EPiC Series in Computing

Volume 73, 2020, Pages 317–333

LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

An ASP-based Approach for Boolean Networks Representation and Attractor Detection

Tarek Khaled and Belaïd Benhamou

Aix Marseille University, University of Toulon, CNRS, LIS, Marseille, France,
{tarek.khaled,belaid.benhamou}@univ-amu.fr

Abstract

In biology, Boolean networks are conventionally used to represent and simulate gene regulatory networks. The attractors are the subject of special attention in analyzing the dynamics of a Boolean network. They correspond to stable states and stable cycles, which play a crucial role in biological systems. In this work, we study a new representation of the dynamics of Boolean networks that are based on a new semantics used in answer set programming (ASP). Our work is based on the enumeration of all the attractors of asynchronous Boolean networks having interaction graphs which are circuits. We show that the used semantics allows to design a new approach for computing exhaustively both the stable cycles and the stable states of such networks. The enumeration of all the attractors and the distinction between both types of attractors is a significant step to better understand some critical aspects of biology. We applied and evaluated the proposed approach on randomly generated Boolean networks and the obtained results highlight the benefits of this approach, and match with some conjectured results in biology.

1 Introduction

A gene regulatory network is a collection of genes that interact with each other. Each gene contains the information that determines the gene function. When a gene is active, a process called transcription takes place, producing a copy of the ribonucleic acid (RNA) of the gene information. This portion of RNA can then govern the production of a protein. A gene regulatory network is a specific biological system that represents how the proteins/genes interact in a cell for its survival, reproduction, or death. Several representations can be used to model gene regulatory networks [2]. One could use quantitative representations. But, this approach requires numerical parameters that have first to be measured or calculated, and are in general difficult to obtain. The other alternative is to use qualitative representations. This solution does not require to know the parameters that are necessary for quantitative representations[22, 7, 14]. Qualitative approaches give, in general, less precision on the dynamics of the regulatory systems than the quantitative ones. Nevertheless, they allow capturing the most important properties, like the attractors.

The Boolean networks have been proposed as a mathematical model for genetic networks[9, 10]. Boolean networks offer a simple and powerful tool to model genetic networks [19]. They reduce the representation of genetic interactions into qualitative logical rules. Boolean networks have a structure consisting of entities that correspond to genes or proteins. Each gene/protein takes a value *on* or *off*,

meaning that the gene/protein is or is not expressed. Two genes are connected if the expression of one of them modulates the expression of the other by activation or inhibition. From a logical point of view, a biological system can be considered as a set of interacting elements changing along a discrete-time.

Despite the simplified and qualitative modeling of the biological reality, it has been shown that Boolean networks express and capture correctly the dynamics of gene regulatory networks that are mostly characterized by their attractors. The attractors are the sets of states to which the system converges. An attractor generally corresponds to the observed characteristics/phenotypes of a biological system [10]. Indeed, if a network controls a phenomenon of specialization, then the cell specializes according to the attractor towards which evolves its underlying Boolean network. That is, the cell acquires a particular phenotype or a specific physiological function for the attractor where converges the Boolean network. It is then essential to identify the attractors of Boolean networks to study their dynamics.

Our goal in this work is to develop an exhaustive approach to analyze the dynamics of Boolean networks and capture all the possible, stable states and enumerating all the stable cycles. We focus here on gene networks that are represented by graph interactions that are circuits. We consider the asynchronous update mode and use the ASP framework to represent and solve the problems mentioned above. ASP [16] is a declarative problem-solving paradigm, resulting from logic programming and non-monotonic reasoning. Several answer set solvers [15, 4, 21] are available. They provide a variety of constructs and features for problem modeling [3], helping the user to express problems more naturally and solve them efficiently.

In this work, we use the method introduced in [13, 11] to deal with gene networks. This method relies on a Boolean enumeration process defined for the ASP paradigm according to the semantic introduced in [1]. This semantics guarantees for any consistent logical program, the existence of extensions or models explaining the considered program. Some of these extensions correspond to stable models and the other ones to extra-models. The extra-models correspond to extra-extensions that are not captured by the stable model semantics [5]. We will see that the extra-models play an essential role in the approach to encode the stable cycle attractors of Boolean networks and the representation of interaction graphs as logic programs interpreted in the semantics introduced in [1] give some formal results that we use to identify the attractors of the networks. Based on these formal results, we designed an algorithm for enumerating all the attractors. The detection of attractors is done without going through the simulation of Boolean networks, unlike the approach proposed in [12].

The remainder of the paper is organized as follows: We start by recalling the preliminaries of the used ASP semantics and the Boolean networks in Section 2. In Section 3, we propose a new approach for attractor search and enumeration. We study the relationships between the transition graph and the logical representation of its corresponding interaction graph in Section 4. We evaluate in Section 5 our approach on Boolean networks generated randomly. We conclude the work and give some perspectives in Section 6,.

2 Preliminaries

2.1 Boolean Networks

Let $V = \{v_1, ..., v_n\}$ be a finite set of Boolean entities $v_i \in \{0, 1\}$ (1 for the value true and 0 for the value false) representing genes in regulatory networks. A configuration $x = (x_1, \ldots, x_n)$ of the system is the assignment of a truth value $x_i \in \{0, 1\}$ to each element of V. The set of all configurations [8], also called *the space of configurations*, is denoted by $X = \{0, 1\}^n$. The dynamics of such a system is expressed by a so called *global transition function* f, and an updating mode that define how the elements of V are updated over time. The global transition function f is defined as $f : X \to X$ such that $x = (x_1, \ldots, x_n)$

$\mapsto f(x) = (f_1(x), \ldots, f_n(x))$, where each function f_i[1] $: X \to \{0, 1\}$ is *a local transition function* that gives the evolution of the state x_i of the gene v_i along time. Boolean networks may be seen as abstractions for gene regulatory networks where the boolean variables x_i represent the state of the genes v_i. The value true for x_i ($x_i = 1$) means that the corresponding gene is active, the value false ($x_i = 0$) means that the gene is inactive.

