Annals of Mathematics and Artificial Intelligence (2023) 91:713–750
https://doi.org/10.1007/s10472-023-09886-7

REGULAR SUBMISSION

Using answer set programming to deal with boolean networks and attractor computation: application to gene regulatory networks of cells

Tarek Khaled[1] · **Belaid Benhamou**[1] · **Van-Giang Trinh**[1]

Accepted: 5 July 2023 / Published online: 31 July 2023

Abstract
Deciphering gene regulatory networks' functioning is an essential step for better understanding of life, as these networks play a fundamental role in the control of cellular processes. Boolean networks have been widely used to represent gene regulatory networks. They allow to describe the dynamics of complex gene regulatory networks straightforwardly and efficiently. The attractors are essential in the analysis of the dynamics of a Boolean network. They explain that a particular cell can acquire specific phenotypes that may be transmitted over several generations. In this work, we consider a new representation of Boolean networks' dynamics based on a new semantics used in Answer Set Programming (ASP). We use logic programs and ASP to express and deal with gene regulatory networks seen as Boolean networks, and develop a method to detect all the attractors of such networks. We first show how to represent and deal with general Boolean networks for the synchronous and asynchronous updates modes, where the computation of attractors requires a simulation of these networks' dynamics. Then, we propose an approach for the particular case of circular networks where no simulation is needed. This last specific case plays an essential role in biological systems. We show several theoretical properties; in particular, simple attractors of the gene networks are represented by the stable models of the corresponding logic programs and cyclic attractors by its extra-stable models. These extra-stable models correspond to the extra-extensions of the new semantics that are not captured by the semantics of stable models. We then evaluate the proposed approach for general Boolean networks on real biological networks and the one dedicated to the case of circular networks on Boolean networks generated randomly. The obtained results for both approaches are encouraging.

Keywords Answer set programming · Logic programming · Systems biology · Gene regulatory network · Boolean network · Attractor · Cellular phenotype

Tarek Khaled and Belaid Benhamou are contributed equally to this work.

✉ Tarek Khaled
tarek.khaled@univ-amu.fr

Extended author information available on the last page of the article

Mathematics Subject Classification (2010) 68

1 Introduction

Proteins synthesized from an organism's genes are involved in cellular processes such as cells' response to changing environmental conditions, cell differentiation during an organism's development, and DNA replication before cell division. Each of these cellular processes is sensitive to the concentration of a large number of proteins. Therefore, we understand why gene expression, that is, the set of processes leading to the synthesis of these proteins, is a highly regulated phenomenon. Many proteins are involved in these different regulation stages. Gene expression is regulated by proteins from other genes' expression. The set of regulatory interactions between genes forms what is called a gene regulatory network.

A gene regulatory network is a biological system representing genes' interaction for the survival, reproduction, or death of a cell. Different approaches have been used to model and simulate gene regulatory networks [1]. Quantitative modeling is mostly used, but it needs numerical parameters that are, in most cases, challenging to obtain. Here, we use a qualitative representation that does not require to know such parameters [2] and allows to capture the fundamental properties of the dynamics of gene regulatory networks.

Boolean networks are a simple but powerful framework for modeling gene regulatory networks [3–5]. They are composed of entities corresponding to genes or proteins. Each entity takes a value *on* or *off*, meaning that the gene/protein is or is not expressed, respectively. Two genes/proteins are connected if the expression of one of them modifies the expression of the other resulting in an activation or an inhibition. From a logical point of view, a biological system can be seen as a set of interacting entities changing along discrete time. An update mode specifies the way that the entities will be updated. There are two most popular update modes for Boolean networks [6]: *synchronous* (i.e., all the entities are simultaneously updated at each step) and *asynchronous* (i.e., only one entity is selected randomly and updated at each step). It has been shown that Boolean networks correctly express and capture the dynamics of gene regulatory networks. Particularly, the attractors of Boolean networks characterize their dynamics.

In gene regulatory networks, an attractor represents the states towards which the network dynamics converges and generally corresponds to the characteristics/phenotypes observed in biological systems [4, 7]. A Boolean network will converge to an attractor, and will then remain in it unless an external force is applied [8, 9]. Thus, it is essential to identify the attractors when studying the dynamics of a network. With this demand, we introduce in this work a new logic approach to express and analyze the dynamics of Boolean networks that allows to capture all the attractors.

We use logic programming and the ASP framework [10] to represent and deal with gene regulatory networks. ASP is a well-known declarative paradigm resulting from research on logic programming and non-monotonic reasoning. Several ASP solvers [11–14] have been developed. They provide a very natural and powerful way [15] to represent knowledge bases and allow to solve efficiently various combinatorial problems. The work presented in this paper is based on the ASP approach introduced in [16, 17] that is itself based on the semantics introduced in [18]. Unlike other semantics, the used one always ensures extensions or models for consistent logic programs, which reflect the meaning of the logic programs. Some of the extensions satisfying what is called a *discriminant* condition correspond to the stable models

of the logic programs. Other ones called extra-extensions identify extra-stable models that are not considered by know classical semantics, such as the one of stable models [19].

First, we propose a method that we use to compute the attractors of general Boolean networks for both the synchronous and asynchronous update modes. Herein, the detection of attractors is done by simulating the Boolean networks' dynamics and then verifying the attractors' existence. Second, we focus on particular gene networks represented by circular graph interactions that play an essential role in biological systems and consider only the asynchronous update mode. For this case, the detection of the attractors is done without any Boolean network simulation. We will see that representing interaction graphs of circular Boolean networks as logic programs interpreted in the new semantics [18] leads to some theoretical results that we use to identify the attractors. Especially, these results reveal a similarity between the answer sets (the stable models) of the logic program representing the interaction graph and the stable configurations of its corresponding transition graph. On the other hand, we draw parallels between the extra-stable models of the logic program and the stable cycles of the transition graph. The extra-stable models of the new semantics [18] are essential in the detection of the stable cycle attractors.

It is worth noting that the present article is the revised and extended version of our previous work [20, 21]. On the one hand, we carefully revise the formal definitions and proofs, the discussions about related work, the experimental design, and all the other details presented in the preliminary papers. On the other hand, we add more new results as follows. First, we have improved our ASP encoding to make it totally declarative but focus only on stable configurations. Second, we showcase that the efficiency of this improved encoding can benefit to a broader approach [22] (also developed by our group) for computing all attractors of an asynchronous Boolean network. Finally, we verify this hypothesis by testing our method on the large real-world models used in [22].

The paper's remainder is organized as follows: We start with summarizing the preliminaries of the used ASP semantics [18] and the Boolean network framework in Section 2. We show how to represent and deal with general Boolean networks in Section 3. In Section 4, we propose an approach dedicated to the particular case of circular Boolean networks. In Section 5, we evaluate the proposed approach for general Boolean networks on real biological networks and the approach dedicated to circular networks on randomly generated networks. We also discuss several related work in Section 6. Finally in Section 7, we conclude the work and give some perspectives.

2 Preliminaries

2.1 Boolean networks

Let $V = \{v_1, \ldots, v_n\}$ be a finite set of Boolean entities $v_i \in \{0, 1\}$ representing genes in a gene regulatory network. A configuration $x = (x_1, \ldots, x_n)$ of the system is the attribution of a truth value $x_i \in \{0, 1\}$ to each element of V. The set of all configurations [23], also called *the space of configurations*, is designated by $X = \{0, 1\}^n$. Boolean networks can be seen as abstractions for gene regulatory networks where each Boolean variable x_i represents the state of the gene v_i. The true value for x_i (i.e., $x_i = 1$) means that the corresponding gene is active, the false value (i.e., $x_i = 0$) means that the corresponding gene is inactive.

The dynamics of a Boolean network is expressed by a *global transition function* f and an update mode that defines how the elements of this Boolean network are updated over time.

The global transition function f is defined as $f : X \mapsto X$ such that $x = (x_1, \ldots, x_n) \mapsto f(x) = (f_1(x), \ldots, f_n(x))$, where each function f_i [1]: $X \mapsto \{0, 1\}$ is a *local transition function* that gives the evolution of the state x_i of the gene v_i over time.

There are an infinite number of possible update modes, but the most used ones are the synchronous and asynchronous modes [6]. In the synchronous mode, all the components are concurrently updated at each step. Consequently, each state has exactly one successor. In the asynchronous mode, only one component is selected randomly and updated at each step. Thus, each state can have up to n successors. In biology, the asynchronous mode fits the actual situation better [6]. Indeed, state changes occur at variable speeds and are usually not simultaneous. However, the question which update mode is the best one is still open so far. Hence, in this work we study both synchronous and asynchronous Boolean networks.

2.1.1 Transition graphs

The Boolean network dynamics is expressed by means of a transition graph TG, defined by a transition function f and an update mode, formally:

Definition 1 Let $X = \{0, 1\}^n$ be the configuration space of a Boolean network, $f : X \to X$ be its associated global transition function and $f_i : X \to \{0, 1\}, i \in \{1, \ldots, n\}$ are the local transition functions forming the function f. The transition graph representing the dynamic of the network is the oriented graph $TG(f) = (X, T(f))$ where the set of vertices is the set of all configurations of X and the set of arcs is $T(f) \subseteq X^2$. In the synchronous update mode, an arc $(x, x') \in T(f)$ if and only if $x'_i = f_i(x), \forall i \in \{1, \ldots, n\}$. In the asynchronous update mode, an $(x, x') \in T(f)$ if and only if $\exists i \in \{1, \ldots, n\}$ such that $x'_i = f_i(x)$ and $x'_j = x_j, \forall j \in \{1, \ldots, n\} \setminus \{i\}$. An arc (x, x') is called a *self* transition if $x = x'$.

An orbit in the transition graph $T(f)$ is a sequence of configurations $(x^0, x^1, x^2, \ldots)$ such that $\forall t \geq 0$, $(x^t, x^{t+1}) \in TG(f)$. A cycle of length r is a sequence of configurations $(x^1, \ldots, x^r, x^1)$ with $r \geq 2$ whose configurations $x^1, \ldots, x^r$ are all different. We can now give the meaning of an attractor in a Boolean network. A configuration $x = (x_1, \ldots, x_n)$ of the transition graph $TG(f)$ is a *stable configuration* when $\forall x_i \in V, x_i = f_i(x)$, thus $x = f(x)$. A stable configuration $x = (x_1, \ldots, x_n)$ forms a trivial attractor of $TG(f)$. Note that the sets of stable configurations of the synchronous and asynchronous transition graphs are the same.

A sequence of configurations $(x^1, x^2, \ldots, x^r, x^1)$ forms a *stable cycle* of $TG(f)$ when $\forall t < r, x^{t+1}$ is the unique successor of x^t and x^1 is the unique successor of x^r. A stable cycle in $TG(f)$ forms a cyclic attractor. Note that a synchronous transition graph can have stable configurations or stable cycles, whereas an asynchronous transition graph can have in addition unstable cycles (i.e., there is a configuration of a cycle that has a successor outside the cycle) or loose attractors [24] (i.e., overlapping of multiple unstable cycles). In this study, we only focus on stable configurations and stable cycles.

Transition graphs are then relevant to study the dynamics of a Boolean networks. Nevertheless, the biological data emanate from observations of experiments that habitually give only correlations between genes, but nothing on the network dynamics.

[1] $f_i(x)$ represents the local change that is made to the state x_i of the gene v_i.

2.1.2 Interaction graphs

The correlations between genes in a gene network are traditionally represented by an *interaction graph* that is a directed graph where the signs (+ or −) label the arcs.

Definition 2 Let $\mathcal{N} = (V, f)$ be a Boolean network, where $V = \{v_1, .., v_n\}$ and $f = \{f_1, ..., f_n\}$. The interaction graph of $\mathcal{N}$ is the signed-oriented graph $IG = (V, I)$ where $I \subseteq V \times \{+, -\} \times V$ is the set of signed arcs. For all $v_i, v_j \in V$, there exists a positive labeled by + (resp. negative labeled by −) arc from v_j to v_i if and only if there exists a configuration $x \in X$ with $x_j = 0$ such that $f_i(x) < f_i(\overline{x}^j)$ (resp. $f_i(x) > f_i(\overline{x}^j)$), where $\overline{x}^j$ denotes the configuration y such that $y_j = 1 - x_j$ and $y_k = x_k$ for all $k \neq j$.

Remark 1 The vertices of the interaction graph depict the genes in the gene regulatory network and the arcs express the interactions between them. An arc labeled by + is said to be positive and denotes a positive interaction between genes, whereas an arc labeled by - is said to be negative and indicates a negative interaction between genes.

An interaction graph of a Boolean network is much smaller than the transition graph of this network; therefore, is straightforward. However, an interaction graph only provides the static information of a Boolean network.

Example 1 Let us consider a Boolean network having a set of vertices $V = \{v_1, v_2\}$ and a transition function f defined as $f(x_1, x_2) = (x_2, x_1 \wedge \neg x_2)$. The function f induces the interaction graph shown in Fig. 1-a. This interaction graph is associated with the global renamed transition function $f : \{1, 2\}^2 \mapsto \{1, 2\}^2$ where $f(1, 2) = (2, 1 \wedge \neg 2)$ and where each Boolean variable x_i is simply denoted by its index i. We can see that the configuration space of the network is $X = \{0, 1\}^2$. From f and X, we deduce the two transition graphs shown in Fig. 1-b and -c corresponding to the synchronous and asynchronous update modes, respectively.

The synchronous graph has two attractors including the stable configuration 00 and the stable cycle {10, 01}. The asynchronous graph has only one attractor represented by the stable configuration 00. We can also notice that the asynchronous graph contains an unstable cycle {10, 11}. It is not stable because 10 has an arc coming out from this cycle.

2.1.3 Circular Boolean networks

The particularity of circular networks has been underlined in [25]. Thomas considered asynchronous Boolean networks and assumed that a Boolean network must contain a positive

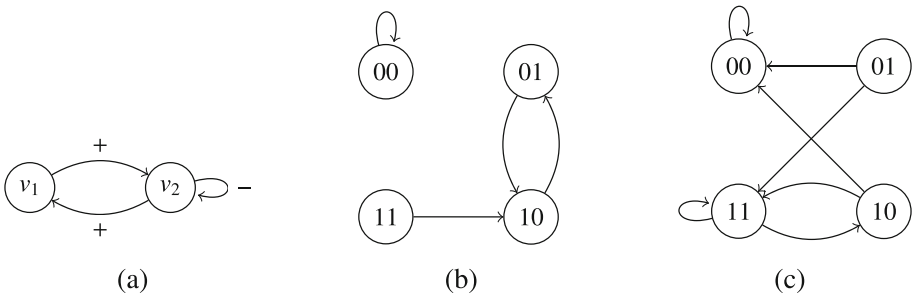


Fig. 1 The synchronous (b) / asynchronous (c) transition graphs of a Boolean network resulting from the interaction graph (a) corresponding to the transition function f

circuit (resp. a negative one) to admit several stable configurations (resp. a non-trivial attractor such as a stable cycle).

