 4 attractors.

3 A SAT-BASED MODEL

The proposed SAT-based approach is based on two parts. The first one is our constraint model for constructing trap domains. The second one is the incremental computation of trap domains. We ensure the minimality of trap domains by computing them starting from the smallest one.

3.1 Propositional logic recap

We consider propositional formulas in conjunctive normal form (CNF). A literal is either a propositional variable or its negation. A clause is a disjunction of literals and a CNF formula is a conjunction of clauses. An *assignment* is a mapping from a set of propositional variables to the Boolean values true or false. A propositional formula is *satisfiable* (SAT) if there

Algorithm 1: Algorithm for Computing Attractors

Data: BAE: Automata network (Σ, S, T) and a bound k
Result: All attractors of size $\leq k$

Function `Main`$((\Sigma, S, T)$ *and a bound* $k)$**:**
  $\Omega := \{\}$; `// set of trap domains`
  for $i = 1$ *to* k **do**
    $\Psi_i :=$ Construct $\Psi_i^L \wedge \Psi_i^G \wedge \Psi_i^T \wedge \bigwedge_{\omega \in \Omega}$ `BlockClauses`(ω, i) ;
    while Ψ_i *has a solution* ω **do**
      $\Omega := \Omega \cup \{f(\omega)\}$; `//` $f(\omega)$ `returns a trap domain from the solution` ω.
      $\Psi_i = \Psi_i \wedge$ `BlockClauses`(ω, i) ;
  return Ω;

Function `BlockClauses`(*Solution* ω*, Current Bound* i)**:**
  generate clauses Ψ_i^ω avoiding computation of trap domains containing a trap domain $f(\omega)$.;
  return Ψ_i^ω

exists an assignment that satisfies it and is *unsatisfiable* (UNSAT) otherwise. In addition to $\vee, \wedge, \neg$, we use Boolean operators $\leftrightarrow, \rightarrow$ which are easily translated in CNF formulas. A *SAT solver* is a program that computes a solution (if any) from a CNF formula.

Note that the constraint model in the following section is partially not propositional, i.e., we use some constraints over integer variables for clarity to the reader. Those non-Boolean constraints are surrounded by symbols $[\![]\!]$. They will be translated into Boolean constraints using the encodings in Table 1 in Section 5.

3.2 Constraint Model for Trap Domain

Variables. For each $t \in T$ and $1 \leq i \leq k$, we introduce a propositional variable $p_{t,i}$ that is true when a transition $t \in T$ is locally playable from a global state $g_i \in S$. For each $t \in T$ and $1 \leq i \neq j \leq k$, we introduce a propositional variable $p_{t,i,j}$ that is true when a transition $t \in T$ is globally playable from a global state $g_i \in S$ to a global state $g_j \in S$. For each $\alpha \in \Sigma$ and $1 \leq i \leq k$, we introduce an integer variable $x_{\alpha,i}$ that is $x_{\alpha,i} = u$ iff an automaton α is in a local state $u \in S_\alpha$ at a global state $g_i \in S$, i.e., $\alpha_u \in g_i$.

Constraints. We introduce the following constraints to compute trap domains using the variables explained. A symbol $\bullet t$ denotes the condition of the local playability of a transition t, that is, $\bullet t = \{\{\alpha_u\} \cup C\}$ when $t = (\alpha_u, \alpha_v, C)$. The first constraint is the definition of the local playability of transitions. We denote this constraint as Ψ_k^L.

$$p_{t,i} \leftrightarrow \left(\bigwedge_{\alpha_u \in \bullet t} [\![x_{\alpha,i} = u]\!] \right) \qquad (t \in T, 1 \leq i \leq k)$$

The second constraint is the definition of the global playability of transitions. We denote this constraint as Ψ_k^G.

$$p_{t,i,j} \leftrightarrow \left(p_{t,i} \wedge [\![x_{\alpha,j} = v]\!] \wedge \bigwedge_{\alpha' \in \Sigma \setminus \alpha} [\![x_{\alpha',i} = x_{\alpha',j}]\!] \right) \qquad (t = (\alpha_u, \alpha_v, C) \in T, 1 \leq i \neq j \leq k)$$

The third constraint is the definition of the trap domain. We denote this constraint as Ψ_k^T.

$$p_{t,i} \rightarrow \left(\bigvee_{\substack{j \neq i \\ 1 \leq j \leq k}} p_{t,i,j} \right) \qquad (t \in T, 1 \leq i \leq k)$$