There are several update modes, and the most known ones are the synchronous and the asynchronous modes. In the synchronous mode, all the components are updated simultaneously at each step. Consequently, each state has, at most, one successor. In the asynchronous mode, only one variable can be updated at each step, and each state can have more than one successor. It has been argued in biology that the asynchronous mode is closer to real biological phenomena than the synchronous one. Indeed, the state changes occur at variable speeds and are rarely simultaneous. But, due to its complexity, fewer studies have been done for the asynchronous mode compared with the synchronous one. It is for this reason that we have chosen here to study the dynamics of asynchronous Boolean systems.

2.1.1 Transition graphs

The dynamic of a Boolean network is naturally described by a transition graph TG that is characterized 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, $f : X \to X$ its associate global transition function and $f_i : X \to X, \in \{1, \ldots, n\}$ are the local transition functions forming the function f. The transition graph representing the dynamic of f 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 $T(f) = \{(x, y) \in X^2 \mid x \neq y, x = (x_1, \ldots, x_i, \ldots, x_n), y = (x_1, \ldots, f_i(x), \ldots, x_n)\}$*

The asynchronous mode is an update in which only one component of the configuration x is updated at each time. That is, only one local transition function f_i is applied on its corresponding state gene x_i at each time. In the asynchronous mode, no assumption is formulated on the updating periods over the time. The different element of x could be updated at different time intervals. Therefore, the transitions are not deterministic. There may be several possible successor configurations for a given configuration representing a node in the transition graph.

An orbit in $TG(f)$ is a sequence of configurations $(x^0, x^1, x^2, ...)$ such that either $(x^t, x^{t+1}) \in T(f)$ or $x^{t+1} = x^t$ when there is no successors for x^t. 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 define the meaning of an attractor in dynamical systems. A state / configuration $x = (x_1, \ldots, x_n)$ of the transition graph $TG(f)$ is a stable state / configuration when $\forall x_i \in x, x_i = f_i(x)$, thus $x = f(x)$. A stable state / configuration $x = (x_1, \ldots, x_n)$ forms a trivial attractor of $TG(f)$. A sequence of states / 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. In the following, when there is no confusion, we represent the set of genes $V = \{v_1, v_2, \ldots, v_n\}$ by only their numbers $V = \{1, 2, \ldots, n\}$.

Example 1. *Consider $V = \{1, 2, 3\}$, $X = \{0, 1\}^3$ and the two following global transition functions f and g defined as $f(x_1, x_2, x_3) = (x_3, \neg x_1, x_2)$ and $g(x_1, x_2, x_3) = (\neg x_3, \neg x_1, x_2)$. The two transition graphs corresponding to both f and g are given in Figure 1. For each arc (x, y) of both transition graphs, if $x \neq y$ then the configuration x differs from the configuration y by a single component. $TG(g)$ has two stable configurations* (100) *and* (011)) *illustrated in bold in Figure 1 (b). Both simple attractors could be written as* $(1, \neg 2, \neg 3)$ *and* $(\neg 1, 2, 3)$ *when considering the corresponding genes . $TG(f)$ has a stable cycle*

[1] f_i(x) represents the local change that is carried on the state x_i of the gene v_i

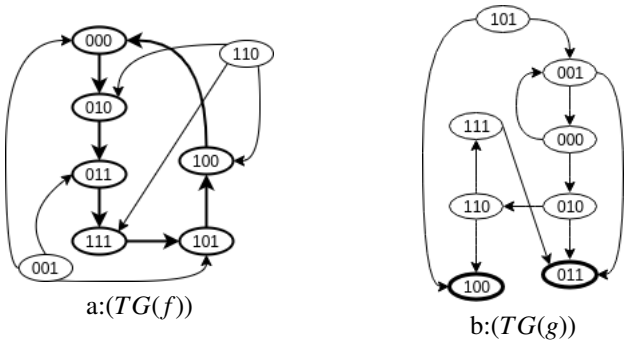

Figure 1: The transition graphs of a positive (b) and a negative (a) Boolean circuits of size 3

((000), (010), (011), (111), (101), (100)) *of six configurations pictured in bold in Figure 1 (a). This cycle attractor could be seen 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 genes.*

2.1.2 Interaction graphs

Transition graphs represent an excellent tool for studying the dynamic behavior of an update function corresponding to a Boolean network. However, in practice, biological data comes from experiments that generally give only correlations between the genes, but nothing on the dynamic of the network. Gene correlations in a gene network are classically represented by an interaction graph which is a directed graph where the arcs are labeled by the sign - or +.

Definition 2. *An interaction graph is a signed-oriented graph* $IG = (V, I)$ *where* $V = \{1, \ldots, n\}$ *is the set of vertices and* $I \subseteq V \times \{+, -\} \times V$ *is the set of signed arcs.*

Remark 1. *The vertices of the interaction graph represent the different genes of the gene regulatory network and the arcs express the interactions between them. An arc that is labeled by + is said to be positive, it denotes a positive gene interaction, while an arc that is labeled by - is said to be negative and it encodes a negative gene interaction.*

Definition 3. *A circuit of the interaction graph* $IG = (V, I)$ *of size* k *is a sequence* $C = (i_1, i_2, \ldots, i_k, i_1)$ *such that* $(i_j, \{+, -\}, i_{j+1})$ *for all* $j \in \{1, \ldots, k-1\}$ *and* $(i_k, \{+, -\}, i_1)$ *are arcs of the graph 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) is even (resp. odd), then the circuit C is positive (resp. negative)*

The interaction graph is a static representation of the regulations between the genes. Each node v_i of the interaction graph is a Boolean variable that represent the state of gene i in the network. More precisely, if $v_i = 1$ (resp. $v_i = 0$)), then the gene i is active (resp. inactive). A positive (resp. a negative) arc $(v_i, +, v_j)$ (resp. $(v_i, -, v_j)$) defined from the node v_i to the node v_j means that the gene i is an activator (resp. or an inhibitor) of the gene j. In the following, when there is no confusion we will lighten the notation by simply writing i to express the node v_i.

Example 2. *Consider the Boolean network having the set of genes* $V = \{1, 2, 3\}$*, a configuration space* $X = \{0, 1\}^3$ *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 (Figure 2 (a)) and* g *induces a positive circuit of size 3 (Figure 2 (b)). These two circuit interaction graphs correspond to the transition graphs showed in Figure 1.*

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

Figure 2: The two circuit interaction graphs corresponding to the global transition functions f and g

The interaction graphs are more compact than the transition graphs, thus more readable. But, unlike the transition graph, they give only static information about the interactions. In the framework of Boolean networks, many works aim to understand the formal relationships between interaction and transition graphs. An important research topic deals with the construction of transition graphs by using only their transition functions and their corresponding interaction graphs.