Definition 3 A circuit of the interaction graph $IG = (V, I)$ of size k is a sequence of vertices $C = (i_1, i_2, \ldots, i_k, i_1)$ such that for all $j \in \{1, \ldots, k-1\}$, $(i_j, \{+, -\}, i_{j+1})$ and $(i_k, \{+, -\}, i_1)$ are arcs of IG. If all the vertices of C are distinct, then C is said to be elementary. If the number of arcs labeled by the sign "-" (negative arcs) of an elementary circuit is even (resp. odd), then this circuit is said to be positive (resp. negative).

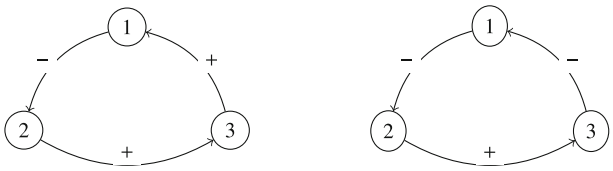
Example 2 Consider the two Boolean networks having the same set of nodes $V = \{1, 2, 3\}$ and two global transition functions f and g defined as $f(x_1, x_2, x_3) = (x_3, \neg x_1, x_2)$ and $g(x_1, x_2, x_3) = (\neg x_3, \neg x_1, x_2)$. Figure 2 shows the interaction graphs corresponding to both f and g. We can see that the function f induces a negative circuit of size 3 (Fig. 2 (a)) and g induces a positive circuit of size 3 (Fig. 2 (b)).

The configuration space here is $X = \{0, 1\}^3$. The two asynchronous transition graphs corresponding to both f and g are given in Fig. 3. We can see that for each arc (x, y) of both the transition graphs, if $x \neq y$ then the successor configuration y differs from its predecessor configuration x by a single component element. The transition graph $TG(g)$ corresponding to g has two stable configurations 100 and 011 illustrated in bold in Fig. 3 (b). Both configurations express two simple attractors that could be written as $(1, \neg 2, \neg 3)$ and $(\neg 1, 2, 3)$ when considering the corresponding nodes. The transition graph $TG(f)$ corresponding to f has a stable cycle attractor $\{000, 010, 011, 111, 101, 100\}$ formed by the six configurations pictured in bold in Fig. 3 (a). This cycle attractor could be denoted as $\{(\neg 1, \neg 2, \neg 3), (\neg 1, 2, \neg 3), (\neg 1, 2, 3), (1, 2, 3), (1, \neg 2, 3), (1, \neg 2, \neg 3)\}$ when considering the associated nodes.

An interaction graph of a single node with a positive arc forms a positive circuit of length 1. If the node is active, then it remains active permanently. If it is inactive, then it remains permanently inactive. Therefore, its transition graph will contain two stable configurations, one where the node is active and one where it is inactive. This property is also valid for interaction graphs forming a positive circuit of any size. In other words, each node in a positive circuit acts positively on itself through all the circuit interactions.

The stabilization state of gene i depends on the stabilization state of the node j preceding i in the circuit. For instance, if the interaction of j on i is positive and j has stabilized in an active state, then i should stabilize in an active state. If j has stabilized in an inactive state, then i should stabilize in an inactive state. The state of each node could then stabilize either in an active or an inactive state. Therefore, regardless of the length of the circuit, there are only two possible stable configurations (two simple attractors) for such networks.

On the other hand, an interaction graph consisting in a single self-inhibitory node forms a negative circuit of length 1. If the node is active, then it inhibits itself and it activates itself



(a) $IG(f)$ (b) $IG(g)$

Fig. 2 Interaction graph of circular positive (b) and negative graphs (a) of size 3

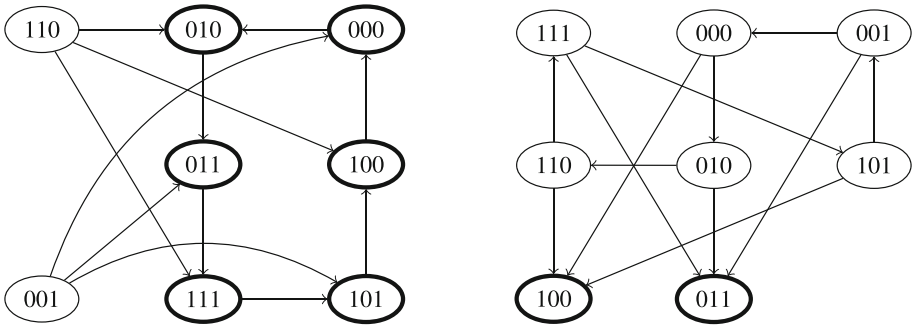


Fig. 3 Transition graphs of circular positive (b) and negative graphs (a) of size 3. For simplification, self transitions are omitted

otherwise (if it is inactive). The state of the node alternates between active and inactive. This property is preserved for interaction graphs with negative circuits of any length. Each node operates on itself through the circuit's interactions, and its state oscillates between active and inactive.

In the case of the asynchronous update mode, it has been shown in [26] that a positive circuit of size n has two attractors, namely two stable configurations x and $\neg x$ of n elements (of size n) in its corresponding transition graph. The configuration $\neg x$ is obtained from x by inverting the truth value of each element of x. It is the complementary configuration of x. On the other hand, a negative circuit of size n admits only one attractor corresponding to a stable cycle of its transition graph formed by $2n$ configurations of n elements.

From a biological point of view, the capacity to have multiple stable configurations may explain that some cells could develop specific phenotypes that could be transmitted over several generations. Stable cycles allow the expression of homeostasis phenomena. This phenomenon's role is to maintain certain critical factors around an ideal value (e.g., temperature, blood sugar level).

2.2 Answer set programming

The ASP paradigm [10] is entirely declarative, it has a high level knowledge representation capability and very powerful solvers. The basic idea of ASP is to represent the knowledge base as a set of rules constituting a logical program, then give to this program a semantics by computing for instance its stable models [19] or its answer sets [27].

A logic program π is a finite set of rules of the form

$$r : head(r) \leftarrow body(r)$$

In general, the rules are given in First-Order Logic. A grounder is used to transform the initial logic program into a ground (propositional logic) program denoted by $Ground(\pi)$ that conserves the initial program's stable models or answer sets. In the sequel, we focus on only ground programs. We shall write just π to mean $Ground(\pi)$.

There are different classes of logic programs. They differ by the presence or the absence of the classical negation and/or the default negation (or the negation as failure) in the rules of the considered program. A *positive* logic program π is a set of rules of the form

$$r : A_0 \leftarrow A_1, A_2, \ldots, A_m$$

where $m \geq 0$ and $A_{i\in\{0,\ldots,m\}}$ is an atom. There is no classical/strong negation or default negation in a positive logic program. A *normal* logic program π is a set of rules of the form

$$r : A_0 \leftarrow A_1, A_2, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n$$

where $0 \leq m \leq n$, $A_{i\in\{0,\ldots,n\}}$ is an atom and `not` is the symbol expressing the default negation. The positive body of r is $body^+(r) = \{A_1, A_2, \ldots, A_m\}$ and the negative one is $body^-(r) = \{A_{m+1}, \ldots, A_n\}$. The intuitive meaning of the rule r is the following: if we prove all the atoms of $body^+(r)$ and at the same time no atom of $body^-(r)$ had been proven, then we infer the head A_0. The positive projection of r is

$$r^+ = A_0 \leftarrow A_1, A_2, \ldots, A_m$$

An extended logic program is a set of rules of the form

$$r : L_0 \leftarrow L_1, L_2, \ldots, L_m, \text{not } L_{m+1}, \ldots, \text{not } L_n$$

where $0 \leq m \leq n$ and $L_{i\in\{0,\ldots,n\}}$ is a literal (i.e., an atom A_i or its negation $\neg A_i$). In addition to the default negation, an extended program contains the classical negation.

2.2.1 Semantics of normal programs

Various semantics are introduced to ASP to give a meaning to logic programs. The stable model semantics [19] is one of the most common semantics used in ASP.

The reduct of a normal logic program π with respect to a given set of atoms X is the positive program π^X obtained by removing from π each rule containing a literal not A_i in its negative body such that $A_i \in X$, and all the literals not A_j of the other rules. Formally,

$$\pi^X := \left\{head(r) \leftarrow body^+(r) \mid r \in \pi, body^-(r) \cap X = \emptyset\right\}$$

A set X of atoms is a *stable* model of π if and only if X is identical to the minimal Herbrand model of the reduct π^X obtained from π with respect to X. This model is also called the *canonical* model of π^X, it is denoted by $Cn(\pi^X)$. Formally, a set X of atoms is a stable model of π if and only if $X = Cn(\pi^X)$.

A new semantics that captures and extends the semantics of the stable models has been presented in [18]. This semantics uses a Horn clausal representation to express the considered logic program. This Horn representation has the same size as the one of the input logic program, it does not increase its size. This semantics is based on a classical propositional language L having two subsets of variables. The subset of variables $V := \{A_i \mid A_i \in L\}$ and the subset of *negated* variables $nV := \{\text{not } A_i \mid A_i \in L\}$. For each variable $A_i \in V$, there is a corresponding variable not $A_i \in nV$ expressing a sort of weak negation by failure of A_i. A

link between the two types of variables is expressed by adding to L an axiom expressing the mutual exclusion between the two types of variables. This axiom induces the set of binary clauses

$$ME := \{(\neg A_i \vee \neg \text{not } A_i) \mid A_i \in V\}.$$

A normal logic program $\pi = \{r : A_0 \leftarrow A_1, A_2, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n\}$ with $0 \leq m \leq n$ is expressed in the propositional language L by the set of Horn clauses

$$HC(\pi) = RC \cup ME = \left\{ \bigcup_{r \in \pi} A_0 \vee \bigvee_{i \in \{1,\ldots,m\}} \neg A_i \vee \bigvee_{i \in \{1,\ldots,n-m\}} \neg \text{not } A_{m+i} \right\} \cup \bigcup_{A_i \in V} \{\neg A_i \vee \neg \text{not } A_i\}$$

representing the union of the subset of rule clauses (RC) and the subset of mutual exclusion clauses (ME). The strong back-door[2] (STB) [28] of the logic program π is formed by the literals of the form not A_i that occur in the negative bodies of its rules. Formally, it is defined by

$$STB := \left\{\text{not } A_i \mid r \in \pi, A_i \in body^-(r)\right\}.$$

Given a logic program π and its STB, an extension of $HC(\pi)$ with respect to the STB, or simply an extension of the pair $(HC(\pi), STB)$ is the set of all consistent clauses derived from $HC(\pi)$ when adding a maximal set of negative literals not $A_i \in STB$ to $HC(\pi)$. See the formal definition at Definition 4.

Definition 4 **(adjusted from [18])** Let $HC(\pi)$ be the Horn CNF encoding of a logic program π, STB be its strong back-door, and $S' \subseteq STB$. The set $E = HC(\pi) \cup S'$ of clauses is then an extension of $(HC(\pi), STB)$ if the following conditions hold.

1. E is consistent.
2. $\forall \text{not } A_i \in STB \setminus S'$, $E \cup \{\text{not } A_i\}$ is inconsistent.

It is shown in [18] that each consistent set of clauses $HC(\pi)$ representing the logic program π admits at least an extension with respect to the corresponding STB. To be formal, see Proposition 1.

Proposition 1 ***(adjusted from** [18]**)*** *Let π be a logic program and STB be its strong back-door. If $HC(\pi)$ is consistent, then there exists at least one extension of the pair $(HC(\pi), STB)$.*

It is also shown in [18] that the set of stable models of a logic program π is in bijection with the subset of extensions E of $HC(\pi)$ that satisfy the discriminant condition: for each atom $A_i \in V$, if E infers $\neg \text{not } A_i$, then E must infer A_i. That is,

$$\forall A_i \in V, E \models \neg \text{not } A_i \Rightarrow E \models A_i.$$

The extensions that do not satisfy the discriminant condition are called extra-extensions. They will define extra-stable models that extend the classical semantics of stable models.

Two main theoretical properties are given in the two following theorems.

[2] We shall see in the sequel that the variables of the strong back-door form the main variables (the decision nodes) of the model search tree.

Theorem 1 *([18]) If X is a stable model of a logic program π, then there exists an extension E of $(HC(\pi), ST B)$ satisfying the discriminant condition such that $X = \{A_i \in V \mid E \models A_i\}$.*

Theorem 2 *([18]) If E is an extension of $(HC(\pi), ST B)$ that satisfies the discriminant condition, then $X = \{A_i \mid E \models A_i\}$ is a stable model of π.*

Example 3 Consider the logic program π including the below rules

$$q \leftarrow \text{not } r$$
$$r \leftarrow \text{not } q$$
$$p \leftarrow \text{not } p$$
$$p \leftarrow \text{not } r$$

The Horn clausal representation of the logic program π is formed by the set $HC(\pi) = RC \cup ME$ where

$$RC = \{q \vee \neg \text{not } r\ ,\ r \vee \neg \text{not } q\ ,\ p \vee \neg \text{not } p\ ,\ p \vee \neg \text{not } r\}$$
$$ME = \{\neg q \vee \neg \text{not } q, \neg r \vee \neg \text{not } r, \neg p \vee \neg \text{not } p\}$$

and its strong back-door is $ST B = \{\text{not } r, \text{not } q, \text{not } p\}$. We can see that $(HC(\pi), ST B)$ admits two extensions $E_1 = HC(\pi) \cup \{\text{not } r\}$ and $E_2 = HC(\pi) \cup \{\text{not } q\}$. Indeed, E_1 and E_2 are maximally consistent with respect to the set $ST B$. We can deduce by unit resolution that

$$E_1 \models \{\neg r, q, p, \neg \text{not } q, \neg \text{not } p\}$$

and

$$E_2 \models \{\neg \text{not } r, r, \neg q, \neg \text{not } p, \neg p\}.$$

The extension E_1 satisfies the discriminant condition, whereas E_2 does not. Thus, the logic program has one stable model $M_1 = \{p, q\}$ whose atoms are deduced from E_1 by unit resolution. On the other hand, $M_2 = \{r\}$ represents an extra-stable model deduced from the extra-extension E_2. Then, we have two models for this program, among one of them is a stable model.