Solutions. Let ω be any solution of $\Psi_k^L \wedge \Psi_k^G \wedge \Psi_k^T$, which are assignments to propositional or integer variables t_i, $t_{i,j}$, and $x_{\alpha,i}$ that satisfies it. In a solution ω, $x_{\alpha,i} = u$ means $\alpha_u \in g_i$. Let f be a mapping from a solution to global states. For instance, suppose that a solution ω contains $x_{a,0} = 1, x_{b,0} = 1, x_{c,0} = 2$ and $x_{a,1} = 1, x_{b,1} = 1, x_{c,1} = 1$. Then, global states $f(\omega)$ are $\{g_1, g_2\}$, where $g_1 = (a_1, b_1, c_2)$ and $g_2 = (a_1, b_1, c_1)$.

Proposition 1. Global states $f(\omega)$ from solutions ω of $\Psi_k^L \wedge \Psi_k^G \wedge \Psi_k^T$ are trap domains.

(Proof) The two propositional variables $p_{t,i}$ in Ψ_k^L and $p_{t,i,j}$ in Ψ_k^G correspond to the definition of the two predicates $P(t, g_i)$ and $P(t, g_i, g_j)$, respectively. The constraint Ψ_k^T is the same as the definition of the trap domain: $\forall t \in T. \forall g_i \in G. (P(t, g_i) \rightarrow \exists g_j (P(t, g_i, g_j))$. Thus, global states $f(\omega)$ are trap domains. □

Note that we do not need $\leftarrow$ of Ψ_k^G to solve BAE since $p_{t,i,j}$ appears only positively in Ψ_k^T.

3.3 Computing Attractors

Algorithm 1 shows an algorithm to compute attractors using the constraint model from the previous section. Input is a BAE instance, i.e., an automata network (Σ, S, T) and a bound k. Output is all attractors whose sizes are less than or equal to k. Line 2 initializes a set Ω that contains attractors. Lines 3 to 7 incrementally compute attractors from size 1 to size k. Line 5 launches a SAT solver to compute an attractor of size i. Lines 9 to 11 discard the attractors found so far. Specifically, the blocking clauses Ψ_i^{ω} are as follows.

$$\bigwedge_{1\leq j\leq i} \bigvee_{\substack{\alpha\in\Sigma \\ \alpha_u\in g}} [\![x_{\alpha,j} \neq u]\!] \qquad (g \in f(w))$$

Proposition 2. Algorithm 1 outputs all attractors sized less than or equal to k.

(Proof) We use mathematical induction on k. i) in the case of $k = 1$, Algorithm 1 outputs all trap domains sized 1. Obviously, those trap domains are minimal. Thus, the proposition holds at $k = 1$. ii) if the proposition holds at k, then Algorithm 1 outputs all trap domains sized $k+1$ and those trap domains are minimal since all trap domains less than or equal to k are enumerated and blocked. Therefore, by i) and ii), the proposition holds. □

4 Tuning Our Model

To represent the next constraints, we need to introduce new propositional variables. A propositional variable $p_{t,i\rightarrow}$ is true iff a propositional variable $p_{t,i,j}$ is true for some j. A propositional variable $p_{t,j\leftarrow}$ is true iff a propositional variable $p_{t,i,j}$ is true for some i. A propositional variable p_t is true iff propositional variables $p_{t,i\rightarrow}$ or $p_{t,i\leftarrow}$ are true for some i.

$$p_{t,i\rightarrow} \leftrightarrow \bigvee_{1\leq j\leq k} p_{t,i,j} \qquad (t \in T, 1 \leq i \leq k)$$

$$p_{t,j\leftarrow} \leftrightarrow \bigvee_{1\leq i\leq k} p_{t,i,j} \qquad (t \in T, 1 \leq j \leq k)$$

$$p_t \leftrightarrow \bigvee_{1\leq i\leq k} p_{t,i\rightarrow} \vee p_{t,i\leftarrow} \qquad (t \in T)$$

Note that the propositional variable $p_{t,i\rightarrow}$ is redundant because it is the same as the propositional variable $p_{t,i}$ in Section3.2. We rewrite here for clarity to the reader.

4.1 For attractors of size greater than 1

When we search attractors sized ≥ 2, we can add the following constraints since each global state in attractors should have at least one indegree and one outdegree in STGs.

$$\bigvee_{t\in T} p_{t,i\rightarrow} \qquad (1 \leq i \leq k)$$

$$\bigvee_{t\in T} p_{t,j\leftarrow} \qquad (1 \leq j \leq k)$$

These constraints are useful when there are a huge number of attractors sized 1 since we do not need to block each of them individually.