An interaction graph reduced to only one self-activating gene forms a positive circuit of length 1. If the gene is inactive, then it remains inactive forever. Inversely, if it is active, then the gene remains active all the time. Thus, its corresponding transition graph should have two stable configurations, one where the gene is active and the one other where it is inactive. This property had been generalized to interaction graphs forming a positive circuit of any size. That is, each gene of a positive circuit acts on itself positively through all the interactions of the circuit. The state of each gene can then stabilize in an active or a passive state. However, the state on which a gene i stabilizes depends on the state on which the gene j preceding i in the circuit stabilizes. For instance, if the interaction j to i is positive, and j stabilizes in an active state, then i should stabilize on an active state. Otherwise, if j stabilizes in an inactive state, then i should stabilize on an inactive state. It follows that whatever the length of the circuit, there are only two possible, stable configurations.

On the other hand, an interaction graph containing only one self-inhibiting gene forms a negative circuit of length 1. Intuitively, if the gene is active, then it inhibits itself. However, if the gene is inactive, it shall lead to its activation. It is understood that the situation of the gene oscillates between the two states. This property is conserved for interaction graphs forming negative circuits of any lengths. Each gene acts on itself via the interactions of the circuit, and its state oscillates between positive and negative.

The authors in [17] show that a positive circuit of size n admits two attractors in the asynchronous update mode, namely two stable configurations x and $\neg x$ of size n where $\neg x$ is the Boolean complementary configuration of x obtained by complementing each gene state in x. On the other hand, a negative circuit of size n admits only one attractor in the asynchronous update mode corresponding to a stable cycle formed by $2n$ configurations representing its length. From a biological point of view, the faculty of having several stable configurations or multi-stationarity can explain the ability of some cells to acquire certain phenotypes transmissible over many generations. Otherwise, negative circuits are related to the presence of stable cycles. These cycles allow the representation of the phenomena of homeostasis. The role of this phenomenon is to maintain some key factors around an optimal value (e.g., temperature, blood glucose).

2.2 Answer Set Programming

2.2.1 The new semantics for general programs

A general logic program π is a set of rules of the form $r : A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ where $A_{i\in\{0...n\}}$ is an atom and *not* the symbol expressing the negation as failure. The positive

body of the rule r is $body^+(r) = \{A_1, A_2, \ldots, A_m\}$, its negative body is $body^-(r) = \{A_{m+1}, A_{m+2}, \ldots, A_n\}$ and A_0 is its head. Several semantics are introduced in ASP to give a meaning to logic programs. In general the semantics are represented by the models that it considers. The stable models semantics [5] is one of the most used in ASP. A new semantics for general logic programs is proposed in [1]. This semantics captures the stable models semantics and extends it. It is based on a classical propositional language L where two types of variables are defined. The subset of classical variables $V = \{A_i : A_i \in L\}$ and the subset of extra-variables $nV = \{not\, A_i : not\, A_i \in L\}$. For each variable $A_i \in V$, there is a corresponding variable $not\, A_i \in nV$ designing a kind of a weak negation by failure of A_i. A connection between the two types of variables is expressed by the addition to the language L of an axiom expressing the mutual exclusion between them. This axiom of mutual exclusion is expressed by a set of binary clauses $ME = \{(\neg A_i \vee \neg not\, A_i) : A_i \in V\}$. A general logic program π=$\{r : A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n\}$, $(0 \leq m < n)$ is expressed in the propositional language L by a set of Horn clauses:

$$HC(\pi) = \{\bigcup_{r\in\pi}(A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_{m+1}, ..., \neg not\, A_n)\} \cup ME = \{(\neg A_i \vee \neg not\, A_i) : A_i \in V\} \quad .$$

The strong backdoor (STB) [23] 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 $STB = \{not\, A_i : \exists r \in \pi, A_i \in body^-(r) \subseteq nV\}$. Given a 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 positive literals $not\, A_i \in STB$ to $HC(\pi)$. Formally:

Definition 4 (see [1]). *Let $HC(\pi)$ be the Horn CNF encoding of a logic program π, STB its strong backdoor and $S' \subseteq STB$. The set* $\mathrm{E} = \mathrm{HC}(\pi) \cup \mathrm{S}'$ *of clauses is then an extension of* $(\mathrm{HC}(\pi), \mathrm{STB})$ *if the following conditions hold*

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

It is shown in [1] that each consistent $HC(\pi)$ admits at least an extension with respect to the corresponding STB, formally:

Proposition 1 (See [1]). *Let π be a logic program and STB its strong backdoor. If $HC(\pi)$ is consistent, then there exists at least an extension of the pair $(HC(\pi), STB)$.*

It is shown in [1], that the set of stable models of a logic program π is in bijection with the set of extensions E of $HC(\pi)$ that satisfy the discriminant condition $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$. The two main proved theoretical properties are given in the two following theorems:

Theorem 1 (See [1]). *If X is a stable model of a logic program π, then there exists an extension E of $(HC(\pi), STB)$ satisfying the discriminant condition $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$ such that $X = \{A_i \in V : E \models A_i\}$.*

Theorem 2 (See [1]). *If E is an extension of $(HC(\pi), STB)$, that verifies the discriminant condition $\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i$, then* $\mathrm{X} = \{\mathrm{A_i} : \mathrm{E} \models \mathrm{A_i}\}$ *is a stable model of π.*

Example 3. *Consider the logic program $\pi = \{q \leftarrow not\, r; \quad r \leftarrow not\, q; \quad p \leftarrow not\, p; \quad p \leftarrow not\, r\}$. The Horn clausal representation of the logic program π is formed by the set $HC(\pi) = CR \cup ME$ where $CR = \{q \vee \neg not\, r\,,\ r \vee \neg not\, q\,,\ p \vee \neg not\, p\,,\ p \vee \neg not\, r\}$, $ME = \{\neg a \vee \neg not\, a, \neg r \vee \neg not\, r, \neg p \vee \neg not\, p\}$ and its strong backdoor is given by $STB = \{not\, r, not\, q, not\, p\}$. We can see that $(HC(\pi), STB)$ admits two extensions $E_1 = HC(\pi) \cup \{not\, r\}$ and $E_2 = HC(\pi) \cup \{not\, p\}$. Indeed, E_1 and E_2 are maximally consistent with respect to the set STB. We can deduce by unit resolution that $E_1 \models \{\neg r, q, p, \neg not\ q, \neg not\ p\}$ and $E_2 \models \{\neg not\ r, r, \neg q, \neg not\ p, \neg p\}$. The extension E_1 satisfies the discriminant condition, but E_2 does not. Thus, the logic program has one stable model $M_1 = \{p, q\}$ deduced from E_1 by unit resolution and*

the extra-extension E_2 induces an extra-model $M_2 = \{r\}$ where r is true and both p and q are false. The stable models of π are in bijections with the extensions of $(HC(\pi), STB)$ satisfying the discriminant condition.

