Computing Attractors of Multi-Valued Gene Regulatory Networks using Fuzzy Answer Set Programming

Mushthofa Mushthofa[*,†,¶] Steven Schockaert[‡] and Martine De Cock[*,§]
[*]Department of Applied Mathematics, Statistics and Informatics, Ghent University, Ghent, Belgium,
Email: {Mushthofa.Mushthofa, Martine.DeCock}@UGent.be
[†]Department of Computer Science, Bogor Agricultural University, Bogor, Indonesia,
Email: mush@ipb.ac.id
[‡]School of Computer Science & Informatics, Cardiff University, UK,
Email: SchockaertS1@cardiff.ac.uk
[§]Institute of Technology, University of Washington Tacoma, USA,
Email: mdecock@uw.edu
[¶]Bioinformatics Institute Ghent, Ghent University, 9052 Ghent, Belgium

***Abstract*—Fuzzy Answer Set Programming (FASP) extends the popular Answer Set Programming (ASP) paradigm to modeling and solving combinatorial search problems in *continuous* domains. The recent development of FASP solvers has turned FASP into a practical tool for solving real-world problems. In this paper, we propose the use of FASP for modeling the dynamics of Gene Regulatory Networks (GRNs), an important kind of biological network. A commonly used simplifying assumption to model the dynamics of GRNs is to assume only Boolean levels of activation of each node. ASP has been used to model such Boolean networks. Our work extends this Boolean network formalism by allowing multi-valued activation levels. We show how FASP can be used to model the dynamics of such networks. We also experimentally assess the plausibility of our method using real biological networks found in the literature.**

I. Introduction

Answer Set Programming (ASP) is a popular declarative programming paradigm [1] which allows for an easy and intuitive encoding of many combinatorial search and optimisation problems. The availability of fast and efficient solvers for ASP, such as clasp [2] and DLV [3], allows for the application of ASP in various fields [4]. Despite its flexibility and expressive power, ASP lacks the ability to directly encode problems in the continuous domain, due to the fact that it is based on Boolean logic.

Fuzzy Answer Set Programming (FASP) [5] is an extension of ASP in which multi-valued semantics is used for evaluating propositions. Recent progress, both in theoretical aspects [6] and the development of solvers [7]–[10] has made it possible to apply FASP to solve real-world problems. To demonstrate this claim, in this paper we look at one particular application, viz. modelling the stable states of biological networks.

Several authors have looked at using ASP for modelling biological networks [11]–[13], and in particular for computing the so-called attractors [14]. An attractor in the context of biological networks represents the states to which the dynamics of the network converges, and usually corresponds to the observed characteristics/phenotypes in biological systems [14], [15]. For example, the attractors of a Gene Regulatory Network (GRN) usually correspond to the expression patterns of the genes in the network for specific types of cells [16], [17]. It is thus of importance to study the dynamics of such network and to be able to identify the attractors of a given network.

A commonly used simplifying assumption to model the dynamics of biological networks is to assume only Boolean levels of activation of each node, i.e., a node in the network is either on or off. Dynamic networks with Boolean levels of activation are commonly referred to as Boolean networks. Many tools have been developed to model the dynamics and to compute the attractors of BNs [18]–[22], including using ASP [11], [23]. While considering Boolean levels of activation may be enough for many cases, there are cases where having only two levels of activation is not enough to fully understand the dynamics of the real biological systems, e.g., when the observed attractors of the network have multi-valued activation levels [17], [24]–[27].

In this paper, we consider the extension of the notion of Boolean networks in the fuzzy/multi-valued domain. In particular, we propose a method to simulate the dynamics of the multi-valued networks and to compute the attractors using FASP. We provide an encoding of the problem in the language of FASP that can be executed/solved using the currently available FASP solver, and prove the correctness of such encoding. We also perform a benchmark test of the encoding using real biological networks.

II. Preliminaries

A. Boolean networks

A Boolean network [14] is a tuple $G = \langle X, F \rangle$, where $X = \langle x_1, \ldots, x_n \rangle$ is a tuple of Boolean variables representing

978-1-5090-0626-7/16/$31.00

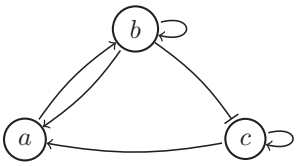

Fig. 1. A Boolean network model with three genes. Edges with arrowed tips are activating interactions and edges with blunt tips are repressing (inhibiting) links.

the nodes of the network, while $F = \langle f_1, \ldots, f_n \rangle$ is a tuple of Boolean functions. An assignment $V : X \to \{0, 1\}$ is called a *network state*. The set of all $2^{|X|}$ network states is called the *state space* of the Boolean network, denoted by S. Each function f_i is a Boolean expression over the constants 0 and 1 and the set of variables in X. The value of the expression f_i, given the assignment V for the variables in X is denoted by $f_i(V)$. The tuple F of functions defines the network update function $f : S \to S$ as follows: the state $f(V)$ for a state V is the state W such that $W(x_i) = f_i(V)$ for any $x_i \in X$.