4.2 With optional cycle constraints

Let T'_α be a set of transitions, which changes the state of an automaton α, i.e., $T'_\alpha \subseteq T_\alpha$. We call T'_α are *cycles of transitions* in an automaton α when T'_α are (not simple, and possibly multiple) cycles.

An important property of attractors sized ≥ 2 is that these consist of (not simple, and possibly multiple) cycles of transitions in each automata T_α. Consider the attractor $\{(a_1,b_1,c_2),(a_1,b_1,c_1)\}$ in the example, it consists of the cycle of transitions of the automata c, i.e., $(c_1,c_2,\{a_1,b_1\})$ and $(c_2,c_1,\{\})$. In general, we can say the following proposition.

Proposition 3. (Transitions consisting attractors). Let T'_α be a set of transitions such that $T'_\alpha \subseteq T_\alpha$. Given an attractor G, we denote all transitions globally playable in G as $\bigcup_{\alpha\in\Sigma} T'_\alpha$. Then, for each α, T'_α are cycles of transitions or an empty set.

(Proof) We use a proof by contradiction. Suppose that the set of transitions T'_α are not cycles of transitions. Then, there is at least one state that cannot be reached by using transitions of T'_α. Since any attractors are strongly connected components in its state transition graph, it is a contradiction. □

For instance, simple cycles of transitions in Figure 1 are listed as follows.
$\gamma_1 = \{(a_0,a_1,\{c_0\}),(a_1,a_0,\{b_0\})\}$
$\gamma_2 = \{(a_0,a_1,\{b_1,c_2\}),(a_1,a_0,\{b_0\})\}$
$\gamma_3 = \{(c_0,c_1,\{a_0\}),(c_1,c_0,\{a_0,b_0\})\}$
$\gamma_4 = \{(c_1,c_2,\{a_1,b_1\}),(c_2,c_1,\{\})\}$
$\gamma_5 = \{(c_1,c_2,\{a_0,b_0\}),(c_2,c_1,\{\})\}$
Then, all sets of transitions globally playable in attractors are combinations from the following list.
$T'_a \in \{\{\},\gamma_1,\gamma_2,\gamma_1\cup\gamma_2\}$
$T'_b \in \{\{\}\}$
$T'_c \in \{\{\},\gamma_3,\gamma_4,\gamma_5,\gamma_3\cup\gamma_4,\gamma_3\cup\gamma_5,\gamma_4\cup\gamma_5,\gamma_3\cup\gamma_4\cup\gamma_5\}$
For instance, all transitions globally playable in the attractors sized ≥ 2 in the example are as follows.

- $\{(a_1,b_1,c_2),(a_1,b_1,c_1)\}$
  - $T'_a = \{\}$ and $T'_c = \gamma_4$.
- $\{(a_0,b_0,c_0),(a_0,b_0,c_1),(a_0,b_0,c_2),(a_1,b_0,c_0)\}$

– $T'_a = \gamma_1$ and $T'_c = \gamma_3 \cup \gamma_4$.

Using Proposition 3, we can add the following optional constraints. Let Γ be the set of simple cycles in a given automata network. We introduce a propositional variable p_γ that is true if a cycle $\gamma \in \Gamma$ is contained in a solution. We add the following constraints.

$$p_\gamma \rightarrow \bigwedge_{t \in \gamma} p_t \qquad (\gamma \in \Gamma)$$

$$p_t \rightarrow \bigvee_{\substack{\gamma \in \Gamma \\ t \in \gamma}} p_\gamma \qquad (t \in T)$$

The first constraint ensures that if a cycle γ is contained in a solution then all its transitions also belong to the solution. The second constraint ensures that if a transition t is playable then one of the cycles γ that contain the transition t belongs to the solution.

4.3 With optional ordering constraints

Solutions of the basic constraints discussed in Section 3.2 contain many symmetries. In the case of an attractor $\{(a_1,b_1,c_2),(a_1,b_1,c_1)\}$ from the example in Figure 2, two solutions can be computed. The first one ω_1 is:

- $x_{a,0} = 1, x_{b,0} = 1, x_{c,0} = 2$ (i.e., (a_1,b_1,c_2))
- $x_{a,1} = 1, x_{b,1} = 1, x_{c,1} = 1$ (i.e., (a_1,b_1,c_1))

The second one ω_2 is:

- $x_{a,0} = 1, x_{b,0} = 1, x_{c,0} = 1$ (i.e., (a_1,b_1,c_1))
- $x_{a,1} = 1, x_{b,1} = 1, x_{c,1} = 2$ (i.e., (a_1,b_1,c_2))

Those two solutions are identical in the sense that both solutions represent the same attractor. The transitions are simply not in the same order.