2.2.2 How the semantics is applied for extended programs

General logic programs allow modeling various problems. However, it turns out that many situations require classical negation. Classical negation is a concept that is extremely necessary when real problems need to be modeled in a declarative way. The semantics of an extended logic program can be defined by its reduction to a general program [6]. This reduction removes the classical negation, and then, one could use the semantic summarized previously for general programs [1] to deduce the answer sets of the input extended logic program. An extended logic program is a set of rules of the form:
$r : L_0 \leftarrow L_1, L_2, ..., L_m, not\ L_{m+1}, ..., not\ L_n, (0 \leq m < n)$ where $L_{i\in\{0...n\}}$ are literals (Atoms A_i or their negations $\neg A_i$). To reduce an extended logic program π into an equivalent general logic program π', one has to replace any negative literals $\neg L$ appearing in π by a new atom L' in π', then add the integrity constraint rule $(\leftarrow L, L')$ which prohibits L and L' to be true in the same model of π'. This avoid to L and $\neg L$ to be true in the same model of the extended logic program π. Thus, we have just to compute the stable models of the resulting general program π' from which we can deduce the answer sets of the original extended program π.

Example 4. *Let $\pi = \{b \leftarrow not\ \neg b\,, a;\quad \neg b \leftarrow not\ b;\quad a \leftarrow not\ \neg a\}$ be an extended logic program. The general logic program resulting from the translation of π is $\pi' = \{b \leftarrow not\ b', a;\ b' \leftarrow not\ b;\ a \leftarrow not\ a';\ \leftarrow a, a';\ \leftarrow b, b'\}$. The Horn clausal representation of the logic program π' is $HC(\pi') = RU(\pi') \cup ME(\pi')$ where $RU(\pi') = \{b \vee \neg not\ b' \vee \neg a,\ b' \vee \neg not\ b,\ a \vee \neg not\ a',\ \neg a \vee \neg a',\ \neg b \vee \neg b'\}$ and $ME(\pi') = \{\neg a \vee \neg not\ a,\ \neg b \vee \neg not\ b,\ \neg a' \vee \neg not\ a', \neg b' \vee \neg not\ b'\}$. Its strong backdoor is $STB = \{not\ a', not\ b, not\ b'\}$. We can see that $(HC(\pi'), STB)$ admits two extensions $E'_1 = HC(\pi') \cup \{not\ b, not\ a'\}$ and $E'_2 = HC(\pi') \cup \{not\ a', not\ b'\}$. Both extensions satisfy 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. It results that the original program π admits the two answer sets $M_1 = \{a, \neg b\}$ and $M_2 = \{a, b\}$.*

3 Representing interaction graphs as logic programs

We have seen that a dynamic of a Boolean network is represented by a transition graph TG, and the static interactions between genes are represented by an interaction graph IG. An important subject of study is to make formal links between these two representations. In what follows, we show how an interaction graph IG is expressed as a logic program P_{IG} given in its Horn clausal form $HC(P_{IG})$. We shall prove the properties related to the extensions of $HC(P_{IG})$. These properties are used to study the relationship between $HC(P_{IG})$ and the corresponding transition graph TG. We will show how to find stable states and stable cycles of the transition graph using merely P_{IG} and its features. The Boolean network formalism associates each entity $i \in \{1, \ldots, n\}$ of the regulatory network with a Boolean variable v_i. To lighten the notation, we use i instead of v_i when there is no confusion.

In [20], the authors used the Hypothesis Logic [18] to represent the interaction graph. This non-monotonic framework is powerful for knowledge representation, but does not contain effective algorithmic tools. To address this situation, we choose the ASP framework and the semantics introduced in [1] that give a good compromise between the efficiency and the power of expressiveness. The ASP solver [13] that we use here to compute the attractors of Boolean networks is based on the same semantics [1] that we used to express them.

Definition 5. *Given an interaction graph IG of a Boolean network representing a gene regulatory network, a logic program P_{IG} representing 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 $(i, +, j)$ is expressed by the rules $\{j \leftarrow not\ \neg i,\ \neg j \leftarrow not\ i\}$*
- *A negative arc $(i, -, j)$ is expressed by the rules $\{j \leftarrow not\ i,\ \neg j \leftarrow not\ \neg i\}$*

Example 5. *The interaction graph of Figure 2 (b) representing the positive circuit is expressed by an extended logic program: $P_{IG}(g) = \{2 \leftarrow not\ 1,\ \neg 2 \leftarrow not\ \neg 1,\ 3 \leftarrow not\ \neg 2,\ \neg 3 \leftarrow not\ 2,\ 1 \leftarrow not\ 3,\ \neg 1 \leftarrow not\ \neg 3\}$. The negative circuit of Figure 2 (a) is translated into the extended logic program: $P_{IG}(f) = \{2 \leftarrow not\ 1,\ \neg 2 \leftarrow not\ \neg 1,\ 3 \leftarrow not\ \neg 2,\ \neg 3 \leftarrow not\ 2,\ 1 \leftarrow not\ \neg 3,\ \neg 1 \leftarrow not\ 3\}$.*

The extended logic program P_{IG} generated is translated into a general logic program P'_{IG} that is expressed by a set of Horn clauses $HC(P'_{IG})$ in the used semantics [1]. An extension of the pair $(HC(P'_{IG}), STB)$ is the set of all consistent clauses derived from $HC(P'_{IG})$ when adding a maximal set of positive literals $not\ A_i \in STB$ to $HC(P'_{IG})$. In this context, the STB set stands for a collection of permission to activate a gene j (Resp. to inhibit a gene j). In the sequel, we will always consider the horn clausal representation $HC(P'_{IG})$ instead of the logic program P'_{IG} that we denote only by $HC(P_{IG})$ when there is no confusion. We will also say simply extensions of $HC(P_{IG})$ to mean extensions of $(HC(P'_{IG}), STB)$.