The dynamics of a Boolean network are defined by the transitions between the network states as determined by the given *update rule* used in the network. Two types of update rules are usually discussed in the literature: *synchronous* and *asynchronous* update [14], [28]. Furthermore, we can differentiate between two types of asynchronicity: *1-asynchronous* and **-asynchronous*. To further explain the concept of update rules, we first define the following. The Hamming distance function over pairs of valuations/states, $\Delta : S \times S \to \{1, \ldots, n\}$ is defined as follows .

$$\Delta(v, w) = |\{x \in X \mid v(x) \neq w(x)\}|$$

The dynamics of a Boolean network are represented as a directed graph $\langle S, \hookrightarrow \rangle$, called the State Transition Graph (STG), where the edge relation $\hookrightarrow$ is determined by the update rule used, as follows:

(i) For the synchronous update rule: $v \hookrightarrow w$ iff $f(v) = w$.
(ii) For the 1-asynchronous update rule: $v \hookrightarrow w$ iff either $v = w$ and $f(v) = v$, or $\Delta(v, w) = 1$ and $\Delta(w, f(v)) < \Delta(v, f(v))$
(iii) For the *-asynchronous update rule: $v \hookrightarrow w$ iff either $v = w$ and $f(v) = v$, or $1 \leq \Delta(v, w) \leq |X|$ and $\Delta(w, f(v)) < \Delta(v, f(v))$

Intuitively, in the synchronous update rule, the network transitions from a state to another state by applying *all* of the update functions to all of the nodes. Conversely, in the asynchronous state, the transition from a state to another is done by applying the update functions to only *some* of the nodes. Specifically, in the 1-asynchronous case, only one update function is applied, while in the *-asynchronous case, any (non-zero) number of update functions are applied. The condition $\Delta(w, f(v)) < \Delta(v, f(v))$ intuitively means that after applying the update functions to one node (in the case of 1-asynchronous) or some nodes (in the case of *-asynchronous), the new state w should be at least as close to $f(v)$ as v, since the updated node(s) in w should have the same values as in $f(v)$.

Given an STG $\langle S, \hookrightarrow \rangle$, a path is a sequence of states $(s_1, s_2, \ldots, s_k)$ where $s_i \hookrightarrow s_{i+1}$ for every $i \in \{1, \ldots, k-1\}$. A non-empty set $T \subseteq S$ is a *trap set* if for every $s \in T$ and t s.t. $s \hookrightarrow t$, it holds that $t \in T$. A minimal trap set w.r.t. set-inclusion is called an *attractor* of the Boolean network. If $T = \{x\}$ is a single state attractor, then x is called a *steady state*. Otherwise, T is usually called a *cyclic* attractor. Note that, in general, the attractors of a Boolean network under different update rules are also different. Steady states, however, do not depend on the particular choice of update rule. This is due to the fact that in a steady state x, $f(x) = x$, which means that $\{x\}$ is an attractor w.r.t. all three update rules.

Traditionally, Boolean networks have been used to model the dynamics of gene regulatory networks [14], [29], [30]. Nodes in the network represent the genes, while the network states represent the activation levels of the genes at a particular point in time. The interactions between genes (both activating and inhibiting interactions) can be encoded using the update function of the Boolean network. The dynamics of the Boolean network then represent the dynamics of the regulations between the genes, while (some of) the attractors of the network correspond to the observed state of the genes in a gene regulatory network. In particular, the states in an attractor usually capture the expression levels of the genes to which the dynamics of the regulation in the network converges, and thus hint towards the functional modes of the regulatory network [14], [15].