Using the current constraints, the solver could find any of them. The other one will be later discarded by the blocking clauses.

It is possible to force the order of the transitions, i.e. to represent each attractor only once. This is a standard technique in constraints programming called "symmetry breaking" (Crawford et al., 1996).

Given a sequence of global states $g_1, g_2, \ldots, g_k$, we add constraints forcing $g_1 \prec g_2 \wedge \cdots \wedge g_{k-1} \prec g_k$, where $g_i \prec g_{i+1}$ means the global state g_i is lexicographically strictly smaller than g_{i+1}. Let Σ be $\{a^1, a^2, \ldots a^n\}$. Then the constraints $g_i \prec g_{i+1}$ are defined as follows. Note that $\lhd$ is $<$ if $j = n$ and $\leq$ otherwise.

$$[\![x_{a^j,i} \leq x_{a^j,i+1}]\!] \qquad (j = 1)$$

$$\left(\bigwedge_{1 \leq m \leq j-1} [\![x_{a^m,i} = x_{a^m,i+1}]\!] \right) \rightarrow [\![x_{a^j,i} \lhd x_{a^j,i+1}]\!] \qquad (2 \leq j \leq n)$$

Table 1: Encoding Constraints

Constraints	Clauses
$[\![x = v]\!]$	$p_{x=v}$
$[\![x = x']\!]$	$\bigwedge_{\substack{d \in D(x) \\ d' \in D(x') \\ d = d'}} p_{x=d} \leftrightarrow p_{x'=d'}$
$[\![x < x']\!]$	$\bigwedge_{\substack{d \in D(x) \\ d' \in D(x') \\ \neg(d < d')}} \neg p_{x=d} \vee \neg p_{x'=d'}$
$[\![x \leq x']\!]$	$\bigwedge_{\substack{d \in D(x) \\ d' \in D(x') \\ \neg(d \leq d')}} \neg p_{x=d} \vee \neg p_{x'=d'}$

By this constraint, only ω_2 can be a solution. It is known that symmetry breaking constraints are often effective for unsatisfiable instances since it reduces search spaces.

5 Encoding Constraints in CNF

The constraints in Section 3.2 and 4 are almost propositional constraints but we need to encode some integer constraints surrounded by $[\![]\!]$. There have been many encoding methods that encode integer constraints into propositional clauses: direct encoding (de Kleer, 1989; Walsh, 2000), log encoding (Iwama and Miyazaki, 1994; Gelder, 2008), order encoding (Crawford and Baker, 1994; Tamura et al., 2009), and hybrid encoding (Soh et al., 2017). The benchmarks used in the next section contain automata with at most 3 states, and biological systems rarely have more than 4 states (Folschette et al., 2015). Thus, we use the direct encoding because it is simple and efficient when there are few integer or domain values to encode.

Let D be a mapping from a variable to its domain values. Let $p_{x=d}$ be a propositional variable that is true when $x = d$ holds, where $d \in D(x)$. We encode integer variables' axiom, a variable is assigned exactly one value, as follows.

$$\bigvee_{d \in D(x)} p_{x=d} \qquad (x \in X)$$

$$\bigwedge_{\substack{d,d' \in D(x) \\ d \neq d'}} \neg p_{x=d} \vee \neg p_{x=d'} \qquad (x \in X)$$

Using the propositional variable $p_{x=d}$, integer constraints are encoded as summarized in Table 1. Note that the translation of symmetry breaking constraints in Section 4.3 can be done in linear size by using Tseitin translation (Tseitin, 1968).

Table 2: Number of Constraints and Clauses

Name	#Constraints	#Clauses
Ψ_k^L	$O(\|T\|\|k\|)$	$O(\|T\|\|k\|\|\bullet t\|)$
Ψ_k^G	$O(\|T\|\|k\|^2\|\Sigma\|)$	$O(\|T\|\|k\|^2\|\Sigma\|\|S_\alpha\|^2)$
Ψ_k^T	$O(\|T\|\|k\|)$	$O(\|T\|\|k\|)$
Cycle	$O(\|T\|)$	$O(\|T\|)$
Sym. Break.	$O(\|k\|\|\Sigma\|)$	$O(\|k\|\|\Sigma\|\|S_\alpha\|^2)$

Table 2 shows the order of the number of constraints and clauses. We here assume the followings to ease the discussion: $|S_\alpha|$ and $|\bullet t|$ are constant values; $|T| > |\Gamma|$; $|\Gamma| \gg |\gamma|$. The table shows us that the encoding is dominated by the definition of globally playable transitions when k becomes large.