In general, it is permitted to have both $not\ \neg i$ and $not\ i$ in the used semantics, but here from the biological point of view we cannot give both the permission to activate a gene j and to inhibit it at the same time. Proposition 2 below expresses this biological aspect:

Proposition 2. *Let $HC(P_{IG})$ be a logic program representing the interaction graph IG. Then, For every $i \in V = \{1, \ldots, n\}$, $\neg(not\ \neg i \wedge not\ i)$ holds.*

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

Proposition 3. *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. □

Definition 7. *Let IG be an interaction graph having the set of entities $V = \{1, ..., n\}$ and E be an extension of $HC(P_{IG})$ obtained by adding to $HC(P_{IG})$ a maximal consistent set of literals $\{not\ i\}$ or $\{not\ \neg i\}$, with $i \in \{1, \ldots, n\}$. Then, we have the following definitions:*

1. *E is complete if for all $i \in V$, not $\neg i \in E$ or not $i \in E$*
2. *The entity $i \in V$ is free in E if $i \notin E$ and $\neg i \notin E$. Otherwise, i is linked in E.*
3. *The degree of freedom of E (denoted $deg(E)$), is the number of free element $i \in V$ in E.*
4. *The mirror of $E = HC(P_{IG}) \cup \{not\ j |\ j \in \{1 \ldots n, \neg 1, \ldots \neg n\}\}$ (denoted $mir(E)$), is defined as $mir(E) = HC(P_{IG}) \cup \{not\ \neg j |\ j \in \{1 \ldots n, \neg 1, \ldots \neg n\}\}$.*

Proposition 4. *Let $HC(P_{IG})$ be the logic program representing the interaction graph IG and E an extension of $HC(P_{IG})$. The mirror of E is also an extension of $HC(P_{IG})$.*

Proof. By definition, if *IG* contains an arc $(i, \{+, -\}, j)$, then its encoding in $HC(P_{IG})$ includes both sets of clauses $\{j \vee \neg not\ \neg i,\ \neg j \vee \neg not\ i\}$ or $\{j \vee \neg not\ i,\ \neg j \vee \neg not\ \neg i\}$. 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 have 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. □

Now we show that in some particular interaction graphs *IG* including circuits, complete extensions $HC(P_{IG})$ are of degree zero and induce answer sets of the logic encoding $HC(P_{IG})$.

Proposition 5. *Let IG be an interaction graph, and E 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 linked 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 $\{j \vee \neg not\ \neg i,\ \neg j \vee \neg not\ i\}$. 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 ($\neg j \in E$). The case of a negative arc is treated in the same way. We will have the rules $\{j \vee \neg not\ i,\ \neg j \vee \neg not\ \neg i\}$. 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, each element j is linked in E and the degree of E is 0. □

Proposition 6. *Let IG be an interaction graph, 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) $\neg not\ i \in E \Rightarrow i \in E$ and (2) $\neg not\ \neg i \in E \Rightarrow \neg i \in E$ hold. Since E is complete, then it is of degree 0 (Proposition 5). It results that either i or $\neg i$ is in E. We have two cases:

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

Now, we will prove that any answer set of $HC(P_{IG})$ correspond to an extension of degree 0.

Proposition 7. *Let IG be an interaction graph, if any node of the interaction graph 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)$\neg not\ i \in E \Rightarrow i \in E$ and (b)$\neg not\ \neg i \in E \Rightarrow \neg i \in E$ 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 2 that $\neg not\ \neg i \in E$. 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 2 we get $\neg not\ i \in E$. Thus, $i \in E$ since 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, $E \models \neg not\ i$ and $E \models \neg not\ \neg i$. 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 are possible. Thus, we have either $i \in E$ or $\neg i \in E$. □

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

Proposition 8. *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 6 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 zero, we deduce from Proposition 6 that each of them induces a stable model of $HC(P_{IG})$ of size n. □

Example 6. *Consider the extended logic program of Example 5 expressing the interaction graph of Example 2 representing the positive circuit of size 3 (Figure 2(b)):*
$P_{IG}(g) = \{2 \leftarrow not\ 1,\ \neg 2 \leftarrow not\ \neg 1,\ 3 \leftarrow not\ \neg 2,\ \neg 3 \leftarrow not\ 2,\ 1 \leftarrow not\ 3,\ \neg 1 \leftarrow not\ \neg 3\}$.
$P_{IG}(g)$ is translated to the equivalent general program:
$P'_{IG}(g) = \{2 \leftarrow not\ 1,\ 2' \leftarrow not\ 1',\ 3 \leftarrow not\ 2',\ 3' \leftarrow not\ 2,\ 1 \leftarrow not\ 3,\ 1' \leftarrow not\ 3'\}$.
$HC(P'_{IG}(g))$ has two extensions $E_1 = HC(P'_{IG}(g)) \cup \{not\ 1,\ not2',\ not3'\}$ and $E_2 = HC(P'_{IG}(g)) \cup \{not\ 1',\ not2,\ not3\}$ that correspond to two stable models. E_1 and E_2 are two extensions that verify the two discriminant conditions: $\neg not\ i \in E \Rightarrow i \in E$ and $\neg not\ i' \in E \Rightarrow i' \in E$ for all i. We remark that E_1 is complete because for all $i \in HC(P'_{IG}(g))$, either not i' (not $\neg i$) belongs to E_1 or not i belongs to E_1. Also, we have either $i \in E_1$ or $i' = \neg i \in E_1$ for all $i \in HC(P'_{IG}(g))$, meaning that the degree of freedom of E_1 is 0.