Remark 2 One can remark that in the models $M_1 = \{p, q\}$ and $M_2 = \{r\}$, we just reported the positive atoms that are true in the models. The other atoms are all assumed to be false by the closed-world assumption.

2.2.2 Search method for stable models and extra-stable models

The method [16] used in this article to compute stable models and extra-stable models is based on the semantics [18] discussed in the previous section. For a given logic program π, this method computes all the extensions of $(HC(\pi), ST B)$ from which the stable models and extra-stable models are deduced. The computation of extensions of the pair $(HC(\pi), ST B)$ is done by progressively adding the literals not A_i of $ST B$ to $HC(\pi)$ and by checking the consistency of the set obtained at each node of the search tree.

The enumeration process of this method explores a Boolean tree search. It looks like the one of a DPLL [29] procedure that is adapted to ASP and to the used semantics [18]. If the computed extension satisfies the discriminant condition, then it induces a stable model. Otherwise, if the discriminant condition is not satisfied, then it corresponds to an extra-stable model. The enumeration process incrementally constructs an extension by alternating in

the search tree between deterministic nodes that correspond to unit propagation and non-deterministic nodes corresponding to choice points. The choice points are defined by the assignment of truth values (true or false) to literals not A_i within the set STB. One of the advantages of this method is that its enumeration process is carried out only on the subset of the literals belonging to the set STB. This advantage makes it possible to reduce the complexity in computational time in practice.

2.2.3 Semantics of extended programs

Normal logic programs are used to model various problems. However, it turns out that many situations require strong negation. Strong negation is essential when real problems have to be modeled declaratively. The semantics of an extended logic program is defined by a reduction to a normal program [27]. This reduction removes the strong negation. Then, we could use the semantics summarized above for normal programs [18] to deduce the answer sets of the extended logic program.

To reduce an extended logic program into an equivalent normal logic program, we replace any negative literals $\neg L$ appearing in the extended logic program by a new literal L', then add the integrity constraint rule $\leftarrow L, L'$. This rule prohibits that L and $\neg L$ to be true in the same model. We compute the stable models of the resulting normal program from which we can obtain the original extended program's answer sets.

Example 4 Let π be an extended logic program with the following rules

$$\begin{aligned} b &\leftarrow \text{not } \neg b, a \\ \neg b &\leftarrow \text{not } b \\ a &\leftarrow \text{not } \neg a \end{aligned}$$

The normal program resulting from the translation of π is π' with the following rules

$$\begin{aligned} b &\leftarrow \text{not } b', a \\ b' &\leftarrow \text{not } b \\ a &\leftarrow \text{not } a' \\ &\leftarrow a, a' \\ &\leftarrow b, b' \end{aligned}$$

The Horn clausal representation of the logic program π' is $HC(\pi') = CR(\pi') \cup ME(\pi')$ where

$$CR(\pi') = \{b \vee \neg \text{not } b' \vee \neg a, b' \vee \neg \text{not } b,\ a \vee \neg \text{not } a', \neg a \vee \neg a', \neg b \vee \neg b'\}$$

and

$$ME(\pi') = \{\neg a \vee \neg \text{not } a, \neg b \vee \neg \text{not } b, \neg a' \vee \neg \text{not } a', \neg b' \vee \neg \text{not } b'\}.$$

Its strong back-door is $STB = \{\text{not } a', \text{not } b, \text{not } b'\}$.

We can see that $(HC(\pi'), STB)$ admits two extensions $E'_1 = HC(\pi') \cup \{\text{not } b, \text{not } a'\}$ and $E'_2 = HC(\pi') \cup \{\text{not } a', \text{not } b'\}$. The two extensions E'_1 and E'_2 verify the discriminant condition. Thus, the logic program π' has two stable models $M'_1 = \{a, b'\}$ and $M'_2 = \{a, b\}$ that are deduced from E_1 and E_2, respectively. It results that the extended logic program π admits the two answer sets $M_1 = \{a, \neg b\}$ and $M_2 = \{a, b\}$.

3 Detection of attractors in general Boolean networks

We are interested here in general Boolean networks, i.e., there are no restrictions on their Boolean functions. We shall show how we detect attractors in both the synchronous and asynchronous update modes. A first and short version of this work has been presented in [20].

3.1 General approach

3.1.1 Representation of interaction graphs

First, we show how to represent an interaction graph IG associated with a Boolean network as an extended logic program denoted by P_{IG}. In other words, we are representing the global transition function associated with IG.

We first start with the rule (r_1) that encodes discrete time:

$$r_1 : time(0..t)$$

to compute the different configuration sequences of a transition graph TG associated with a given Boolean network. We need to study its behavior under a certain initial state condition. The possible number of combinations for the initial state could be significant, making the task very difficult for a manual user. This is the reason behind our decision to automate the process.

Next, we introduce the two rules r_2 and r_3 to generate all possible combinations of the initial state automatically:

$$r_2 : v_i(0) \leftarrow \text{not } \neg v_i(0)$$

and

$$r_3 : \neg v_i(0) \leftarrow \text{not } v_i(0)$$

These two rules express the fact that in the absence of $\neg v_i(0)$, we deduce $v_i(0)$ and in the absence of $v_i(0)$, we deduce $\neg v_i(0)$. These rules guarantee the choice of the active or inactive state for each gene. Consequently, all the answer sets are automatically generated for each possible initial state.

The four next rules r_4, r_5, r_6, r_7 encode the effects of one gene on another, i.e., the activation or inhibition of one gene by another. The rules r_4 and r_5 state that if the gene v_i is active (resp. inactive) at time step t, then it will activate (resp. inhibit) the gene v_j at time step $t+1$. These two rules represent the positively oriented arc $(v_i, +, v_j)$ of the interaction graph. The rules r_6 and r_7 encode the negative oriented arc $(v_i, -, v_j)$ of the interaction graph. In this case both rules express the fact that activating (resp. inhibiting) the gene v_i at time step t will inhibit (resp. activate) the gene v_j at time step $t+1$.

$$r_4 : v_j(t+1) \leftarrow v_i(t)$$
$$r_5 : \neg v_j(t+1) \leftarrow \neg v_i(t)$$
$$r_6 : v_j(t+1) \leftarrow \neg v_i(t)$$
$$r_7 : \neg v_j(t+1) \leftarrow v_i(t)$$

The following rules r_8 and r_9 express inertia, i.e., what happens if there is no change in gene state between the steps t and $t+1$. A gene maintains its state at step $t+1$ unless it has

been changed at step t.

$$r_8 : v_i(t+1) \leftarrow v_i(t), \text{not } \neg v_i(t+1)$$
$$r_9 : \neg v_i(t+1) \leftarrow \neg v_i(t), \text{not } v_i(t+1)$$

Now, we present the rules for dealing with general Boolean networks where a given gene has several interactions. We express each local transition function f_i as a set of rules. We assume that each local function f_i is given in the disjunctive normal form (DNF). Given the gene vector $v(t) = (v_1, v_2, \ldots, v_n)$ at time step t, we express for each node $v_i \in V$ of the interaction graph, its corresponding function f_i by the following DNF formula:

$$v_i(t+1) = f_i(v(t)) = \bigvee_{j=1}^{l} m_i^j$$

where m_i^j is a conjunction of literals.

Let $DNF\ (\neg f_i(v(t)) = \bigvee_{j=1}^{e} m_i'^j$ be the DNF of $\neg f_i(v(t))$. The formula m'^j_i is a conjunction of literals. The set of rules that encodes each function f_i is defined as follows:

$$r_{10} : \{v_i(t+1) \leftarrow m_i^j(t) \mid 1 \leq j \leq l\}, i \in \{1, \ldots, n\}$$
$$r_{11} : \{\neg v_i(t+1) \leftarrow m'^j_i(t) \mid 1 \leq j \leq e\}, i \in \{1, \ldots, n\}$$

Note that the rules r_{10} and r_{11} apply only when considering the synchronous update mode, where all the local transition function f_i are simultaneously applied at each time step.

Example 5 Consider the interaction graph given in Example 1. The set of rules generated by the generic rules r_{10} and r_{11} for this interaction graph are

$$1(t+1) \leftarrow 2(t)$$
$$2(t+1) \leftarrow 1(t), \neg 2(t)$$

and

$$\neg 1(t+1) \leftarrow \neg 2(t)$$
$$\neg 2(t+1) \leftarrow \neg 1(t)$$
$$\neg 2(t+1) \leftarrow 2(t)$$

respectively.

Now, we show how to encode the asynchronous update mode. The rules r_{12} and r_{13} express the existence of a state change for the gene v_i, when its state in step $t+1$ is different from its state in t.

$$r_{12} : Change(v_i, t) \leftarrow v_i(t+1), \neg v_i(t)$$
$$r_{13} : Change(v_i, t) \leftarrow \neg v_i(t+1), v_i(t)$$

The fact that the predicate $Change(v_i, t)$ is true indicates that a gene v_i has been updated.

Next, we introduce a new predicate $Block(v_i, t)$ that indicates the fact that a gene v_i is blocked for updating at time step t, i.e., only the unblocked genes could be updated.

For the asynchronous update mode, we set the rule r_{14} to allow only one gene to be updated and block all the others. This rule says that if a gene v_i is not blocked, then all the other genes v_j will be blocked.

$$r_{14} : \left\{ Block(v_i, t) \leftarrow Change(v_j, t), \text{not } Block(v_j, t) \mid \forall i \in \{1, \ldots, n\} \backslash \{j\} \right\}$$

For the asynchronous update mode, we adapt both rules r_{10} and r_{11} by involving the new predicate $Block(v_i, t)$ and obtain the two new generic rules r_{15} and r_{16}, respectively.

$$r_{15} : \{v_i(t+1) \leftarrow m_i^j(t), \text{not } Block(v_i, t) \mid 1 \leq j \leq l\}, i \in \{1, \ldots, n\}$$
$$r_{16} : \{\neg v_i(t+1) \leftarrow m'^j_i(t), \text{not } Block(v_i, t) \mid 1 \leq j \leq e\}, i \in \{1, \ldots, n\}$$

These rules state that a gene could be updated unless it is blocked.

Example 6 The set of rules generated by the generic rules r_{14}, r_{15}, and r_{16} when applied to the interaction graph of Example 1 are as follows:

$$Block(1, t) \leftarrow Change(2, t), \text{not } Block(2, t)$$
$$Block(2, t) \leftarrow Change(1, t), \text{not } Block(1, t)$$

and

$$1(t+1) \leftarrow 2(t), \text{not } Block(1, t)$$
$$2(t+1) \leftarrow 1(t), \neg 2(t), \text{not } Block(2, t)$$

and

$$\neg 1(t+1) \leftarrow \neg 2(t), \text{not } Block(1, t)$$
$$\neg 2(t+1) \leftarrow \neg 1(t), \text{not } Block(2, t)$$
$$\neg 2(t+1) \leftarrow 2(t), \text{not } Block(2, t)$$

respectively.

The rules described previously, constitute the logic program P_{IG} representing the interaction graph of a general Boolean network. We now establish the relationship between the answer sets of P_{IG} and the corresponding transition graph's configuration sequences.

Proposition 2 *Let P_{IG} be the logic program representing the interaction graph IG of a gene network having a global transition function f and $TG(f)$ be its corresponding transition graph. A tuple $x = (x^0, \ldots, x^t)$ is a sequence of configurations of $TG(f)$, if and only if $I = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t))\}$ is an answer set of P_{IG} such that the set $(v_1(i), \ldots, v_n(i))$ of literals fixed at time step $i \in \{0, \ldots, t\}$ corresponds to the state of the genes of the configuration $x^i \subseteq x$ defined at time step i in the transition graph $TG(f)$.*

Proof We prove the property by induction on the step t. First, we have to verify the property for the initial step $t = 0$. That is, we prove that $I_0 = \{v_1(0), \ldots, v_n(0)\}$ is an answer set of the logic program P^0_{IG} defined at $t = 0$, if and only if $x_0 = (x^0) = (v_1(0), \ldots, v_n(0))$ forms the initial configuration of $TG(f)$. At the initial step $t = 0$, we have $P^0_{IG} = \{r_2, r_3\} = \{v_i(0) \leftarrow \text{not } \neg v_i(0), \neg v_i(0) \leftarrow \text{not } v_i(0)\}$.

Given the initial configuration $x_0 = (v_1(0), \ldots, v_n(0))$, we show that the subset of literals $I_0 = \{v_1(0), \ldots, v_n(0)\}$ is an answer set of P^0_{IG}. For each $v_i(0)$, we have two possibilities:

1. if $v_i(0) \in I_0$, then the rule $v_i(0) \leftarrow$ belongs to the reduct $(P_{IG})^{I_0}$,
2. otherwise $\neg v_i(0) \in I_0$, and $\neg v_i(0) \leftarrow$ belongs to the reduct $(P_{IG})^{I_0}$.

We can see that the reduct program $(P_{IG})^{I_0}$, admits I_0 as a single complete minimal Herbrand model, where each variable is either proven to be true or false. That is, for each variable $v_i(0)$, I_0 contains either the positive literal $v_i(0)$ or the negative literal $\neg v_i(0)$. Thus, I_0 is an answer set of P_{IG}. For the converse, let $I_0 = \{v_1(0), \ldots, v_n(0)\}$ be an answer set of P^0_{IG}. Then I_0 is complete in the sense that for each variable $v_i(0)$ either the positive literal $v_i(0)$ or the negative one $\neg v_i(0)$ is present in I_0. Otherwise, if a variable $v_i(0)$ is missing in I_0, then the reduct $(P_{IG})^{I_0}$, will contain both rules $v_j(0) \leftarrow$ and $\neg v_j(0) \leftarrow$, which lead to an inconsistency. Since for all $i \in \{1, \ldots, n\}$, we have either $v_i(0)$ or $\neg v_i(0)$ or inactive at $t = 0$, we conclude that $x^0 = (v_1(0), \ldots, v_n(0))$ is the initial state of $TG(f)$.

Now, suppose that the property holds until a time step t. That is, A tuple $x_t = (x^0, \ldots, x^t)$ is a sequence of configurations of $TG(f)$, if and only if

$$I_t = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t))\}$$

is an answer set of P^t_{IG} such that the set of all the literals $(v_1(i), \ldots, v_n(i))$ fixed at time step $i \in \{0, \ldots, t\}$ corresponds to the state of the genes of the configuration $x^i \subseteq x$ defined at time step $i \in \{0, \ldots, t\}$ in the transition graph $TG(f)$.