6 Experiments

To check the performance of the proposed SAT-based method, we carried out an experimental comparison against the state-of-the-art ASP-based method from (Ben Abdallah et al., 2017). Our approach is implemented using Scala 2.12.4 to generate the CNF from the automata network description. We use two SAT solvers in our implementation. For $k = 1$, we use `BDD_MINISAT_ALL` (Toda and Soh, 2016). This solver is good at problems that are easy but have many solutions. For $k > 1$, we use `CaDiCaL` (Biere et al., 2020). This solver is good at problems that are hard to solve.

6.1 Experimental Conditions

Existing Benchmark. 30 automata networks are used. The first 8 networks are all instances used in the previous study (Ben Abdallah et al., 2017), which are inspired from real organisms and found in the literature: Example (Ben Abdallah et al., 2017), Lambda phage (Thieffry and Thomas, 1995), Trp-reg (Simão et al., 2005), Fission-yeast (Davidich and Bornholdt, 2008), Mamm (Fauré et al., 2006), Tcr-sig (Klamt et al., 2006), FGF (Mbodj et al., 2013), and T-helper (Abou-Jaoudé et al., 2015). These networks contain relatively small-sized attractors.

Crafted Benchmark. We further contribute to an artificial benchmark set named `star` to check the scalability w.r.t. the number of automata, the number of transitions, and the size of cyclic attractors. This benchmark is available on the repository [1].

Given a parameter $N > 1$, `starN` is an automata network (Σ, S, T) defined as follows.

- $\Sigma = \{a^i \mid 1 \leq i \leq N\}$
- $S = \{a_0^1, a_1^1\} \times \{a_0^2, a_1^2\} \times \cdots \times \{a_0^N, a_1^N\}$
- $T = \{(a_0^i, a_1^i, \{a_0^j \mid 1 \leq j \leq N, i \neq j\}) \mid 1 \leq i \leq N\} \cup \{(a_1^i, a_0^i, \{\}) \mid 1 \leq i \leq N\}$.

Each `starN` automata network contains N automata that have two states, $2N$ transitions, and only one attractor of size $N+1$.

`star03` is given below.

- $\Sigma = \{a^1, a^2, a^3\}$
- $S = \{a_0^1, a_1^1\} \times \{a_0^2, a_1^2\} \times \{a_0^3, a_1^3\}$
- $T = T_{a^1} \cup T_{a^2} \cup T_{a^3}$
  - $T_{a^1} = \{(a_0^1, a_1^1, \{a_0^2, a_0^3\}), (a_1^1, a_0^1, \{\})\}$
  - $T_{a^2} = \{(a_0^2, a_1^2, \{a_0^1, a_0^3\}), (a_1^2, a_0^2, \{\})\}$
  - $T_{a^3} = \{(a_0^3, a_1^3, \{a_0^1, a_0^2\}), (a_1^3, a_0^3, \{\})\}$

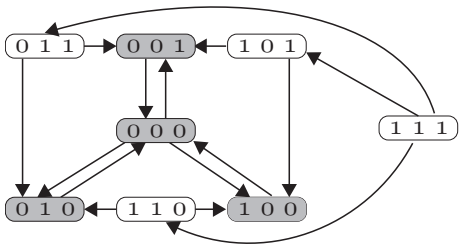


Figure 3: The State Transition Graph of `star03`

Figure 3 shows the attractor of `star03` in gray.

Bound k. For all benchmarks, the attractors are known. In this experiment, we set a bound k for each automata network so that solvers can compute the largest attractors.

Environment. We execute all experiments on the machine that equips 3GHz CPU and 16GB RAM. The time limit is 3 hours.