The extension E_2 is the mirror of E_1. The stable models induced by E_1 and E_2 are $M'_1 = \{1',\ 2,\ 3\}$ and $M'_2 = \{1,\ 2',\ 3'\}$. The corresponding answers sets of the extended program $P_{IG}(g)$ are $M_1 = \{\neg 1,\ 2,\ 3\}$ and $M_2 = \{1,\ \neg 2,\ \neg 3\}$. We can see that the two previous answers sets correspond to the two stable configurations of the transition graph (Figure 1-(b)) of the positive circuit of Example 2.

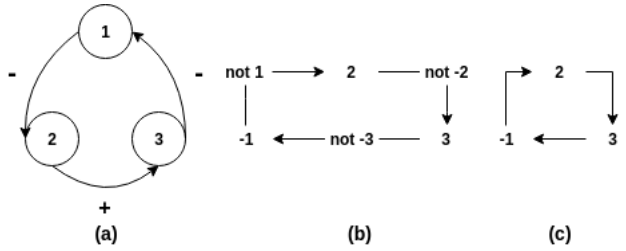

Figure 3: (a) $IG(f)$, (b) Construction of E_1 (c) The graph of M_1

An intuition of the computation of E_1 is given by the construction process described in Figure 3(b). The interaction graph is depicted in Figure 3(a) while Figure 3(b) gives the different construction steps of E_1. Initially, E_1 is empty. We start the process by supposing *not* 1 in E_1. Thus, by application of the rule $2 \leftarrow not\ 1$ we deduce 2 and then $\neg not\ 2$ is inferred from the mutual exclusion clause $(\neg 2 \vee \neg not\ 2)$. The construction of E_1 goes on by adding *not* $\neg 2$ to E_1 and then we deduce 3. The mutual exclusion $(\neg 3 \vee \neg not\ 3)$ prohibits the application of *not* 3. Then, we add *not* $\neg 3$ to E_1 from which we deduce $\neg 1$.

If we are interested only on the gene literals i, then we obtain the restricted graph of E_1 depicted in Figure 3(c) representing the corresponding stable model M_1. This model corresponds to one of the two stable configurations of the corresponding transaction graph of Example 1. The extension E_2 is build in a similar way as E_1. To get E_2, one has to start the process by supposing of *not* $\neg 1$ in E_2.

From the biological point of view, the variables of the answer sets represent the state of each gene of the regulatory network. For instance $M_1 = \{\neg 1,\ 2,\ 3\}$; expresses the fact 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 will show that each interaction graph *IG* representing a negative circuit of n nodes, has $2n$ extra-extensions of degree 1 inducing $2n$ extra-models that encode a stable cycle of size $2n$ in the corresponding graph transition.

Proposition 9. *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-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 we initially 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 deduce i and $\neg i$ simultaneously. This deduction means that we cannot have a complete extension in both cases.

Then, we obtain an incomplete extension as well as its mirror extension where there is neither the literal *not* j nor the literal *not* $\neg j$ with j being the predecessor of i. Neither i nor $\neg i$ is true in the two obtained extensions. On the other hand, all the other elements different from i are linked in the two extensions in question. 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, in the obtained extension, we have $\neg$*not* i without having i, and in its mirror extension, we have $\neg$*not* $\neg i$ without having $\neg i$.

Therefore, we have two mirror extra-extensions of degree 1 inducing two extra-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-models of sizes $n-1$. In total, there will therefore be $2n$ extra-extensions of degree 1 inducing $2n$ extra-models of sizes $n-1$. □

Example 7. *Consider the extended logic program of Example 5 expressing the interaction graph of Example 2 corresponding to a negative circuit of size 3 (Figure 2(a)):*
$P_{IG}(f) = \{2 \leftarrow not\ 1,\ \neg 2 \leftarrow not\ \neg 1,\ 3 \leftarrow not\ \neg 2,\ \neg 3 \leftarrow not\ 2,\ 1 \leftarrow not\ \neg 3,\ \neg 1 \leftarrow not\ 3\}$.
After translation, we get the following general logic program:
$P'_{IG}(f) = \{2 \leftarrow not\ 1,\ 2' \leftarrow not\ 1',\ 3 \leftarrow not\ 2',\ 3' \leftarrow not\ 2,\ 1 \leftarrow not\ 3',\ 1' \leftarrow not\ 3\}$.
The logic encoding $HC(P'_{IG}(f))$ *has* (E'_i) *six extra-extensions that correspond to six extra models* (M'_i)*:*
$E'_1 = HC(P'_{IG}(f)) \cup \{not\ 1,\ not\ 2'\} \Rightarrow E'_1 \models \{not\ 1, 2, not\ 2', 3\} \Rightarrow M'_1 = \{2, 3\}$
$E'_2 = HC(P'_{IG}(f)) \cup \{not\ 1,\ not\ 3\} \Rightarrow E'_2 \models \{not\ 1, 2, not\ 3, 1'\} \Rightarrow M'_2 = \{2, 1'\}$
$E'_3 = HC(P'_{IG}(f)) \cup \{not\ 1',\ not\ 3'\} \Rightarrow E'_3 \models \{not\ 1', 2',\ not\ 3', 1\} \Rightarrow M'_3 = \{2', 1\}$
$E'_4 = HC(P'_{IG}(f)) \cup \{not\ 1',\ not\ 2\} \Rightarrow E'_4 \models \{not\ 1', 3',\ not\ 2, 3'\} \Rightarrow M'_4 = \{2', 3'\}$
$E'_5 = HC(P'_{IG}(f)) \cup \{not\ 2',\ not\ 3'\} \Rightarrow E'_5 \models \{not\ 2', 3,\ not\ 3', 1\} \Rightarrow M'_5 = \{3, 1\}$
$E'_6 = HC(P'_{IG}(f)) \cup \{not\ 2,\ not\ 3\} \Rightarrow E'_6 \models \{not\ 2, 3', not\ 3, 1'\} \Rightarrow M'_6 = \{3', 1'\}$

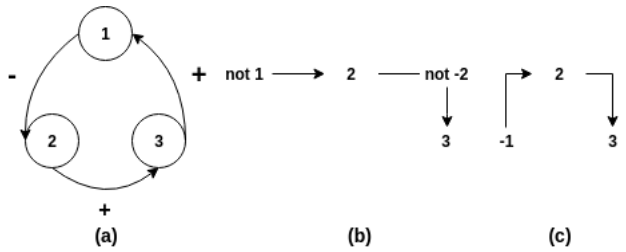

Figure 4: (a) $IG(g)$ the negative circuit, (b) construction of E'_1 (c) construction of M'_1 .