We prove that the property holds at time step $t + 1$. That is, a tuple $x_{t+1} = (x^0, \ldots, x^t, x^{t+1})$ is a sequence of configurations of $TG(f)$, if and only if

$$I_{t+1} = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t)), (v_1(t+1), \ldots, v_n(t+1))\}$$

is an answer set of P^{t+1}_{IG} such that the set of all the literals $(v_1(i), \ldots, v_n(i))$ fixed at time step $i \in \{0, \ldots, t, t+1\}$ corresponds to the state of the genes of the configuration $x^i \subseteq x$ defined at the step $i \in \{0, \ldots, t, t+1\}$ in the transition graph $TG(f)$.

Given the configuration sequence $x_{t+1} = (x^0, \ldots, x^t, x^{t+1})$ of $TG(f)$, we have to prove that the set of literals $I_{t+1} = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t)), (v_1(t+1), \ldots, v_n(t+1))\}$ is an answer set of P^{t+1}_{IG} such that the set of all the literals $(v_1(t+1), \ldots, v_n(t+1))$ fixed at time step $t + 1$ corresponds to the state of the genes of the configuration $x^{t+1} \subseteq x$ defined at time step $t + 1$ in the transition graph $TG(f)$.

To do this, we need to demonstrate that $I_{t+1} = I_t \cup \{v_1(t+1), \ldots, v_n(t+1)\}$ is an answer set of P^{t+1}_{IG} which is an extension of I_t produced by the application of the rules r_{10} and r_{11}. From these rules encoding the transition function, we have $f_i(v(t)) = \bigvee_{j=1}^{l} m_i^j$ and $\neg f_i(v(t)) = \neg(\bigvee_{j=1}^{l} m_i^j) = \bigvee_{i=1}^{e} m_i'^j$. If $\bigvee_{j=1}^{l} m_i^j$ is true at time step t (resp. $\bigvee_{j=1}^{e} m_i'^j$ is true at timestep t), then $\bigvee_{j=1}^{e} m_i'^j$ is false at time step t (resp. $\bigvee_{j=1}^{l} m_i^j$ is false at time step t. By application of the rules r_{10} and r_{11}, we will have either $v_i(t+1)$ or $\neg v_i(t+1))$ assigned to the value true at time step $t + 1$ expressing the fact that the corresponding gene i is active or inactive at time step $t + 1$. This means that both general rules r_{10} and r_{11} are satisfied and the literals fixed at time step $t + 1$ corresponds to the state of the genes of the configuration $x^{t+1} \subseteq x$. By using the induction hypothesis, we can conclude that I_{t+1} is an answer set of P^{t+1}_{IG} such that the set of all the literals $(v_1(i), \ldots, v_n(i))$ fixed at time step $i \in \{0, \ldots, t, t+1\}$ correspond to the state of the genes of the configuration $x^i \subseteq x$ defined at time step $i \in \{0, \ldots, t, t+1\}$ in the transition graph $TG(f)$.

For the converse, let

$$I_{t+1} = I_t \cup \{v_1(t+1), \ldots, v_n(t+1)\}$$
$$= \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t)), (v_1(t+1), \ldots, v_n(t+1))\}$$

be an answer set of P_{IG}^{t+1}. We know by the induction hypothesis that a tuple $x_t = (x^0, \ldots, x^t)$ is a sequence of configurations of $TG(f)$ if and only if $I_t = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t))\}$ is an answer set of P_{IG}^t such that the set of all the literals $(v_1(i), \ldots, v_n(i))$ fixed at the step $i \in \{0, \ldots, t\}$ corresponds to the state of the genes of the configuration $x^i \subseteq x$ defined at the step $i \in \{0, \ldots, t\}$ in the transition graph $TG(f)$.

Now, we will prove that $x_{t+1} = (x^0, \ldots, x^t, x^{t+1})$ is a configurations sequence of $TG(f)$. To do this, we need to demonstrate that for each variable $v_i(t+1)$ at time step $t+1$ either the positive literal $v_i(t+1)$ or the negative one $\neg v_i(t+1)$ is present. This is true, because the application of the rules r_{10} and r_{11}, ensure to have either $v_i(t+1)$ or $\neg v_i(t+1))$ at time step $t+1$. Thus, I_{t+1} is complete in the sense that for each variable $v_i(t+1)$ at step $t+1$ either the positive literal $v_i(t+1)$ or the negative one $\neg v_i(t+1)$ is present in I_{t+1}.

By the induction hypothesis, we have $x_t = (x^0, \ldots, x^t)$ is a sequence of configuration. We can conclude that $x_{t+1} = (x^0, \ldots, x^t, x^{t+1})$ is a configurations sequence of $TG(f)$. □

3.1.2 Detection of attractors

The method we use to study the dynamics of a Boolean network consists of enumerating all the possible initial configurations and then carrying out a simulation based on each of them. The method lists all possible configuration sequences of the transition graph, ensuring that all attractors will be detected. We look for all configuration sequences of a given length n in the transition graph. As we can have an exponential number of configurations, the enumeration could be memory demanding for wide networks. We want to avoid the enumeration done by a naive simulation of the network dynamics. Thus, we keep a trace of the cycles already found to eliminate them during the next iterations.

The main idea of the attractor detection algorithm is the following: once a sequence of configurations is found, we verify if it includes a cycle. A cycle in a sequence of configurations is identified by looking if the last configuration occurs twice in the sequence. In the affirmative case, all the configurations between both occurrences of this configuration belong to the cycle. For the synchronous update mode, each configuration in the synchronous transition graph has a unique successor. Thus, when a sequence of configurations enters a cycle, it never leaves it. This means that each cycle in the synchronous transition graph is stable. However, in the asynchronous update mode, a configuration of the asynchronous transition graph can have several successors. Therefore, cycles of the transition graph are not necessarily stable. There could be stable cycles and unstable cycles. If no cycle for a given sequence of length n is detected, then the algorithm doubles the length to $2n$ and looks for other configurations. The algorithm will stop when no new configuration sequence is found. This means that all the cycles have already been computed. Once all the cycles have been discovered, only configuration sequences having shorter lengths than the fixed current length can be found.

The general schema of the proposed method is presented in Algorithm 1. Note that it is applicable for both the synchronous and asynchronous update modes.

The method starts by generating an extended logic program P_{IG} representing the interaction graph according to the rules described in the previous subsection. We use the ASP system presented in [16] to compute the answer sets representing the sequences of configurations having a fixed length n in the transition graph. When an answer set is found, the

Algorithm 1 The general schema of the cycle search algorithm.

Require: P_{IG}: the logic program representing the interaction graph
I = ASP-Solver(P_{IG})
while I is a new answer set of P_{IG} **do**
  attractor_is_found = false
  $x_I = (x^0, x^1, \ldots, x^t)$ is the sequence of configurations corresponding to I
  $i = t - 1$
  while (($i \geq 0$ and not (attractor_is_found)) **do**
    if $x^t = x^i$ **then**
      attractor_is_found = true
      $attractors = attractors \cup \{(x^{i+1}, \ldots, x^t)\}$
      $P_{IG} = P_{IG} \cup_{j \in \{i+1,t\}} \{\leftarrow v_1(j), v_2(j), \ldots, v_n(j)\}$.
    end if
    $i = i - 1$
  end while
  if not(attractor_is_found) **then**
    $t = 2 \times t$
  end if
  I = ASP-Solver(P_{IG})
end while

algorithm checks if there is a stable cycle or a stable configuration in the the corresponding sequence of configurations. For each detected attractor (stable cycle or stable configuration), the method adds constraint rules to the logic program P_{IG} to avoid this attractor in the remaining search. By adding these rules, it eliminates all the answer sets that could contain an attractor already found. All attractors are identified when no new answer set is generated (i.e., no new configuration sequence is generated).

In the case of the synchronous update mode, all the cycles of the transition graph are stable, which means that each cycle represents a potential attractor of the considered network. However, for the asynchronous update mode, the cycles in the transition graph are not necessarily stable. To detect the instability of a cycle, one can check at each configuration of the cycle, whether the current configuration could evolve to a new configuration that is not part of the cycle. If so, then the cycle is unstable; otherwise it is stable.

Next, we show how to test the stability/instability of a cycle of the transition graph using the answer sets of P_{IG} that we compute.

Proposition 3 *Let P_{IG} be the logic program representing the interaction graph IG having a global transition function f, $TG(f)$ be the corresponding transition graph and $I = \{(v_1(0), \ldots, v_n(0)), \ldots, (v_1(t), \ldots, v_n(t))\}$ be an answer set of P_{IG} corresponding to the sequence of configuration x_I in $TG(f)$. If a subset of literals $I_s = \{(v_1(1), \ldots, v_n(1)), \ldots, (v_1(r), \ldots, v_n(r)), (v_1(r+1), \ldots, v_n(r+1))\} \subseteq I$ corresponding to a sequence of configurations $(x^1, \ldots, x^r, x^1) \subseteq x_I$ forms a stable cycle in $TG(f)$, then every answer set J of P_{IG} different from I ($J \neq I$), is such that $\forall i \in \{1, \ldots, r\}, (v_1(i), \ldots, v_n(i)) \notin J \cap I_s$.*

Proof Assume that I is answer set of P_{IG} corresponding to the configuration sequence x_I in $TG(f)$, and $I_s = \{(v_1(1), \ldots, v_n(1)), \ldots, (v_1(r), \ldots, v_n(r)), (v_1(r+1), \ldots, v_n(r+1))\} \subseteq I$ is a subset of literals of I, corresponding to a sequence of configurations $x_{I_s} = (x^1, \ldots, x^r, x^1) \subseteq x_I$ forming a stable cycle in $TG(f)$. Suppose that P_{IG} has an answer set $J \neq I$ corresponding to a sequence of configurations x_J, such that $\exists k \in \{1, \ldots, r\}, (v_1(k), \ldots, v_n(k)) \in J \cap I_s$. Thus, it results that x_{I_s} and x_J share the con-

figuration x^k corresponding to $(v_1(k), \ldots, v_n(k))$. This means that the configuration $x^k \in I_s$ has more than one successor, and this contradicts the fact that I_s forms a stable cycle. □

We perform the stability verification by a small modification of the ASP solver [16] that we used to compute the answer sets. Indeed, for each answer set of the program P_{IG} containing a cycle, we check for each of its subsets of literals $\{(v_1(i), \ldots, v_n(i))\}$ corresponding to a configuration x^i of the cycle, if a new subset of literals $\{(v_1(i+1), \ldots, v_n(i+1))\}$ corresponding to a configuration x^{i+1} different from the successor of x^i in the cycle could be deduced. In the affirmative case we conclude that the cycle is not stable, otherwise the stability of the cycle is proven.

In practice, we do this by attempting to produce a different configuration at each choice point of the branch of the search tree corresponding to the cycle included in the stable model. This is done by setting a new literal not $Block(v_j, t)$ different from the one representing this choice point. We have integrated this operation in the resolution process of the method [16] that we used to compute the answer sets of the logic program P_{IG} expressing the interaction graph of the considered network.

Note that our incremental approach does not require to know the maximum path size in advance, which is hard to determine because it is related to the diameter of the state transition graph, which is very hard to compute [31]. However, it might be possible to implement this approach by using the incremental ASP control loops and multi-shot solving [30]. In particular, we might not need to determine the maximum path size in advance thanks to Clingo's APIs. We leave the implementation using Clingo as one of our future work

3.2 Detection of stable configurations

In the previous subsection, we have proposed a generic and natural ASP encoding for general Boolean networks under both the synchronous and asynchronous update scheme. From the generic encoding, we have proposed an iterative method for computing all stable configurations and stable cycles of a Boolean network under both the synchronous and asynchronous update scheme. In a sense of ASP, this approach is non-totally declarative. It is inevitable because in general a Boolean network may have stable cycles of size up to 2^n and it is hard to know in advance the maximum length that is related to the diameter of the state transition graph [31].

We have now improved our approach with the focus on only stable configurations, it has become totally declarative. It is worth noting that stable configurations of a Boolean network are the same for both the synchronous and asynchronous update modes [32]. In the improved approach, we simplified the encoding introduced in the previous subsection by just keeping the two constraints $r10$ and $r11$ that remain necessary in the new approach. With this new approach, the calculation of stable configurations is completely declarative and we obtained much better results (see Section 5.2). The details are as follows.

First, we remove the time t from the ASP encoding. Now, each node v_i is represented by two atoms (not predicates) v_i and $\neg v_i$. Second, since in a stable configuration, $v_i(t+1)$ is always equal to $v_i(t)$ for every node v_i, we replace $v_i(t+1)$ and $v_i(t)$ by v_i. Formally, the two constraints $r10$ and $r11$ become

$$r'_{10} : \{v_i \leftarrow m_i^j \mid 1 \leq j \leq l\}, i \in \{1, \ldots, n\},$$
$$r'_{11} : \{\neg v_i \leftarrow m'^j_i \mid 1 \leq j \leq e\}, i \in \{1, \ldots, n\},$$

respectively. Third, we add the two set of ASP rules to ensure that an answer set corresponds to a configuration:

$$r_{17}: \bigcup_{i\in\{1,\ldots,n\}} v_i, \neg v_i \leftarrow$$
$$r_{18}: \bigcup_{i\in\{1,\ldots,n\}} \leftarrow v_i, \neg v_i$$

We show an illustration and the correctness of the above encoding in Example 7 and Proposition 4, respectively.

Example 7 Consider the Boolean network given in Example 1. The normal logic program following the above encoding is as follows:

$$\begin{aligned} 1 &\leftarrow 2\\ 2 &\leftarrow 1, 2'\\ 1' &\leftarrow 2'\\ 2' &\leftarrow 1'\\ 2' &\leftarrow 2\\ 1, 1' &\leftarrow\\ &\leftarrow 1, 1'\\ 2, 2' &\leftarrow\\ &\leftarrow 2, 2' \end{aligned}$$

Proposition 4 *Let π be the logic program of a Boolean network following the above encoding. Then an answer set of π is equivalent to a stable configuration of the Boolean network.*

Proof Let I be a set of atoms of π. Let x be a state of the Boolean network such that $x_i = 1$ if and only if $v_i \in I$ and $x_i = 0$ if and only if $\neg v_i \in I$ for every node v_i. Based on the proof of Proposition 2, we can imply that I is an answer set of π if and only if (x, x) is a transition of the transition graph of the Boolean network. Herein, x_i plays the roles of both $v_i(t + 1)$ and $v_i(t)$. (x, x) is a transition of the transition graph of the Boolean network is equivalent to that x is a stable configuration of the Boolean network. We can conclude the proof. □

Finally, we showcase a crucial application of our new approach. Recently, we have developed a new method called mtsNFVS [22] for computing all attractors of an asynchronous Boolean network (i.e., stable configurations, stable cycles, and loose attractors). mtsNFVS first computes a Negative Feedback Vertex Set (NFVS) of the interaction graph of the asynchronous Boolean network. Based on the chosen NFVS U^-, mtsNFVS randomly chooses a set B^- of Boolean values corresponding to the nodes in U^-. From U^- and B^-, mtsNFVS builds a new Boolean network called the reduced-dynamics Boolean network whose set of stable configurations exactly covers all attractors of the asynchronous Boolean network (see Theorem 3). The word "cover" means that for every attractor at least one of its configurations belongs to the set of stable configurations of the reduced-dynamics Boolean network. Then mtsNFVS uses the reachability analysis on the asynchronous Boolean network to filter out this set. Finally, mtsNFVS returns the set A of configurations (not attractors) that one-to-one covers the set of attractors of the asynchronous Boolean network.