6.2 Results

Table 3 shows comparisons between the proposed method and the state-of-the-art existing ASP-based method (Ben Abdallah et al., 2017). The first column denotes the name of the automata networks. The second column denotes the number of automata included. The third column denotes the number of transitions included. The fourth column denotes the sizes of the attractors. The number of attractors for each size is displayed between the parenthesis. The fifth column denotes the bound k given to the solvers. The sixth column denotes the maximum number of states of automata denoted as $S_{\max}$. The seventh to tenth columns denote the CPU time of the proposed method. "Basic" denotes the basic constraint model described in Section 3.2 with at least one constraint in Section 4.1. "+C" denotes the basic constraint model with the cycle constraint discussed in Section 4.2. "+S" denotes the basic constraint model with the symmetry breaking constraint discussed in

[1] https://doi.org/10.5281/zenodo.7460388

Table 3: Comparisons between Proposed SAT-based Methods and the Existing Method

	Statictics					Proposed				Existing
Instance	$\|\Sigma\|$	$\|T\|$	Attractors	k	$\|S_{\max}\|$	Basic	+C	+S	+C+S	ASP
Example	4	12	1(3):2(1):4(1)	4	3	1.6	1.6	1.7	1.7	**0.2**
Lambda phage	4	46	1(1):2(1)	2	4	1.7	1.8	1.8	1.7	**0.1**
Trp-reg	4	14	1(2):4(1)	4	3	1.7	1.7	1.7	1.7	**0.2**
Fission-yeast	9	43	1(1)	1	3	1.4	1.4	1.5	1.4	**<0.1**
Mamm.	10	34	1(1)	1	2	1.3	1.3	1.3	1.3	**<0.1**
Tcrsig	40	85	1(8)	1	2	1.6	1.7	1.8	1.8	**<0.1**
FGF	59	102	1(1536)	1	3	1.8	1.8	1.9	1.8	**<0.1**
T-helper	101	316	1(5875504)	1	3	88.6	**86.6**	88.9	88.4	148.3
star02	2	4	3(1)	3	2	1.3	1.3	1.3	1.3	**0.1**
star03	3	6	4(1)	4	2	1.4	1.5	1.3	1.5	**0.3**
star04	4	8	5(1)	5	2	1.7	1.6	1.6	1.6	**0.6**
star05	5	10	6(1)	6	2	2.0	1.9	1.7	1.8	**1.6**
star06	6	12	7(1)	7	2	2.6	2.2	**2.1**	**2.1**	3.6
star07	7	14	8(1)	8	2	6.5	2.8	2.4	**2.3**	10.7
star08	8	16	9(1)	9	2	46.5	5.9	2.9	**2.7**	27.1
star09	9	18	10(1)	10	2	538.6	36.7	3.4	**3.2**	65.2
star10	10	20	11(1)	11	2	6598.8	442.7	4.1	**4.0**	148.0
star11	11	22	12(1)	12	2	T.O.	8787.0	**4.6**	**4.6**	1088.2
star12	12	24	13(1)	13	2	T.O.	T.O.	5.9	**4.8**	9959.3
star13	13	26	14(1)	14	2	T.O.	T.O.	7.8	**5.2**	T.O.
star14	14	28	15(1)	15	2	T.O.	T.O.	12.1	**5.9**	T.O.
star15	15	30	16(1)	16	2	T.O.	T.O.	19.8	**7.6**	T.O.
star16	16	32	17(1)	17	2	T.O.	T.O.	24.5	**9.7**	T.O.
star17	17	34	18(1)	18	2	T.O.	T.O.	38.8	**11.5**	T.O.
star18	18	36	19(1)	19	2	T.O.	T.O.	57.7	**14.5**	T.O.
star19	19	38	20(1)	20	2	T.O.	T.O.	87.7	**19.3**	T.O.
star20	20	40	21(1)	21	2	T.O.	T.O.	146.3	**28.5**	T.O.
star30	30	60	31(1)	31	2	T.O.	T.O.	6250.3	**614.4**	T.O.
star40	40	80	41(1)	41	2	T.O.	T.O.	T.O.	**7296.7**	T.O.

Section 4.3. "+C+S" denotes all constraints described in Section 3.2 and Section 4. The eleventh column denotes the CPU time of the existing ASP-based method.

In the biologically inspired benchmarks, all methods solved them within a few seconds. The main reason is that they contain small-sized attractors and thus k is also small. The difference between the proposed methods and the ASP-based method mainly comes from the implementation language: C++ and Scala. The proposed method is implemented on Scala which is running on a Java Virtual Machine (JVM) for the modeling part and thus there is a small disadvantage on this point. One exception is T-helper which contains a large number of attractors. On this problem, our approach takes advantage of the SAT solver `BDD_MINISAT_ALL` dedicated to the fast enumeration of solutions.