Example 1. *Consider the Boolean network* $G_1 = \langle \{a, b, c\}, F \rangle$*, depicted in Figure 1. The Boolean functions* F *describing the interaction between nodes in the network are given by*

$$a_{t+1} = b_t \vee c_t$$
$$b_{t+1} = b_t \wedge a_t$$
$$c_{t+1} = c_t \wedge \neg b_t$$

The dynamics of the network under the synchronous update rule can be described using the STG given in Figure 2. For example, starting from the state $\langle 0, 1, 0 \rangle$*, we move to the state* $\langle 1, 0, 0 \rangle$*, i.e.,* $f(\langle 0, 1, 0 \rangle) = \langle 1, 0, 0 \rangle$*. From the figure, we can see that the Boolean network has 3 attractors, all of which coincidentally have size = 1, giving us exactly 3 steady states:* $\langle 0, 0, 0 \rangle$*,* $\langle 1, 0, 1 \rangle$*, and* $\langle 1, 1, 0 \rangle$*.*

From the definition of asynchronous updates, we can derive that the STG of a Boolean network may no longer be deterministic, since from a state x*, there can be more than one outgoing edge. Figure 3 depicts part of the STG of the example Boolean network when the 1-asynchronous and *-asynchronous update rules are used (corresponding only to the outgoing transitions of states* $\langle 0, 0, 0 \rangle$*,* $\langle 0, 0, 1 \rangle$ *and* $\langle 0, 1, 0 \rangle$*). We can also verify that when using the 1-asynchronous or the *-asynchronous update, the states* $\langle 0, 0, 0 \rangle, \langle 0, 1, 0 \rangle$ *and* $\langle 0, 0, 1 \rangle$ *are also steady states of the network.*

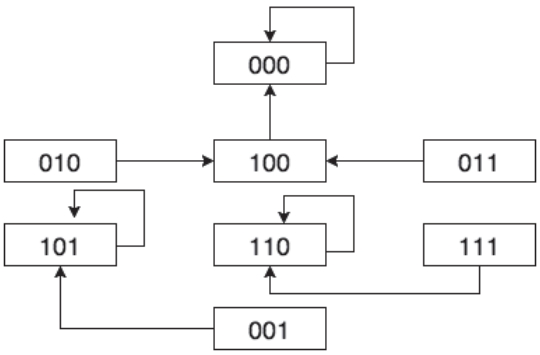

Fig. 2. State Transition Graph of the example Boolean network under the synchronous update rule

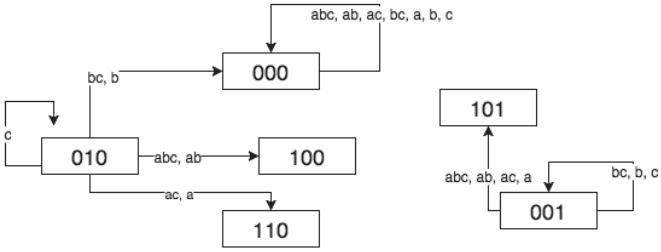

Fig. 3. Part of the State Transition Graph of the example Boolean network under the asynchronous update rule. The labels on the edges represent the choices of the nodes that are updated. For example, the label abc, ab for a transition $x \hookrightarrow y$ means that updating either all of the nodes a, b, c or just a and b will result in a transition from state x to y. For 1-asynchronous update, only single-variable labels are relevant.

B. Answer Set Programming

Answer Set Programming (ASP) is a logic-based programming paradigm commonly used to solve combinatorial search and optimisation problems [1]. An ASP program consists of a set of rules of the form

$$r \equiv a_1 \vee \ldots \vee a_n \leftarrow b_1 \wedge \ldots \wedge b_k \wedge \textbf{not}\ c_1 \ldots \wedge \textbf{not}\ c_m$$

where each a_i, b_i and c_i is a Boolean propositional symbol taken from the universal set of symbols $\mathcal{B}$, and **not** is the negation-as-failure operator. A literal is either an atomic proposition a (positive literal) or a Negation-As-Failure (NAF) literal **not** a (negative literal). We also write $H(r) = \{a_1, \ldots, a_n\}$, $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{c_1, \ldots, c_m\}$ to indicate the sets of literals appearing in a rule r.

An interpretation $I \subseteq \mathcal{B}$ is said to *model* a rule r iff $I \cap H(r) \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. An interpretation is a model of a program P iff it is a model of every rule $r \in P$.

If $B^-(r) = \emptyset$, the rule r is said to be positive. If all of the rules of an ASP program P are positive, we say that P is a positive program. For a positive program P, we define the answer sets of P as the minimal models (w.r.t. set inclusion) of P. For non-positive programs P, the Gelfond-Lifschitz [31] transform of P w.r.t. an interpretation I is obtained by:

(i) deleting all rules r s.t. $B^-(r) \cap I \neq \emptyset$
(ii) deleting any of the remaining expressions of the form **not** a in the program.