Theorem 3 *([22]) Let $\mathcal{A}$ be an asynchronous Boolean network and U^- be an NFVS of it. Let B^- be a set of Boolean values corresponding to the nodes in U^-. We first construct the reduced-dynamics Boolean network denoted by $\mathcal{A}^{red}$ as follows. $\mathcal{A}^{red}$ includes the set of nodes of $\mathcal{A}$ and its set of Boolean functions is given by:*

$$\begin{cases} f_i^{red} = f_i \quad \text{if } x_i \notin U^-, \\ f_i^{red} = [(x_i \leftrightarrow b_i) \wedge b_i] \vee [\neg(x_i \leftrightarrow b_i) \wedge f_i] \quad \text{if } x_i \in U^-, \end{cases}$$

where $\leftrightarrow$ denotes the bi-implication logical operator. Then for any NFVS U^- and any set of Boolean values B^-, the set of fixed points of the reduced-dynamics Boolean network covers exactly all attractors of $\mathcal{A}$.

Crucially, the number of stable configurations of the reduced-dynamics Boolean network depends on and may be exponential in the size of U^- [22]. In several real-world models, U^- is usually large (e.g., > 25), leading to too many stable configurations. Hence, in mtsNFVS, computing stable configurations of the reduced-dynamics Boolean network is a demanding task. Indeed, we observed in [22] that the most of the running time was spent for computing stable configurations. Our above ASP encoding can benefit to this task of mtsNFVS, thus it can speedup the whole process of mtsNFVS. Furthermore, the computation of stable configurations is also the bottleneck in many analysis and control methods for Boolean networks, which again can be overcome by applying our proposed method.

Note that it is difficult to compare the whole method mtsNFVS with our approach in the present article. The reasons include (1) our approach returns the set of stable configurations and cycles of a Boolean network whereas mtsNFVS returns a set of configurations, (2) our approach deals with both the synchronous and asynchronous update modes whereas mtsNFVS is specifically designed for the asynchronous update mode.

4 Detection of attractors in circular Boolean networks

In this section, we deal with the specific case of Boolean networks called *circular* Boolean networks. We shall focus on the computation of the attractors for the asynchronous update mode. A first release of this work has been presented in [21].

4.1 Representation of interaction graphs

The dynamic of a Boolean network is represented by its transition graph TG, and the interaction graph IG expresses interactions between genes. An important subject of study is to make formal links between these two representations [25]. In the following, we show how an interaction graph IG is expressed as an extended logic program P_{IG} given in its Horn clausal form $HC(P_{IG})$. We shall prove some important theoretical properties on the representation $HC(P_{IG})$ and its extensions that we will use to establish the relation between $HC(P_{IG})$ representing IG and TG.

Our approach is to calculate the stable configurations and the stable cycles of the transition graph TG by calculating the stable and extra-stable models of the logic program P_{IG}. The formalism of Boolean networks associates an entity $i \in \{1, \ldots, n\}$ to a Boolean variable v_i. To lighten the notation, we will use in the sequel i instead of v_i when possible.

To address this situation, we opted for the answer set programming (ASP) framework where we use the semantics introduced in [18]. ASP gives a good compromise between the expressiveness of the knowledge representation language and the efficiency of the associated resolution tools.

Definition 5 Given the interaction graph IG of a Boolean network representing a gene regulatory network, the logic program P_{IG} expressing IG and a gene i, we define the following.

- i means that the gene i is active in the cell.
- $\neg i$ means that the gene i is not active in the cell.
- not $\neg i$ (resp. $\neg$ not $\neg i$) means that the cell gives (resp. does not give) the permission to activate i. In other words, the cell has (resp. has not) the ability to activate i.
- not i (resp. $\neg$ not i) means that the cell gives (resp. does not give) the permission to disable i. In other words, the cell has (resp. has not) the ability to inhibit i.

Definition 6 The translation of IG into a logic program P_{IG} is done by transcribing every arc in IG into the following pair of rules.

- A positive arc $(i, +, j)$ between the two genes i and j is expressed by the two rules $j \leftarrow$ not $\neg i$. and $\neg j \leftarrow$ not i.
- A negative arc $(i, -, j)$ between the two genes i and j is expressed by the two rules $j \leftarrow$ not i. and $\neg j \leftarrow$ not $\neg i$.

Example 8 The negative circuit of Fig. 2 (a) is translated into the extended logic program $P_{IG}(f)$ with the following rules

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
\neg 2 &\leftarrow \text{not } \neg 1\\
3 &\leftarrow \text{not } \neg 2\\
\neg 3 &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } \neg 3\\
\neg 1 &\leftarrow \text{not } 3
\end{aligned}
$$

The interaction graph of Fig. 2 (b) representing the positive circuit is expressed by the extended logic program $P_{IG}(g)$ with the following rules

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
\neg 2 &\leftarrow \text{not } \neg 1\\
3 &\leftarrow \text{not } \neg 2\\
\neg 3 &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } 3\\
\neg 1 &\leftarrow \text{not } \neg 3
\end{aligned}
$$

In our approach, each extended logic program P_{IG} is transformed to an equivalent normal logic program P'_{IG} that is expressed in the end by a set of Horn clauses $HC(P'_{IG})$ in the used semantics [18]. An extension of the pair $(HC(P'_{IG}), STB)$ is the consistent set formed

by all the clauses derived from $HC(P'_{IG})$ when assigning a maximal set of positive literals not $A_i \in STB$ to $HC(P'_{IG})$ the value true. In this context, the STB set acts as a set of permissions to activate genes (resp. to inhibit a gene). In the sequel, we will consider the Horn clausal representation $HC(P'_{IG})$ instead of the logic program P'_{IG} that we denote by $HC(P_{IG})$ when there is no confusion. We will also simply say extensions of $HC(P_{IG})$ to mean extensions of $(HC(P'_{IG}), STB)$.

Remark 3 In this context, the role of an extension appears to gather a maximum of consistent permissions. Note that even if not $\neg i$ stands for the cell permitting to attempt the production of j, this production is not mandatory. It can be produced or not, according to the context (i.e., the set of all interactions in the cell). We could establish a similar reasoning for the case of the literal not i that gives the authorization to disable j.

In general, it is permitted to have both not $\neg i$ and not i in the used semantics. But from a biological perspective, we cannot give permission to both activate a gene j and inhibit it at the same moment. Proposition 5 below expresses this biological aspect.

Proposition 5 *If $HC(P_{IG})$ is a logic program representing the interaction graph IG, then for every $i \in V = \{1, \ldots, n\}$, the condition $\neg(\text{not } \neg i \wedge \text{not } i)$ holds in $HC(P_{IG})$.*

Proof By definition, if IG contains a signed arc $(i, \{+, -\}, j)$, then the translation of this arc induces two sets of clauses $\{j \vee \neg \text{ not } \neg i, \neg j \vee \neg \text{ not } i\}$ or $\{j \vee \neg \text{ not } i, \neg j \vee \neg \text{ not } \neg i\}$. In both cases, if not $\neg i \wedge$ not i holds, then, we infer $j \wedge \neg j$ that expresses an inconsistency. Thus, $\neg(\text{not } \neg i \wedge \text{not } i)$ holds in $HC(P_{IG})$. □

Proposition 6 *Let IG be an interaction graph whose logic encoding is $HC(P_{IG})$, we have the following.*

1. *$HC(P_{IG})$ is consistent.*
2. *$HC(P_{IG})$ has at least one extension.*

Proof 1. The encoding $HC(P_{IG})$ is formed by a set of binary Horn clauses. That is, each clause contains at least one negative literal. The assignment of all literals to false is then a model of $HC(P_{IG})$. Thus, $HC(P_{IG})$ is consistent.
2. Since $HC(P_{IG})$ is consistent, it results from Proposition 1 that $HC(P_{IG})$ has at least one extension.

□

The following definitions and propositions are important to understand the intuition behind the representation described in this section.

Definition 7 Let IG be an interaction graph whose set of vertices is $V = \{1, \ldots, n\}$, $HC(P_{IG})$ its logic encoding and E an extension of $HC(P_{IG})$ obtained by adding to $HC(P_{IG})$ a maximal consistent set of literals $\{\text{not } k\}$, with $k \in \{1, \ldots, n, \neg 1, \ldots, \neg n\}$. Then we have the following.

1. E is complete if for all $i \in V$, not $\neg i \in E$ or not $i \in E$.
2. i is free in E if $i \notin E$ and $\neg i \notin E$. Otherwise, it is fixed.
3. The degree of freedom of E (denoted by $deg(E)$) is the number of its free elements $i \in V$.
4. The mirror of $E = HC(P_{IG}) \cup \{\text{not } k\}$ (denoted by $mir(E)$) is defined as $mir(E) = HC(P_{IG}) \cup \{\text{not } \neg k\}$.

Proposition 7 *If $HC(P_{IG})$ is the logic encoding of the program P_{IG} representing the interaction graph IG and E is an extension of $HC(P_{IG})$, then the mirror of E is also an extension of $HC(P_{IG})$.*

Proof By definition, if IG contains a signed arc $(i, \{+, -\}, j)$, then its encoding in $HC(P_{IG})$ includes both sets of clauses $\{j \vee \neg \text{not} \neg i, \neg j \vee \neg \text{not} i\}$ or $\{j \vee \neg \text{not} i, negj \vee \neg \text{not} \neg i\}$. An extension is the set of all consistent clauses derived from $HC(P_{IG})$ when adding a maximal set of positive literals not i to $HC(P_{IG})$. If we inverse each literal not i in the extension, i.e., we replace not i (resp. not $\neg i$) by not $\neg i$ (resp. not i), then we get two cases. The first case corresponds to the presence of a positive arc in the interaction graph IG. In this case, we infer j when not $\neg i$ holds, or $\neg j$ if not i holds. The second case corresponds to the presence of a negative arc in the interaction graph IG. In this case, we infer $\neg j$ when not $\neg i$ holds and infer j when not i holds. Thus, it is trivial to see that the extension E and its mirror $mir(E)$ are symmetrical. It results that $mir(E)$ is an extension too. □

In the following, we prove that in some particular interaction graphs IG including circuits, complete extensions of $HC(P_{IG})$ are of degree 0 and induce answer sets of $HC(P_{IG})$.

Proposition 8 *Let IG be an interaction graph, $HC(P_{IG})$ be the logic encoding of the program P_{IG} representing IG and E be an extension of $HC(P_{IG})$. If every node of IG has at least one incoming arc, then any complete extension of $HC(P_{IG})$ is of degree 0.*

Proof Let E be a complete extension of $HC(P_{IG})$. To prove that E is of degree 0, we have to prove that each variable j of $HC(P_{IG})$ is not free in E. In other words, for each node j in the interaction graph IG, we have either $\neg j \in E$ or $j \in E$. By the hypothesis, j has a positive/negative incoming arc $(j, +/-, i)$ in IG.

If the arc is positive, then it is expressed by the pair of clauses $\{j \vee \neg \text{not} \neg i, \neg j \vee \neg \text{not} i\}$. Since E is complete, we have either not $i \in E$ or not $\neg i \in E$. If not $\neg i \in E$, then j is inferred $(j \in E)$. If not $i \in E$, then $\neg j$ is inferred $(\neg j \in E)$. Then in both cases j is not free in E.

The case of a negative arc is treated in the same way. We will have the rules $\{j \vee \neg \text{not} i, \neg j \vee \neg \text{not} \neg i\}$. If not $\neg i \in E$, then we infer $\neg j$ and if not $i \in E$, then we derive j. Therefore, for all the assumptions we infer either j or $\neg j$. Thus, there is no free element j in E and $deg(E) = 0$. □

Proposition 9 *Let IG be an interaction graph and $HC(P_{IG})$ be the logic encoding of the program P_{IG} representing IG. If any node of IG has at least one incoming arc, then any complete extension of $HC(P_{IG})$ corresponds to an answer set of $HC(P_{IG})$.*

Proof Let E be a complete extension of $HC(P_{IG})$. E corresponds to an answer set if for any node i, the discriminant condition holds for both i and $\neg i$. That is both conditions (1) $\neg \text{not} i \in E \Rightarrow i \in E$ and (2) $\neg \text{not} \neg i \in E \Rightarrow \neg i \in E$ hold. Since E is complete, then it is of degree 0 (Proposition 8). It results that either i or $\neg i$ is in E. We have two cases as follows.

If $i \in E$, then (1) is trivially verified. According to the mutual exclusion $ME = \{(\neg i \vee \neg \text{not} i)\}$, we obtain $\neg \text{not} i \in E$. In this case, we have $\neg i \notin E$. Suppose now that $\neg \text{not} \neg i \in E$, this means that not $\neg i \notin E$. As E is complete, we have not $i \in E$, and this contradicts the fact that $\neg \text{not} i \in E$. Thus, the condition (2) is verified.

If we have $\neg i \in E$, then the condition (2) is trivially verified. According to the mutual exclusion ME, we obtain $\neg \text{not} \neg i \in E$. In this case, we have $i \notin E$ and if we suppose that $\neg \text{not} i \in E$, then not $i \notin E$. As E is complete, then not $\neg i \in E$, and this contradicts the fact that $\neg \text{not} \neg i \in E$. Therefore the condition (1) is verified.

Since E verifies the discriminant condition in both cases, then E induces an answer set of $HC(P_{IG})$ (Theorem 2). □

We now show that any answer set of $HC(P_{IG})$ corresponds to an extension of degree 0.