On `star` benchmarks, the difference between each method is more obvious. The CPU time of the ASP-based method increases exponentially, and it cannot solve `star13` within 3 hours. On the other hand, although "Basic" cannot solve `star11`, the two optional constraints successfully improve "Basic". The best results are obtained by combining those constraints.

7 CONCLUSIONS

This paper describes a SAT-based method for finding attractors of bounded size in asynchronous automata networks. Attractors are crucial to validate the initial design of a biological model and predict possible asymptotic behaviors, e.g., how cells may result through maturation in differentiated cell types. Given a bound k, we propose a constraint model of trap domains and compute attractors by incrementally computing them from 1 to k. We further propose two optional kinds of constraints to improve the computation time of our approach. The first one denotes cycle constraints which utilize a property of transitions in au-

tomata networks. The second one denotes symmetry breaking constraints which reduce redundant search spaces contained in the initial constraint model. Experimental evaluations are carried out over 30 automata networks. While the proposed SAT-based approach and the state-of-the-art ASP one could hardly be discriminated on the few existing biologically inspired benchmarks, their behavior is quite different on the crafted benchmarks we contribute. While the performance of the initial SAT-based model is not so effective, adding optional constraints allows it to scale much better than the ASP approach on benchmarks with controlled attractor size. Future work is listed as follows. Extending the comparisons to methods of other networks like Boolean network (Mori and Akutsu, 2022; Trinh et al., 2022; Rozum et al., 2021; Inoue, 2011) is necessary. Extending the proposed method to more complex networks like Petri net is interesting. Supplementary materials of this paper is available on the repository [2].

ACKNOWLEDGEMENTS

This work was financially supported by the "PHC Sakura" program (43009SC, JPJSBP120193213), implemented by the French Ministry for Europe and Foreign Affairs, the French Ministry of Higher Education, Research and Innovation and the Japan Society for Promotion of Science. This work was also supported by JSPS KAKENHI Grant Numbers JP20K11748 and JP20H05794.

REFERENCES

Abou-Jaoudé, W., Monteiro, P. T., Naldi, A., Grandclaudon, M., Soumelis, V., Chaouiya, C., and Thieffry, D. (2015). Model checking to assess t-helper cell plasticity. *Frontiers in bioengineering and biotechnology*, 2:86.

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):1–23.

Biere, A., Fazekas, K., Fleury, M., and Heisinger, M. (2020). CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M., editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki.

Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors (2021). *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Chai, X., Ribeiro, T., Magnin, M., Roux, O., and Inoue, K. (2020). Static analysis and stochastic search for reachability problem. *Electronic Notes in Theoretical Computer Science*, 350:139–158.

Crawford, J. M. and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, pages 1092–1097.

Crawford, J. M., Ginsberg, M. L., Luks, E. M., and Roy, A. (1996). Symmetry-breaking predicates for search problems. In Aiello, L. C., Doyle, J., and Shapiro, S. C., editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 148–159. Morgan Kaufmann.

Davidich, M. I. and Bornholdt, S. (2008). Boolean network model predicts cell cycle sequence of fission yeast. *PloS one*, 3(2):e1672.

de Kleer, J. (1989). A comparison of ATMS and CSP techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pages 290–296.

Fauré, A., Naldi, A., Chaouiya, C., and Thieffry, D. (2006). Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131.

Fitime, L. F., Roux, O., Guziolowski, C., and Paulevé, L. (2017). Identification of bifurcation transitions in biological regulatory networks using Answer-Set Programming. *Algorithms for Molecular Biology*, 12(1):19.

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.

Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., and De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925.

[2] https://github.com/TakehideSoh/bioinformatics2023-supplemental-material

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelder, A. V. (2008). Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243.

Gershenson, C. (2002). Classification of random boolean networks. In *Proceedings of the Eighth International Conference on Artificial Life*, pages 1–8.

Grieco, L., Calzone, L., Bernard-Pierrot, I., Radvanyi, F., Kahn-Perles, B., and Thieffry, D. (2013). Integrative modelling of the influence of mapk network on cancer cell fate decision. *PLoS computational biology*, 9(10):e1003286.

Huang, S., Eichler, G., Bar-Yam, Y., and Ingber, D. E. (2005). Cell fates as high-dimensional attractor states of a complex gene regulatory network. *Physical review letters*, 94(12):128701.

Inoue, K. (2011). Logic programming for boolean networks. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 924–930.