Figure 4(a) gives the negative circuit that we expressed as a logic program. Figure 4(b) shows the construction of the extension E'_1. It is constructed by adding to $HC(P'_{IG}(f))$ both literals *not* 1, and *not* $2'$. We can see in Figure 4(b), that it is impossible to deduce $1'$. Indeed, to obtain $1'$, we should use the rule ($1' \leftarrow$ *not* 3). But this is impossible since $\neg$*not* 3 results from the mutual exclusion ($\neg 3 \vee \neg$*not* 3). On the other hand, we can not get 1. As *not* 1 holds, then from the mutual exclusion ($\neg 1 \vee \neg$*not* 1) we get $\neg 1$. Thus, we cannot have 1. We can remark 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, since $1 \notin E'_1$ and $\neg 1 \notin E'_1$. It results that, E'_1 is an extension of degree 1. Figure 4(c) gives the restriction of E'_1 to the corresponding extra-model M'_1.

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

In this section, we study the relationship between the logical representation $HC(P_{IG})$ of the interaction graph IG and its corresponding transition graph TG. To do that, we will see that the vertices of the transition graph TG corresponding to stable configurations or to stable cycles could represent in fact, extensions / extra-extensions (answer sets or extra-models) of the logical encoding $HC(P_{IG})$.

Given a Boolean network, having IG as its interaction graph, TG its corresponding transition graph, and $HC(P_{IG})$ the horn clausal representation of the logic program P_{IG}, we will show for positive circuits (Theorem 3) that there is an isomorphism between the stable configurations of TG and the answer sets of $HC(P_{IG})$. Moreover, we shall also prove (Theorem 4) that any stable cycle of the transition graph corresponding to a negative circuit interaction graph of size n is encoded as a set of $2n$ extra models of degree 1 of $HC(P_{IG})$.

Proposition 10. *Given a Boolean network represented by the interaction graph IG where TG is the transition graph associated with IG and $HC(P_{IG})$ 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 will be k accessible vertices from s. □

Theorem 3. *Given a Boolean network where IG is the interaction graph and $HC(P_{IG})$ 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 $s = (x_1, x_2, \ldots, x_n)$ the vertex representing E in TG. As E is an extension, then its degree of freedom is 0. According to Proposition 10, it follows that the only accessible node from s is itself. Thus, s is a stable state of TG.

2. Now if $s = (x_1, x_2, \ldots, x_n)$ is a stable state 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 7 that $s = (x_1, x_2, \ldots, x_n)$ forms an answer set of P_{IG}.

□

Example 8. *The right part of Figure 5 shows both extensions obtained for the logic program corresponding to the positive circuit of Example 5. We can see that these extensions induce two answer sets that encode both stable configurations of the transition graph (left part of Figure 5) drawn in bold font.*

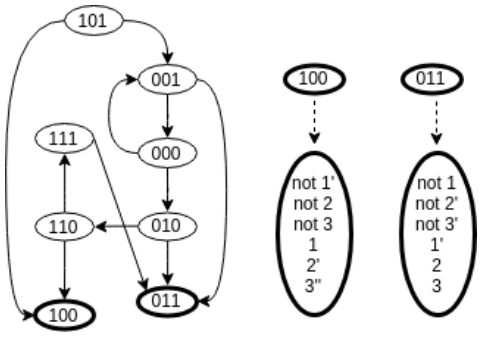

Figure 5: The stable configurations of TG expressed as stable models of $HC(P_{IG})$.

Theorem 4. *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 of the associated transition graph TG.*

Proof. Proposition 9 guarantees the existence of $2n$ extra-extension (extra-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 9. *The right part of Figure 5 shows the extra-extensions obtained for the logic program corresponding to the negative circuit of Example 5. We can see that six extra-extensions of degree 1 inducing six extra-models are found and each of them identifies a configuration of the stable cycle of the corresponding transition graph given in bold font (Left part of Figure 5).*

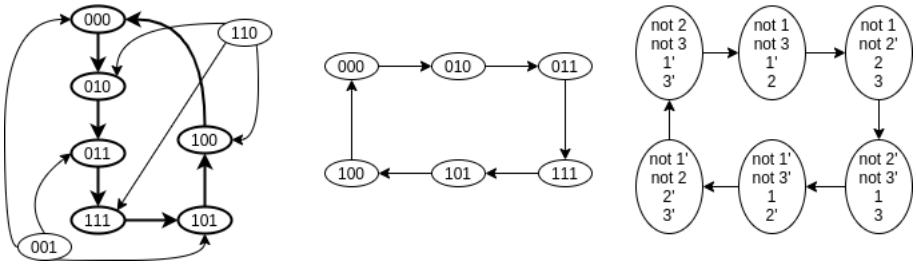

Figure 6: A stable cycle of TG seen as a set of $2n$ extra-extensions of $HC(P_{IG})$.

5 Empirical Validation

To demonstrate the validity of our approach on Boolean network attractor discovery, we applied it on some randomly generated circular networks. The circular networks of size n are generated by selecting for a current node $i \in \{1, 2, \ldots, n\}$ independently and uniformly exactly one successor $j \in \{1, 2, \ldots, i-1, i+1, \ldots, n\}$. The label s of the corresponding arc (i, s, j) is generated by choosing randomly a sign between positive and negative ($s \in \{+, -\}$). The process is repeated for node j and all the successor

nodes generated up to the last node, which must have as successor the starting node i making the graph cyclical.