Proposition 10 *Let IG be an interaction graph. If any node of IG has at least one incoming arc, then any answer set of $HC(P_{IG})$ corresponds to an extension E of degree 0.*

Proof Let E be an extension inducing an answer set of $HC(P_{IG})$. By definition, E is maximally consistent with respect to the literals of the form not $i \in E$ or not $\neg i \in E$ and verifies the discriminant conditions (a) $\neg \text{not}\, i \in E \Rightarrow i \in E$ and (b) $\neg \text{not}\, \neg i \in E \Rightarrow \neg i \in E$ corresponding to both i and $\neg i$. The extension E induces then an answer set of $HC(P_{IG})$. We have to prove that for all $i \in HC(P_{IG})$, we have either $i \in E$ or $\neg i \in E$. There are three study cases:

1. The case where not $i \in E$ and not $\neg i \notin E$. It results from Proposition 5 that $\neg \text{not}\, \neg i \in E$. Then, from the discriminant condition (b), we get $\neg i \in E$.
2. The case where not $\neg i \in E$ and not $i \notin E$. From Proposition 5, we get $\neg \text{not}\, i \in E$. Thus, $i \in E$ because the condition (a) holds.
3. The case where not $i \notin E$ and not $\neg i \notin E$. In this case, we have not $i \wedge E \models \square$ and not $\neg i \wedge E \models \square$. Thus, $E \models \neg \text{not}\, i$ and $E \models \neg \text{not}\, \neg i$. From (a) and (b), we have $E \models i$ and $E \models \neg i$. Thus, we get an inconsistency that contradicts the fact that E is an extension.

It results that only the first and the second case could be possible. Thus, we have either $i \in E$ or $\neg i \in E$, and $deg(E) = 0$. □

In what follows, we shall show that for an interaction graph IG representing a positive circuit of nodes, the corresponding logic encoding $HC(P_{IG})$ has two answer sets of n elements.

Proposition 11 *If the interaction graph IG is a positive circuit of n entities, then its logical form $HC(P_{IG})$ has two extensions that induce two answer sets of size n.*

Proof The proof is based on the results of Proposition 9 and the fact that in a positive circuit each gene acts positively on itself through the circuit. Indeed, if we give at the beginning the authorization to activate the gene i (by supposing not $\neg i$), then we will end up deducing that i is active. Conversely, if we initially give the authorization to deactivate i (by supposing not i) then we will deduce that i is inactive (we get $\neg i$). We can then construct two complete extensions of degree 0. The first one is made by supposing at the beginning the literal not $\neg i$ and the second one is its mirror extension that is obtained by supposing at the beginning the literal not i. Both extensions are complete and are of degree 0. As the two extensions are complete and of degree 0, we deduce from Proposition 9 that each of them induces a stable model of $HC(P_{IG})$ of size n. □

Example 9 Consider the extended logic program of Example 8 expressing the interaction graph of Example 2 representing the positive circuit of size 3 (Fig. 2 (b)), we have $P_{IG}(g)$

as follows.

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
\neg 2 &\leftarrow \text{not } \neg 1\\
3 &\leftarrow \text{not } \neg 2\\
\neg 3 &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } 3\\
\neg 1 &\leftarrow \text{not } \neg 3
\end{aligned}
$$

$P_{IG}(g)$ is translated to the equivalent normal program $P'_{IG}(g)$:

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
2' &\leftarrow \text{not } 1'\\
3 &\leftarrow \text{not } 2'\\
3' &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } 3\\
1' &\leftarrow \text{not } 3'\\
\bot &\leftarrow 1, 1'\\
\bot &\leftarrow 2, 2'\\
\bot &\leftarrow 3, 3'
\end{aligned}
$$

$HC(P'_{IG}(g))$ has two extensions $E_1 = HC(P'_{IG}(g)) \cup \{\text{not } 1, \text{not } 2', \text{not } 3'\}$ and $E_2 = HC(P'_{IG}(g)) \cup \{\text{not } 1', \text{not } 2, \text{not } 3\}$ that correspond to two stable models. E_1 and E_2 are two extensions that verify both discriminant conditions: $E \models \neg \text{not } i \Rightarrow E \models i$ and $E \models \neg \text{not } i' \Rightarrow E \models i'$ for all i and i'.

We notice that E_1 is complete because for every $i \in HC(P'_{IG}(g))$, either $\text{not } i'$ ($\text{not } \neg i$) belongs to E_1 or $\text{not } i$ belongs to E_1. In addition, we have either $i \in E_1$ or $i' = \neg i \in E_1$ for every $i \in HC(P'_{IG}(g))$, which means that the degree of freedom of E_1 is 0.

The extension E_2 is the mirror of E_1. The same reasoning could be applied to show that E_2 is complete and of degree 0. The stable models induced by E_1 and E_2 are $M'_1 = \{1', 2, 3\}$ and $M'_2 = \{1, 2', 3'\}$, respectively. The corresponding answers sets of the extended program $P_{IG}(g)$ are $M_1 = \{\neg 1, 2, 3\}$ and $M_2 = \{1, \neg 2, \neg 3\}$, respectively. We can see that the two precedent answers sets correspond to the two stable configurations of the transition graph (Fig. 3(b)) of the positive circuit presented in Example 2.

The intuition behind the computation of E_1 is given by the construction scheme described in Fig. 4(b). The interaction graph is represented in Fig. 4(a), whereas Fig. 4(b) gives the different construction steps of E_1. Initially, E_1 is empty. We begin the process by assuming

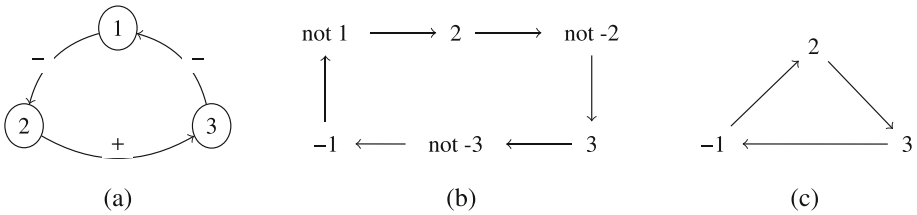


Fig. 4 (a) $IG(f)$, (b) Construction of E_1, (c) The graph of M_1

that not 1 is in E_1. Thus, by applying the rule $2 \leftarrow \text{not } 1$, we deduce 2 and then $\neg \text{not } 2$ is deduced from the mutual exclusion clause $(\neg 2 \vee \neg \text{not } 2)$. The construction of E_1 continues by adding not $\neg 2$ to E_1, and then we deduce 3. The mutual exclusion $(\neg 3 \vee \neg \text{not } 3)$ prohibits the application of not 3. Then, we add not $\neg 3$ to E_1 from which we infer $\neg 1$. If we are only interested in the gene literals i, then we obtain the restricted graph of E_1 shown in Figure 4(c) that represents the corresponding stable model M_1. This model corresponds to one of the two stable configurations of the transition graph of Fig. 3(b) of Example 2. The extension E_2 is built in the same way as E_1. To get E_2, we must start the process by assuming that $\text{not } 1' = \text{not } \neg 1$ is true in E_2.

From the biological point of view, the answer sets' variables represent the state of each gene of the regulatory network. For example, $M_1 = \{\neg 1, 2, 3\}$ says that 2 and 3 are active and 1 is inactive. Similarly, $M_2 = \{1, \neg 2, \neg 3\}$ means that both 2 and 3 are inactive and 1 is active.

In the following, we show that each interaction graph IG representing a negative circuit of n nodes has $2n$ extra-extensions of degree 1 inducing $2n$ extra-stable models that encode a stable cycle of size $2n$ in the transition graph.

Proposition 12 *If the interaction graph IG is a negative circuit of size n then $HC(P_{IG})$ has $2n$ extra-extensions of degree 1 inducing $2n$ extra-stable models of size $n-1$.*

Proof The proof is based on the fact that in a negative circuit, a gene acts negatively on itself through the circuit. Indeed, if we give at the beginning the authorization to activate the gene i by supposing not $\neg i$, then when we close the cycle we deduce that i is inactive ($\neg i$ is true). Conversely, if initially we authorize to inhibit i by supposing not i, then we deduce that i is active (i is true) when we close the cycle. We then obtain an inconsistency in both cases because we have both i and $\neg i$ simultaneously. This deduction means that we cannot have a complete extension in both cases.

Then, we obtain an incomplete extension E and its mirror extension $mir(E)$ which is also incomplete. In both them, there is neither the literal not j nor the literal not $\neg j$ with j being the predecessor of i. Thus, neither i nor $\neg i$ is true in both extensions. On the other hand, all the other elements different from i are linked in these two extensions. It follows that the two extensions are therefore of degree 1. It is also trivial to see that both extensions do not satisfy the discriminating condition. Indeed, we have $\neg \text{not } i$ in E without having i in E, and we have $\neg \text{not } \neg i$ in $mir(E)$ without having $\neg i$ in $mir(E)$.

Therefore, we have two mirror extra-extensions of degree 1 inducing two extra-stable models of size $n-1$. Each time we change the starting element i, we get two other mirror extra-extensions of degrees 1, which induce two other extra-stable models of sizes $n-1$. In total, there will be be $2n$ extra-extensions of degree 1 inducing $2n$ extra-stable models of sizes $n-1$. □

Example 10 Consider the extended logic program of Example 8 expressing the interaction graph of Example 2 corresponding to a negative circuit of size 3 (Fig. 2 (a)). We obtain

$P_{IG}(f)$ as follows.

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
\neg 2 &\leftarrow \text{not } \neg 1\\
3 &\leftarrow \text{not } \neg 2\\
\neg 3 &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } \neg 3\\
\neg 1 &\leftarrow \text{not } 3
\end{aligned}
$$

After translation, we get the normal logic program $P'_{IG}(f)$ as follows.

$$
\begin{aligned}
2 &\leftarrow \text{not } 1\\
2' &\leftarrow \text{not } 1'\\
3 &\leftarrow \text{not } 2'\\
3' &\leftarrow \text{not } 2\\
1 &\leftarrow \text{not } 3'\\
1' &\leftarrow \text{not } 3\\
\bot &\leftarrow 1, 1'\\
\bot &\leftarrow 2, 2'\\
\bot &\leftarrow 3, 3'
\end{aligned}
$$

The logic encoding $HC(P'_{IG}(f))$ has six extra-extensions (E'_i) that induce six extra-stable models (M'_i) as follows.

1. $E'_1 = HC(P'_{IG}(f)) \cup \{\text{not } 1, \text{not } 2'\}$ and $E'_1 \models \{\text{not } 1, 2, \text{not } 2', 3\}$, it induces $M'_1 = \{2, 3\}$.
2. $E'_2 = HC(P'_{IG}(f)) \cup \{\text{not } 1, \text{not } 3\}$ and $E'_2 \models \{\text{not } 1, 2, \text{not } 3, 1'\}$, it induces $M'_2 = \{2, 1'\}$.
3. $E'_3 = HC(P'_{IG}(f)) \cup \{\text{not } 1', \text{not } 3'\}$ and $E'_3 \models \{\text{not } 1', 2', \text{not } 3', 1\}$, it induces $M'_3 = \{2', 1\}$.
4. $E'_4 = HC(P'_{IG}(f)) \cup \{\text{not } 1', \text{not } 2\}$ and $E'_4 \models \{\text{not } 1', 3', \text{not } 2, 3'\}$, it induces $M'_4 = \{2', 3'\}$.
5. $E'_5 = HC(P'_{IG}(f)) \cup \{\text{not } 2', \text{not } 3'\}$ and $E'_5 \models \{\text{not } 2', 3, \text{not } 3', 1\}$, it induces $M'_5 = \{3, 1\}$.
6. $E'_6 = HC(P'_{IG}(f)) \cup \{\text{not } 2, \text{not } 3\}$ and $E'_6 \models \{\text{not } 2, 3', \text{not } 3, 1'\}$, it induces $M'_6 = \{3', 1'\}$.

Figure 5(a) shows the considered negative circuit expressed as a logic program. Figure 5(b) illustrates the construction of the extension E'_1. It is built by adding to $HC(P'_{IG}(f))$ both literals not 1 and not $2'$. We can see in Fig. 5(b) that it is impossible to deduce $1'$. Indeed, to get $1'$, we must use the rule ($1' \leftarrow \text{not } 3$). But this is impossible because $\neg$ not 3 results from the mutual exclusion ($\neg 3 \vee \neg \text{not } 3$). On the other hand, we cannot get 1. As not 1 holds, then from the mutual exclusion ($\neg 1 \vee \neg \text{not } 1$), we get $\neg 1$. Thus, we cannot have 1. We can notice that the extension E'_1 is not complete because it contains neither not 3 nor not $3'$. The element 1 is free in E'_1 because $1 \notin E'_1$ and $\neg 1 \notin E'_1$. As a result, E'_1 is an extension of degree 1. Figure 5(c) gives the restriction of E'_1 to the corresponding extra-stable model M'_1.

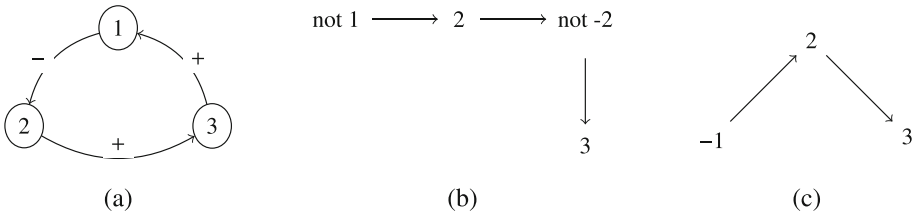


Fig. 5 (a) $IG(g)$ the negative circuit, (b) Construction of E'_1, (c) Construction of M'_1

4.2 The relation between the transition graph and the logical representation of its interaction graph

Hereafter, we shall explore the relationship between the logical representation $HC(P_{IG})$ of the interaction graph IG and the corresponding transition graph TG. In order to do this, we see that the vertices of the transition graph TG corresponding to stable configurations or stable cycles could be expressed, in fact, by the extensions/extra-extensions (stable models or extra-stable models) of the logical encoding $HC(P_{IG})$.

Given a Boolean network, having an interaction graph IG, a transition graph TG, and $HC(P_{IG})$ is the clausal horn representation of the associated logic program P_{IG}, we shall show for positive circuits (Theorem 4) that there is an isomorphism between the stable configurations of TG and the answer sets of $HC(P_{IG})$. Furthermore, we shall also prove (Theorem 5) that any stable cycle of the transition graph of a negative circuit interaction graph of size n is encoded as a set of $2n$ extra-stable models of degree 1 of $HC(P_{IG})$.