In general, an interpretation I is an answer set of P iff it is an answer set of the Gelfond-Lifschitz reduct of P w.r.t. I.

Example 2. *Consider the ASP program P_1 having the following rules:*

$$r_1 : open \leftarrow \textbf{\textit{not}}\ close$$
$$r_2 : close \leftarrow \textbf{\textit{not}}\ open$$

The program has exactly two answer sets: $A_1 = \{open\}$ and $A_2 = \{close\}$.

An ASP program is usually written to describe/encode a problem that needs to be solved, and then submitted to an ASP solver, such as Potassco [2] and DLV [3]. The solver will then produce/compute the answer set(s) of the program, which in turn correspond to the intended solution(s) of the problem. ASP has found applications in many different fields [32], including planning [33], Semantic Web [4] and bioinformatics/computational biology [11], [12], [23].

C. Fuzzy answer set programming

Fuzzy Answer Set Programming (FASP) [5] is an extension of ASP into the fuzzy domain, where atomic propositions in $\mathcal{B}$ can take a graded truth value and rules are defined using fuzzy logic connectives. In this case, an interpretation is defined as a function $I : \mathcal{B} \rightarrow [0, 1]$. In this paper, we will only consider the Łukasiewicz connectives [6], [7], [34], defined as follows:,

- $I(\alpha \otimes \beta) = \max(I(\alpha) + I(\beta) - 1, 0)$.
- $I(\alpha \oplus \beta) = \min(I(\alpha) + I(\beta), 1)$.
- $I(\alpha \veebar \beta) = \max(I(\alpha), I(\beta))$.
- $I(\alpha \barwedge \beta) = \min(I(\alpha), I(\beta))$.
- $I(\neg\alpha) = 1 - I(\alpha)$.
- $I(\beta \rightarrow \alpha) = \min(1 - I(\beta) + I(\alpha), 1)$.

In a FASP program, a head expression is an expression of the form $a_1 \oplus a_2 \oplus \ldots \oplus a_n$, where a_i's are literals, while a body expression is an expression defined recursively as follows:

- A constant term $\bar{c}$ where $c \in \{0, 1\}$, a positive literal a and a negative literal **not** a are body expressions.
- If a and b are body expressions, then so are $a \oplus b$, $a \otimes b$, $a \veebar b$ and $a \barwedge b$.

A FASP program consists of rules of the form

$$\alpha \leftarrow \beta$$

where α is a head expression and β is a body expression. We sometimes write $Head(r)$ and $Body(r)$ to denote the head and body expressions of the rule r, respectively. A FASP rule is said to be positive iff it contains no applications of the **not** operator. A FASP program is positive iff it only contains positive rules.

An interpretation I is a model of a rule r iff $I(r) = 1$, and I is a model of a program P iff $I(r) = I(Body(r) \rightarrow Head(r)) = 1$ for every $r \in P$. We write $I \leq J$ for two interpretations I and J iff $I(a) \leq J(a)$ for any $a \in \mathcal{B}$.

Furthermore, we define $I = J$ whenever $I \leq J$ and $J \leq I$, while $I < J$ is defined as $I \leq J$ but $I \neq J$. A model I of a positive program P is an answer set of P iff there is no model J of P s.t. $J < I$. For a non-positive program P, a generalization of the so-called Gelfond-Lifschitz reduct is defined in [35] as follows: the reduct of a rule r w.r.t. an interpretation I is the positive rule r^I obtained by replacing each occurrence of **not** a by the constant $\overline{I(\textbf{not } a)}$. The reduct of a FASP program $\mathcal{P}$ w.r.t. an interpretation I is then defined as the positive program $\mathcal{P}^I = \{r^I \mid r \in \mathcal{P}\}$. A model I of $\mathcal{P}$ is called an answer set of $\mathcal{P}$ iff I is an answer set of $\mathcal{P}^I$. Although the development of solvers for FASP is still not on par with the development of ASP solvers, the recent availability of FASP solvers such as the ones in [7]–[10] opens the door to real-world applications.