Iwama, K. and Miyazaki, S. (1994). SAT-variable complexity of hard combinatorial problems. In *Proceedings of the IFIP 13th World Computer Congress*, pages 253–258.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467.

Khaled, T. and Benhamou, B. (2020). An asp-based approach for boolean networks representation and attractor detection. In *LPAR*, pages 317–333.

Klamt, S., Saez-Rodriguez, J., Lindquist, J., Simeoni, L., and Gilles, E. (2006). A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, 7(1):56.

Levy, N., Naldi, A., Hernandez, C., Stoll, G., Thieffry, D., Zinovyev, A., Calzone, L., and Paulevé, L. (2018). Prediction of Mutations to Control Pathways Enabling Tumour Cell Invasion with the CoLoMoTo Interactive Notebook (Tutorial) . *Frontiers in Physiology*, 9:787.

Mbodj, A., Junion, G., Brun, C., Furlong, E. E., and Thieffry, D. (2013). Logical modelling of drosophila signalling pathways. *Molecular BioSystems*, 9(9):2248–2258.

Mori, T. and Akutsu, T. (2022). Mini review attractor detection and enumeration algorithms for boolean networks. *Computational and Structural Biotechnology Journal*.

Naldi, A., Hernandez, C., Levy, N., Stoll, G., Monteiro, P. T., Chaouiya, C., Helikar, T., Zinovyev, A., Calzone, L., Cohen-Boulakia, S., et al. (2018). The colomoto interactive notebook: accessible and reproducible computational analyses for qualitative biological networks. *Frontiers in physiology*, 9:680.

Paulevé, L. (2016a). Goal-oriented reduction of automata networks. In *International Conference on Computational Methods in Systems Biology*, volume 9859 of *Lecture Notes in Bioinformatics*, pages 252–272. Springer.

Paulevé, L. (2016b). Goal-oriented reduction of automata networks. In *International Conference on Computational Methods in Systems Biology*, pages 252–272. Springer.

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, L., Andrieux, G., and Koeppl, H. (2013). Under-approximating cut sets for reachability in large scale automata networks. In *International Conference on Computer Aided Verification*, pages 69–84. Springer.

Paulevé, L., Magnin, M., and Roux, O. (2012). From the Process Hitting to Petri Nets and Back. Technical Report hal-00744807, ETH Zürich.

Rougny, A., Paulevé, L., Teboul, M., and Delaunay, F. (2021). A detailed map of coupled circadian clock and cell cycle with qualitative dynamics validation. *BMC bioinformatics*, 22(1):1–24.

Rozum, J. C., Deritei, D., Park, K. H., Gómez Tejeda Zañudo, J., and Albert, R. (2021). pystablemotifs: Python library for attractor identification and control in Boolean networks. *Bioinformatics*, 38(5):1465–1466.

Sahin, Ö., Fröhlich, H., Löbke, C., Korf, U., Burmester, S., Majety, M., Mattern, J., Schupp, I., Chaouiya, C., Thieffry, D., et al. (2009). Modeling erbb receptor-regulated g1/s transition to find novel targets for de novo trastuzumab resistance. *BMC systems biology*, 3(1):1–20.

Simão, E., Remy, E., Thieffry, D., and Chaouiya, C. (2005). Qualitative modelling of regulated metabolic pathways: application to the tryptophan biosynthesis in e. coli. *Bioinformatics*, 21(suppl 2):ii190–ii196.

Soh, T., Banbara, M., and Tamura, N. (2017). Proposal and evaluation of hybrid encoding of CSP to SAT integrating order and log encodings. *International Journal on Artificial Intelligence Tools*, 26(1):1–29.

Tamura, N., Taga, A., Kitagawa, S., and Banbara, M. (2009). Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272.

Thieffry, D. and Thomas, R. (1995). Dynamical behaviour of biological regulatory networks—ii. immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57(2):277–297.

Thomas, R. (1973). Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585.

Toda, T. and Soh, T. (2016). Implementing efficient all solutions SAT solvers. *ACM J. Exp. Algorithmics*, 21(1):1.12:1–1.12:44.

Trinh, V.-G., Hiraishi, K., and Benhamou, B. (2022). Computing attractors of large-scale asynchronous boolean networks using minimal trap spaces. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–10.

Tseitin, G. S. (1968). On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic Part II*, pages 115—-125.

Walsh, T. (2000). SAT v CSP. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, pages 441–456.

Wuensche, A. (1998). Genomic regulation modeled as a network with basins of attraction. In *Pacific Symposium on Biocomputing*, volume 3, pages 89–102.