Proposition 13 *Given a Boolean network represented by the interaction graph IG where TG is its associated transition graph and $HC(P_{IG})$ is the Horn clausal representation of the logic program P_{IG} expressing IG. If s is a vertex (a configuration) of TG representing an extension/extra-extension E of $HC(P_{IG})$ of degree k, then s has exactly k successors.*

Proof If i is free in the extension/extra-extension E representing the configuration s, then either $\neg i$ or i is true in an extension/extra-extension corresponding to a state s' accessible from s. By construction of TG, s' is the single successor of s that verifies this statement. This property is verified for each free element i in E. Thus, if the degree of freedom of E is k, then there is k accessible vertices from s. □

Theorem 4 *Given a Boolean network where IG is the interaction graph and $HC(P_{IG})$ is the Horn clausal representation of the logic program P_{IG} associated with IG. Then the following assumptions hold:*

1. *If $X = (x_1, x_2, \ldots, x_n)$ is an answer set of P_{IG} induced by an extension of $HC(P_{IG})$, then $X = (x_1, x_2, \ldots, x_n)$ is a stable configuration of the transition graph TG.*
2. *If $X = (x_1, x_2, \ldots, x_n)$ is a stable configuration of the transition graph TG, then $X = (x_1, x_2, \ldots, x_n)$ corresponds to an answer set of P_{IG} induced by an extension of $HC(P_{IG})$.*

Proof 1. Let E be the extension inducing the answer set $X = (x_1, x_2, \ldots, x_n)$ and $s = (x_1, x_2, \ldots, x_n)$ be the vertex representing E in TG. As E is an extension, then its degree of freedom is 0. According to Proposition 13, it follows that the only accessible node from s is itself. Thus, s is a stable configuration of TG.

2. If $s = (x_1, x_2, \ldots, x_n)$ is a stable configuration of the associated transition graph TG, then no arcs come out of s. The only vertex accessible from s is itself. It follows that for each element x_i (resp. $\neg x_i$) of s, either x_i is true or $\neg x_i$ is true. Then, all the x_i are linked in the extension E corresponding to the configuration s. That is, the freedom degree of E is 0. It results from Proposition 10 that $s = (x_1, x_2, \ldots, x_n)$ forms an answer set of P_{IG}.

□

Example 11 Figure 6(b) shows the two extensions obtained for the logic program of the positive circuit of Example 8. Both extensions induce two answer sets that encode the two stable configurations of the transition graph (Fig. 6(a)) that are drawn in bold font.

Theorem 5 *Given a Boolean network where the interaction graph IG is a negative circuit of size n and P_{IG} the logic program expressing IG. Then, the set of $2n$ extra-extensions of $HC(P_{IG})$ correspond to a stable cycle in the associated transition graph TG of size $2n$.*

Proof Proposition 12 guarantees the existence of $2n$ extra-extension (extra-stable models) of degree 1. We have to consider here the fact that all the $2n$ extra-extensions are of degree 1. This implies that there is a single transition from each extra-extension of degree 1 to another extra-extension of degree 1, producing a stable cycle of $2n$ extra-extensions. This corresponds to a stable cycle of size $2n$ in TG, where each extra-extension identifies a configuration in the cycle of TG. □

Example 12 Figure 7(c) shows the extra-extensions obtained for the logic program corresponding to the negative circuit of Example 8. We can see that six extra-extensions of degree 1 inducing six extra-stable models are found and each of them identifies a configuration of the stable cycle of the corresponding transition graph given in bold font (Fig. 7(a)). Figure 7(b) shows the stable cycle separately from the rest of the transition graph.

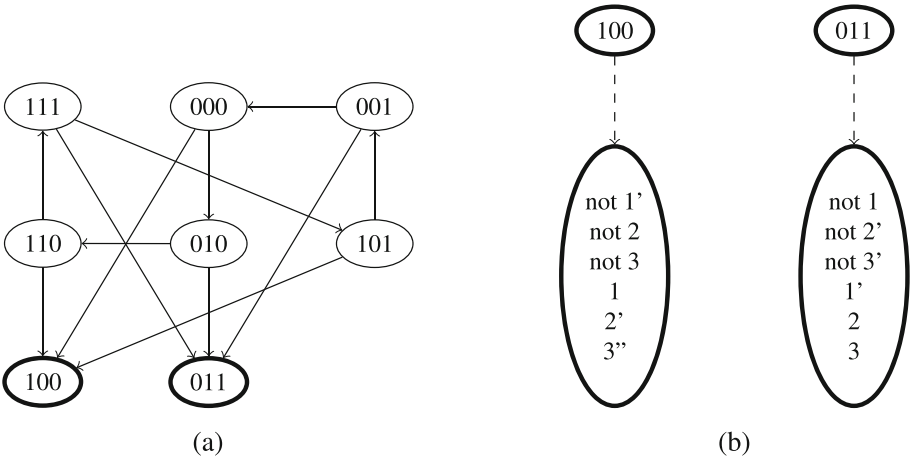


Fig. 6 The stable configurations of TG expressed as stable models of $HC(P_{IG})$. For simplification, self transitions are omitted

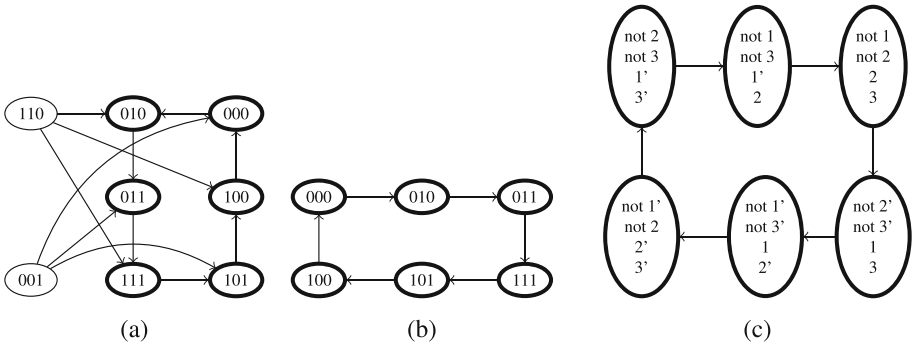


Fig. 7 A stable cycle of TG seen as a set of $2n$ extra-extensions of $HC(P_{IG})$. For simplification, self transitions are omitted

5 Empirical validation

In this section, we evaluate our proposed methods for general Boolean networks (see Section 3) and circular Boolean networks (see Section 4). In Section 5.1, we test our method (presented in Section 3.1 for computing all stable configurations and cycles of a general Boolean network) on several popular networks in the literature. In Section 5.2, we test our method (presented in Section 3.1 for computing only stable configurations of a general Boolean network) on the seven reduced-dynamics networks of the seven real-world models used in [22]. In Section 5.3, we test our method (presented in Section 4 for computing only stable configurations and cycles of a circular Boolean network) on many randomly generated networks. Our code and benchmark problems are publicly available in the GitHub repository [3].

5.1 Stable configurations and cycles of general networks

To illustrate the soundness of our approach introduced in Section 3 for the simulation of Boolean networks and the detection of attractors, we applied it to real biological networks. We evaluated the method for the synchronous and asynchronous update modes on real genetic networks found in the literature. We experimented the method on networks corresponding to *yeast cell cycle* [33] and *fission yeast cell cycle* studied in [34]. We also applied the method to *T-helper cell differentiation* and its Boolean network described in [6].

We are interested here in the computation time and the number of attractors found. Table 1 shows the obtained results. We can see that the method is efficient on all the networks. We also note that the attractors in the synchronous case often coincide with those in the asynchronous case. The similarity is due to a large number of stable configurations compared to the number of stable cycles in these networks. It is well known that the stable configurations are generally the same in both the synchronous and asynchronous updating modes [6].

[3] https://github.com/tarekhaledasp/ASP-BN.git

Table 1 The results obtained on common graph regulatory networks found in the literature

Network	Genes	Attractors	Update mode	Time (sec)
Yeast cell cycle	11	6	Synchronous	2.21
	11	6	Asynchronous	0.56
Fission Yeast	10	11	Synchronous	1.82
	10	12	Asynchronous	0.50
T-helper cell differentiation	23	2	Synchronous	0.37
	23	2	Asynchronous	0.43

5.2 Stable configurations of reduced-dynamics networks

To evaluate our new approach for computing only stable configurations of a Boolean network (see Section 3.2), we applied it to the reduced-dynamics networks of the seven selected real-world networks used in [22]. The number of stable configurations of a reduced-dynamics Boolean network depends on the size of the minimum negative feedback vertex set of the original network and this number is large in most cases [22]. Hence, computing stable configurations of reduced-dynamics networks is computationally demanding task.

In [22], we used the state-of-the-art method PyBoolNet [35] for computing stable configurations of reduced-dynamics networks. This method also relies on ASP but it uses another encoding based on prime-implicants of Boolean functions. It uses Clingo [36] as the underlying ASP solver. In contrast, our ASP encoding uses the disjunctive normal forms of Boolean functions only. We used both Clingo and our own solver system (i.e., HC-asp [16]) as the underlying ASP solvers.

Table 2 shows the running time comparison between PyBoolNet and our new approach on the reduced-dynamics networks. Columns n and e denote the number of nodes and the number of interactions of the reduced-dynamics Boolean network, respectively. Column $|F|$ denotes the number of stable configurations. Columns 6-7 denote the running time (in seconds) of our approach using the Clingo solver and the HC-asp solver, respectively. Columns 8-9 denote the speedups compared to PyBoolNet of our approach using the Clingo solver and the HC-asp solver, respectively.

From the results shown in Table 2, we can first see that the HC-asp solver is quite faster than the Clingo solver in most networks (except the Colon-Cancer network with very small running time). This is consistent to the conclusion shown in [16] that HC-asp (even with its initial prototype) is better than Clingo. In addition, when the number of solutions (i.e., stable configurations) increases, the difference between HC-asp and Clingo also increases. This trend is also reported in [16]. Hereafter, we shall only compare PyBoolNet with our approach using Clingo.

First, our approach is much more efficient than PyBoolNet. The speedup is significant even between one and two orders of magnitude. Second, our approach seems to scale much better than PyBoolNet with respect to the problem complexity (i.e., the number of nodes n and the number of solutions F). Finally, our method depends on not only n and F but also the Boolean functions that can be partially exhibited by the average number of interactions (i.e., e/n) [22]. For example, the number of stable configurations of the PROSTATE-CANCER network is smaller than that of the IL6-Signalling network (24800 and 32768, respectively). However, the running time of our approach for the PROSTATE-CANCER network is much slower

Table 2 Running time in seconds of PyBoolNet (PBN) and our new approach on the reduced-dynamics networks used in [22]

Network	n	e	$\|F\|$	PBN	New approach (sec)		Speedup	
					Clingo	HC-asp	Clingo	HC-asp
IL6-Signalling	55	99	32768	4.86	2.16	1.20	2.25	4.05
TLGL-Survival	58	195	3236	0.55	0.22	0.13	2.50	4.23
Colon-Cancer	66	154	52	0.12	0.02	0.04	6.00	3.00
A-Model	74	209	14	0.17	0.02	0.01	8.50	17.00
Cell-Cycle-2019	87	370	4176	10.50	0.56	0.32	18.75	32.81
PROSTATE-CANCER	116	390	24800	274.36	5.27	3.10	52.06	88.50
CASCADE3	176	468	58	0.60	0.06	0.05	10.00	12.00

than that for the IL6-Signalling network (5.27s and 2.16s, respectively). The reason may be that e/n of the PROSTATE-CANCER network is much larger than that of the IL6-Signalling network (3.36 and 1.80, respectively).

5.3 Stable configurations and cycles of circular networks

To illustrate the validity of our approach on the discovery of circular Boolean network attractors, we experimented the proposed approach on a large number of randomly generated networks. The networks are generated by choosing for each node, independently and uniformly, exactly one predecessor and one successor from the set of n nodes. The transition functions were also generated randomly by choosing each time a sign between positive and negative arcs. We then applied the presented approach to these randomly generated Boolean networks where the size is up to 7,000 nodes for positive circular networks and up to 40 nodes for negative circular networks. Figure 8 shows the running time of our approach.

We can notice that the method is very efficient as it computed the two attractors of each positive circuit very quickly. In particular, the running time is less than 60 seconds for the networks with 7,000 nodes. For the negative circuits, it computed the stable cycles for networks very quickly, in particular in less than 60 seconds for networks of 40 nodes.

The number of simple attractors (stable configurations) in the case of positive circuit graphs is always 2 whereas the size of the cyclic attractors (stable cycles) of the negative circuit graphs having n nodes is $2n$. This confirms the results that are known on attractors in Bioinformatics for the case of circular networks in [26].

6 Related work

Boolean networks were first introduced in [3]. This modeling formalism is simple yet powerful in systems biology. Boolean networks have been used to describe gene regulatory network dynamics in cases where we have good knowledge about the interactions between genes but have no good kinetic information. The dynamics of Boolean networks, particularly the attractors, generally correspond to biologically relevant phenotypes such as cell types. For example, in [9, 37], the evolution of *Arabidopsis thaliana* was modeled using Boolean networks. The

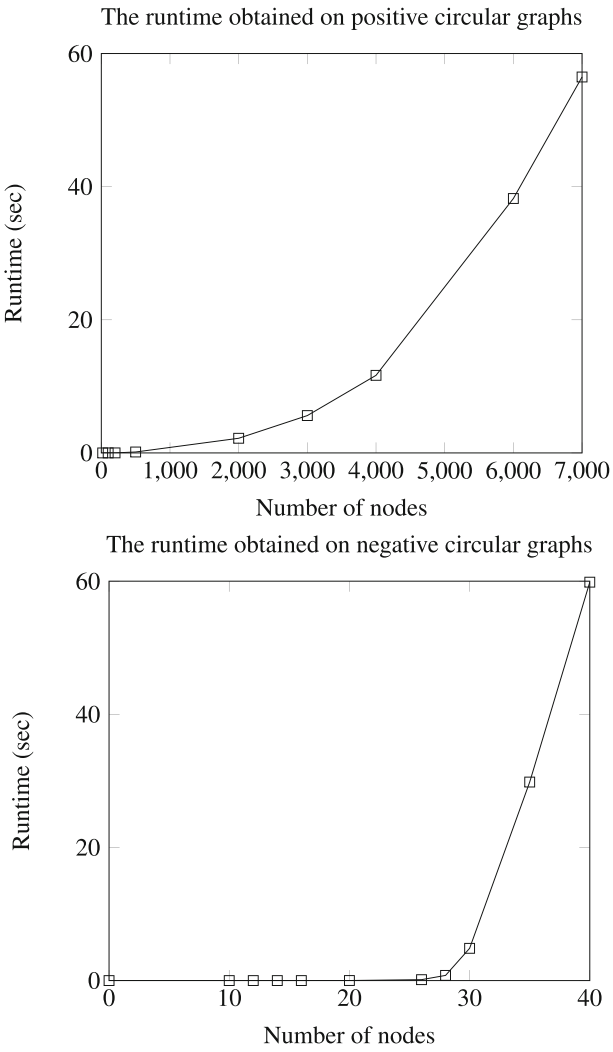


Fig. 8 The runtime obtained on the randomly generated circular graphs

attractors were shown to correspond to levels of gene expression during the stages of the development of *Arabidopsis thaliana*. In [38], the authors used a Boolean network to model the different stages of the yeast cell cycle, where the attractors were shown to correspond to phases of the process. The authors of [39] described the different states of the immune system, using a Boolean network. Boolean networks were also used in [40–42] to study the gene regulatory networks of the development of *Drosophila melanogaster*.