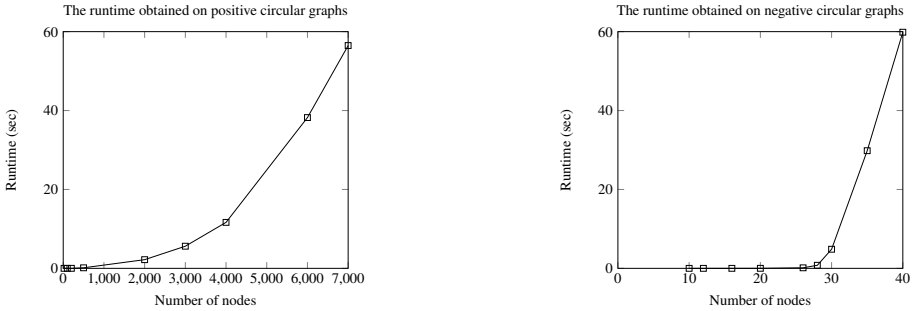

Figure 7: The CPU times of the attractors of the positive and negative circuits

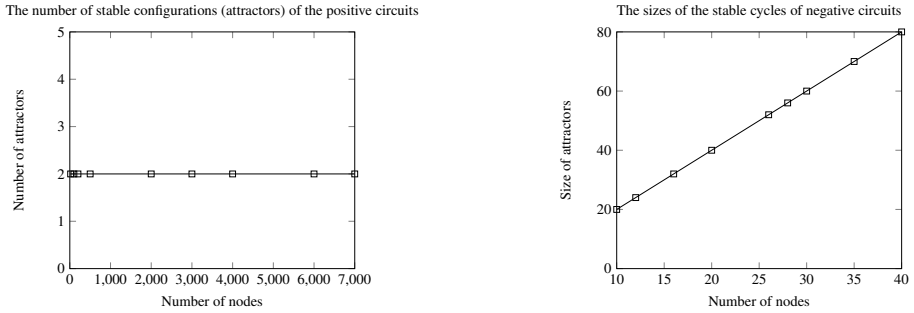

Figure 8: The number / size of attractors obtained on the randomly generated circular graphs

In this experiment, we applied the proposed approach to randomly generated Boolean networks up to 7000 nodes for positive circular networks and up to 40 nodes for the negative circular ones. The resulting runtimes to get all the attractors of each instance are shown in Figure 7. We can see that the method is powerful; it computes the two attractors (stable configurations) of positive circuits having 7000 nodes in less than 60 seconds. For the negative circuits it calculates the stable cycles for networks with a size up to 40 nodes in less than 60 seconds. We can also see in Figure 8 that the number of stable configurations (attractors) for all the generated positive circuits is 2, while for the negative circuits, there is only one stable cycle of size $2n$ for each graph instance. These experimental results correlate well with biology and confirm the validity of the theoretical properties demonstrated in this article.

6 Conclusion

Boolean networks represent a widespread modeling technique for analyzing the dynamic behavior of gene regulatory networks. By using Boolean networks, we can capture network attractors, which are often useful for studying the biological function of a cell. We discussed in this paper the particular case of circuits that plays an essential role in biological systems. We proved several properties that establish a correspondence between the attractors of transition graphs and the stable models / extra-models of the logic programs expressing the circular interaction graphs. In particular, the representation of stable cycles of negative circuits by a set of extra-models shows the advantage of the extension offered by the semantics used to those of stable models [5].

In addition to the theoretical results, we proposed an approach that computes efficiently all the answer sets / extra-models representing the stable configurations or the stable cycles of the transition graph of the considered Boolean network. The fact that the logical representation of a positive circuit has two stable mirror models and that one of a negative circuit has a set of $2n$ extra-models gives a good witness for the validity of the method since this corresponds to the known results in biology [17]. The correlation of our results with those of biology has been demonstrated theoretically and experimentally.

As future work, we are looking to take into account other modes of updating like the parallel mode, then generalize the study to the sequential blocks, which are deterministic periodical updates. On the other hand, we are interested in characterizing non-stable cycles in Boolean networks.

References

[1] Belaïd Benhamou and Pierre Siegel. A new semantics for logic programs capturing and extending the stable model semantics. *Tools with Artificial Intelligence (ICTAI)*, pages 25–32, 2012.

[2] Hidde De Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103, 2002.

[3] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37:53–68, 2016.

[4] Martin Gebser, Benjamin Kaufmann, Andrá Neumann, and Torsten Schaub. Conflict-driven answer set solving. *IJCAI*, 7:386–392, 2007.

[5] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. *ICLP/SLP*, 50:1070–1080, 1988.

[6] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9:365–385, 1991.

[7] Katsumi Inoue. Logic programming for boolean networks. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[8] François Jacob and Jacques Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of molecular biology*, 3(3):318–356, 1961.

[9] Stuart A Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467, 1969.

[10] Stuart A Kauffman. *The origins of order: Self-organization and selection in evolution*. OUP USA, 1993.

[11] Tarek Khaled and Belaid Benhamou. 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.

[12] Tarek Khaled and Belaid Benhamou. 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*, pages 295–301, 2019.

[13] Tarek Khaled, Belaïd Benhamou, and Pierre Siegel. A new method for computing stable models in logic programming. *Tools with Artificial Intelligence (ICTAI)*, pages 800–807, 2018.

[14] Hannes Klarner, Alexander Bockmayr, and Heike Siebert. Computing maximal and minimal trap spaces of boolean networks. *Natural Computing*, 14(4):535–544, 2015.

[15] Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, pages 115–137, 2004.

[16] Victor W. Marek and Miros L. Truszczynski. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm*, pages 375–398, 1999.

[17] Elisabeth Remy, Brigitte Mossé, Claudine Chaouiya, and Denis Thieffry. A description of dynamical graphs associated to elementary regulatory circuits. *Bioinformatics*, 19(suppl_2):ii172–ii178, 2003.

[18] Camilla Schwind and Pierre Siegel. A modal logic for hypothesis theory. *Fundamenta Informaticae*, 21:89–101, 1994.

[19] Ilya Shmulevich, Edward R Dougherty, and Wei Zhang. From boolean to probabilistic boolean networks as models of genetic regulatory networks. *Proceedings of the IEEE*, 90(11):1778–1792, 2002.

[20] Pierre Siegel, Andrei Doncescu, Vincent Risch, and Sylvain Sené. Towards a boolean dynamical system representation in a monmonotonic modal logic. 2018.

[21] Patrik Simons, Ilkka Nimelä, and Timo Soininen. Extending and implementing the stable model semantic. *Artificial Intelligence*, 138:181–234, 2002.

[22] Nam Tran and Chitta Baral. Hypothesizing about signaling networks. *Journal of Applied Logic*, 7:253–274, 2009.

[23] Ryan Williams, Carla P Gomes, and Bart Selman. Backdoors to typical case complexity. *International joint conference on artificial intelligence*, 18:1173–1178, 2003.