Several methods and tools have been developed to detect attractors in Boolean networks. In [32], the authors used Binary Decision Diagrams (BDDs) to compute the attractors of both synchronous and asynchronous Boolean networks. The implemented tool called genYsis has been widely used in the systems biology community. The tool geneFatt [24] was developed with slight improvements to genYsis. geneFAtt was reported a little more efficient for attractor

detection than genYsis. Although genYsis and geneFatt use BDDs to symbolically represent the transition graph of the Boolean network, they still rely on the traversal of the whole state space. Hence, their efficiency is limited to small to medium networks. In [33], the authors developed a mathematical approach that also uses BDDs. By using state-space pruning and random state space traversal methods, they have improved the scalability of attractor detection compared to the BDD method [32]. However, the method of [33] only computes stable configurations, whereas the BDD-based methods [24, 32] compute all attractors of a Boolean network. Berntenis et al. [43] studied the detection of attractors in large networks by limiting the detection to only relevant sub-spaces of the transition graph of an asynchronous Boolean network. Specifically, their proposed method detects the stable configurations and stable cycles with up to a given size of the asynchronous Boolean network.

Dubrova et al. [31] developed a method based on the SAT formalism to compute all attractors (i.e., stable configurations and cycles, unstable cycles) of a Boolean network under the synchronous update mode. In this approach, the attractors are searched on the transition graph of Boolean networks with SAT-Based Model Checking, in which the length of trajectories is incrementally varied. It has been shown more efficient in running time and space required than the BDD-based approach. Some slightly improved studies [44, 45] in the line of using SAT have been done, but they are all inefficient with Boolean networks where Boolean functions are complex and it is difficult to extend them to deal with the asynchronous update scheme.

Besides, ASP has been used to model Boolean networks [46–48] and thus benefited to the computation of attractors in Boolean networks [47, 48]. The work [46] concerned a use of ASP to detect and characterize inconsistency in large biological networks. It is not intended to deal with attractors at least with its current form.

The approach in [48] simulates the dynamics of gene regulatory networks expressed in the framework of Boolean networks [48, 49]. It is difficult to compare our approach with the method presented in [48]. With this method, the user must select a particular activation semantics on which the dynamic trend will be established. There are two activation semantics. The first one consists of activating a gene if at least one of its activators is active and no inhibitor is active. In the second semantics, a gene is activated if it has more expressed activators than inhibitors. The chosen activation semantics are applied to all genes, whereas our method's activation rules are specific to each gene and based on transition functions.

In [47], the authors use a fix-point semantics [50] for logic programming to characterize Boolean networks' trajectories and stable configurations. This is done by translating the Boolean networks to a logic program. At the theoretical level, there are therefore two fundamental differences with our work: on the one hand they use the semantics of the fixed point (captured by that of the stable models) and we use the semantics of the stable models and an extension of this semantics to the extra-stable models. The second difference lies in the fact that they only deal with simple attractors reduced to a stable configuration because of the semantics used, whereas in our case the extension of the semantics of stable models to extra-stable models allowed us to treat the attractors corresponding to the stable cycles. In practice, Inoue et al. only proposed a method for computing stable configurations of a Boolean network. This method has not been implemented yet. Moreover, since it requires to compute supported models (not stable models) of the ASP encoding, it is hard to implement this method directly using a more standard ASP solver.

7 Conclusion

Boolean networks are a well-established modeling technique for analyzing the dynamics of gene regulatory networks. By using Boolean networks, we can detect the attractors, which are pertinent to study cell's biological functions. We have developed an ASP based method to identify the attractors of general Boolean networks where we can detect both the stable configurations and the stable cycles for both the chosen update modes: synchronous and asynchronous. v We have also addressed the particular case of circuits that play an essential role in biological systems. Thanks to the use of the semantics introduced in [18], we were able to demonstrate several theoretical proprieties that express the characteristics and the dynamics of cyclic Boolean networks. In particular, the stable cycles of such networks have been represented in the form of linked sets of extra-stable models. The extension of stable models to extra-stable models introduced in [18] is very important for the characterization of cyclic attractors.

Using the proven theoretical results, we have designed a reliable method for the computation of attractors of Boolean networks for a chosen update mode. It is a declarative method based on the ASP paradigm that has the advantage of guaranteeing an exhaustive enumeration of all the attractors of the Boolean network considered. We have succeeded in designing a system that allows to compute all the stable models and the extra-stable models representing the stable configurations and the stable cycles of the associated transition graph, respectively. The approach for general Boolean networks have been applied to real-life gene regulatory networks, the obtained results seem promising and significant improvements were obtained. The approach dedicated to circular Boolean networks enumerates all the attractors without going through any simulation. The logical representation of a positive circuit has two stable mirror extensions (two stable models) corresponding to the two stable configurations of the transition graph. In addition, the logical representation of a negative circuit has a set of $2n$ extra-extensions inducing $2n$ extra-stable models that express the single stable cycle of the transition graph. Both the theoretical and practical results confirm the validity of our approach, since they are consistent with the results obtained in [26] but using other proof techniques.

As a perspective work, we are first interested in improving the method that deals with general Boolean network by considering some necessary optimizations in order to handle larger Boolean networks. The technique we currently use could be memory intensive when processing large networks. Second, we want to perform the stable configurations and cycle detection in a fully declarative way by adding some specific rules. Finally, for the approach dedicated to circular networks, we seek to consider other updating modes than the asynchronous mode, such as the synchronous mode and the generalization to sequential blocks, which are periodic deterministic updates.

Data Availability The benchmarks are available with the identifier https://zenodo.org/record/8103494

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. De Jong, H.: Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. **9**(1), 67–103 (2002)

2. Tran, N., Baral, C.: Hypothesizing about signaling networks. J. Appl. Log. **7**, 253–274 (2009)
3. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. J. Theor. Biol. **22**(3), 437–467 (1969)
4. Kauffman, S.A., et al.: The origins of order: Self-organization and selection in evolution. Oxford University Press, (1993)
5. Shmulevich, I., Dougherty, E.R., Zhang, W.: From boolean to probabilistic Boolean networks as models of genetic regulatory networks. Proc. IEEE **90**(11), 1778–1792 (2002)
6. Garg, A., Xenarios, I., Mendoza, L., DeMicheli, G: An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments, pp. 62–76 (2007). Springer
7. De Jong, H., Page, M.: Search for steady states of piecewise-linear differential equation models of genetic regulatory networks. IEEE/ACM Trans. Comput. Biol. Bioinforma. **5**(2), 208–222 (2008)
8. Mendoza, L.: A network model for the control of the differentiation process in Th cells. BioSyst. **84**(2), 101–114 (2006)
9. Espinosa-Soto, C., Padilla-Longoria, P., Alvarez-Buylla, E.R.: A gene regulatory network model for cell-fate determination during Arabidopsis thaliana flower development that is robust and recovers experimental gene expression profiles. Plant Cell **16**(11), 2923–2939 (2004)
10. Marek, V.W., Truszczynski, M.L.: Stable models and an alternative logic programming paradigm. Logic Programming Paradigm, pp. 375–398 (1999)
11. Lin, F., Zhao, Y.: Assat: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence, pp. 115–137 (2004)
12. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: Veloso, M.M. (ed.) IJCAI 2007, Proceedings of the 20th International joint conference on artificial intelligence, Hyderabad, India, January 6-12, 2007, p. 386 (2007)
13. Simons, P., Nimelä, I., Soininen, T.: Extending and implementing the stable model semantic. Artif. Intell. **138**, 181–234 (2002)
14. Alviano, M., Dodaro, C., Faber, W, Leone, N., Ricca, F.: Wasp: A native ASP solver based on constraint learning. In: International conference on logic programming and nonmonotonic reasoning, pp. 54–66 (2013). Springer
15. Erdem, E., Gelfond, M., Leone, N.: Applications of answer set programming. AI Mag. **37**, 53–68 (2016)
16. Khaled, T., Benhamou, B., Siegel, P.: A new method for computing stable models in logic programming. Tools with Artificial Intelligence (ICTAI), pp. 800–807 (2018)
17. Khaled, T., Benhamou, B.: Symmetry breaking in a new stable model search method. 22nd International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR-22), Kalpa Publications in Computing 9, 58–74 (2018)
18. Benhamou, B., Siegel, P.: A new semantics for logic programs capturing and extending the stable model semantics. Tools with Artificial Intelligence (ICTAI), pp. 25–32 (2012)
19. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. ICLP/SLP **50**, 1070–1080 (1988)
20. Khaled, T., Benhamou, B.: An ASP-based approach for attractor enumeration in synchronous and asynchronous Boolean networks. Proceedings 35th International Conference on Logic Programming, ICLP 2019, Las Cruces, NM, USA, 295–301 (2019)
21. Khaled, T., Benhamou, B.: An ASP-based approach for boolean networks representation and attractor detection. In: LPAR, pp. 317–333 (2020)
22. Trinh, V.-G., Hiraishi, K., Benhamou, B.: 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, pp. 1–10 (2022)
23. Jacob, F., Monod, J.: Genetic regulatory mechanisms in the synthesis of proteins. J. Mol. Biol. **3**(3), 318–356 (1961)
24. Zheng, D., Yang, G., Li, X., Wang, Z., Liu, F., He, L.: An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. PloS One **8**(4), 60593 (2013)
25. Thomas, R.: On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. In: Numerical methods in the study of critical phenomena, pp. 180–193 (1981)
26. Remy, E.., Mossé, B.., Chaouiya, C.., Thieffry, D..: A description of dynamical graphs associated to elementary regulatory circuits. Bioinforma. **19**(suppl–2), 172–178 (2003)
27. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**, 365–385 (1991)
28. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. Int. Jt. Conf. Artif. Intell. **18**, 1173–1178 (2003)

29. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**, 394–397 (1962)
30. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019)
31. Dubrova, E., Teslenko, M.: A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Trans. Comput. Biol. Bioinforma. **8**(5), 1393–1399 (2011)
32. Garg, A., Xenarios, I., Mendoza, L., DeMicheli, G.: An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments. In: Annual international conference on research in computational molecular biology, pp. 62–76 (2007). Springer
33. Ay, F., Xu, F., Kahveci, T.: Scalable steady state analysis of Boolean biological regulatory networks. PloS One **4**(12), 7992 (2009)
34. Davidich, M.I., Bornholdt, S.: Boolean network model predicts cell cycle sequence of fission yeast. PloS One **3**(2), 1672 (2008)
35. Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. Bioinform. **33**(5), 770–772 (2017). https://doi.org/10.1093/bioinformatics/btw682
36. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011). https://doi.org/10.3233/AIC-2011-0491
37. Sanchez-Corrales, Y.-E., Alvarez-Buylla, E.R., Mendoza, L.: The Arabidopsis thaliana flower organ specification gene regulatory network determines a robust differentiation process. J. Theor. Biol. **264**(3), 971–983 (2010)
38. Li, F., Long, T., Lu, Y., Ouyang, Q., Tang, C.: The yeast cell-cycle network is robustly designed. Proc. Natl. Acad. Sci. **101**(14), 4781–4786 (2004)
39. Kaufman, M., Urbain, J., Thomas, R.: Towards a logical analysis of the immune response. J. Theor. Biol. **114**(4), 527–561 (1985)
40. Sánchez, L., Thieffry, D.: A logical analysis of the Drosophila gap-gene system. J. Theor. Biol. **211**(2), 115–141 (2001)
41. Albert, R., Othmer, H.G.: The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. J. Theor. Biol. **223**(1), 1–18 (2003)
42. González, A., Chaouiya, C., Thieffry, D.: Logical modelling of the role of the Hh pathway in the patterning of the Drosophila wing disc. Bioinforma. **24**(16), 234–240 (2008)
43. Berntenis, N., Ebeling, M.: Detection of attractors of large Boolean networks via exhaustive enumeration of appropriate subspaces of the state space. BMC Bioinforma. **14**(1), 1–10 (2013)
44. He, Q., Xia, Z., Lin, B.: An efficient approach of attractor calculation for large-scale Boolean gene regulatory networks. J. Theor. Biol. **408**, 137–144 (2016). https://doi.org/10.1016/j.jtbi.2016.08.006
45. He, Q., Xia, Z., Lin, B.: P_UNSAT approach of attractor calculation for Boolean gene regulatory networks. J. Theor. Biol. **447**, 171–177 (2018). https://doi.org/10.1016/j.jtbi.2018.03.037
46. Gebser, M., Schaub, T., Thiele, S., Veber, P.: Detecting inconsistencies in large biological networks with answer set programming. Theor. Prac. Log. Program. **11**(2–3), 323–360 (2011). https://doi.org/10.1017/s1471068410000554
47. Inoue, K.: Logic programming for Boolean networks. In: 22nd International joint conference on artificial intelligence (2011)
48. Mushthofa, M., Torres, G., Van de Peer, Y., Marchal, K., De Cock, M.: Asp-g: an ASP-based method for finding attractors in genetic regulatory networks. Bioinforma. **30**(21), 3086–3092 (2014)
49. Fayruzov, T., De Cock, M., Cornelis, C, Vermeir, D.: Modeling protein interaction networks with answer set programming. In: 2009 IEEE International conference on bioinformatics and biomedicine, pp. 99–104 (2009). IEEE
50. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Foundations of deductive databases and logic programming, pp. 89–148 (1988)

Authors and Affiliations

Tarek Khaled[1] · Belaid Benhamou[1] · Van-Giang Trinh[1]

Belaid Benhamou
belaid.benhamou@univ-amu.fr

Van-Giang Trinh
trinh.van-giang@lis-lab.fr

[1] Aix-Marseille University, University of Toulon, CNRS, LIS, Marseille, France