Following [7], we consider the finite-valued answer sets of a FASP program P, by restricting the values of the interpretation function I to the set $\mathbb{Q}_k = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$. Any answer set derived by using this restriction is called a k-answer set of the program. Formally, we call an interpretation of a program P a k-interpretation, iff $I(a) \in \mathbb{Q}_k$ for every proposition a. Consequently, a k-interpretation is a k-model of a program P iff it satisfies every rule of P. For a positive program P, a k-model of P is a k-answer set of P iff there is no k-model J of P such that $J < I$. For a non-positive program P, a k-model P is a k-answer set of P iff it is a k-answer set of P^I. Note that every answer set of a FASP program is necessarily a k-answer set of the program for some finite k, but the converse is generally not true: a k-answer set of a program may not be an answer set of that program [7], [36].

Example 3. *Consider the FASP program P_2 having the same rules as program P_1 from Example 2. This program has infinitely many answer sets I_x having $I_x(open) = x$ and $I_x(close) = 1-x$, where $x \in [0, 1]$. Furthermore, the program has exactly $k + 1$ k-answer sets for each positive integer k, where each answer set I_x is of the form $I_x(open) = x$ and $I_x(close) = 1 - x$, with $x \in \mathbb{Q}_k$.*

III. Related Work

Although Boolean networks provide a useful simplification to study the dynamics of gene regulatory networks, a deeper analysis requires the expression levels of the genes to take on more than two values (on or off) [17], [24]–[26], [29]. Indeed, using only Boolean levels of activation may cause one to miss many of the important attractors that are relevant for the understanding of the biological system. It is thus of importance to be able to extend the notion of Boolean networks to allow more levels of activation for each of the nodes. In [25], an extension of Boolean networks into multi-valued networks in which each node is allowed to have k levels of activation (where $k \geq 2$) is considered. Using the so-called 1-hot encoding, these multi-valued networks are reduced into a representation which allows techniques already used in Boolean networks, such as Binary Decision Diagrams (BDD), to be applied. However, the use of encoding scheme such as 1-hot encoding can make the representation quite cumbersome, especially for larger values of k, since it requires the explicit definition of the logical operators at each value of k. As we will show later, the use of FASP can overcome this problem by using the computation of k-answer sets, which is a feature built into FASP.

ASP has been successfully applied to model the dynamics of gene regulatory networks in the Boolean setting, see e.g. [11], [23]. In these works, the encoding of the update function is restricted to two specific types (denoted as r^* ans r^+ in [23]), due to the particular way that the encoding of the dynamics is written (i.e., encoding the update function at a *meta-level*). In this paper, we propose a new method to encode the dynamics of multi-valued networks using FASP which incorporates two things:

- allowing the graded activation levels in the nodes of the networks, and
- allowing a more flexible definition of the network update function. In [37], it was suggested that each of the node's update function of a Boolean network can be directly encoded as a rule in ASP. This allows for a more generic encoding of the network update function. Furthermore, it was shown that the steady states of the network are directly obtainable using the semantics of ASP. The obtain the cyclic attractors, however, [37] proposes an extension of the ASP semantics. Since such an extension is not practical for the purpose of this paper, we choose to use a more direct approach: by encoding the dynamics of the network using a *time* argument, as will be shown in the next section.

IV. Multi-valued Networks

We first formally define the concept of a multi-valued network as follows. A multi-valued network is a tuple $G = \langle X, F, k\rangle$ where X is a tuple of multi-valued variables denoting the nodes of the network, F is a tuple of update functions associated to each $x \in X$, and $k \geq 1$ is a parameter describing the number of activation levels for each node. Specifically, for each node $x \in X$, we allow $k + 1$ activation levels, i.e., the value for each x is taken from the set $\mathbb{Q}_k = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$. The state space of the network is thus the set of all $(k + 1)^n$ valuations for the nodes in X. Furthermore, the functions $f_i \in F$ are defined using the Łukasiewicz connectives $\otimes, \oplus, \veebar\!\!\wedge$, and $\neg$, instead of the Boolean connectives.

Note that Boolean networks correspond to the special case of multi-valued networks where $k = 1$. The network update function, the update rule, the state transition diagram and the attractors are then defined similarly as in Boolean networks.

Example 4. *Consider any network $G_2 = \langle\{a, b, c\}, F, k\rangle$, where the update functions F are given as follows*

$$a_{t+1} = b_t \oplus c_t$$
$$b_{t+1} = b_t \otimes a_t$$
$$c_{t+1} = c_t \otimes \neg b_t$$

It can be verified that this network "generalises" the Boolean network G_1 from Example 1, in the sense that all the attractors of the network from the Boolean case are also still attractors for every $k > 1$. Furthermore, we can check that there are additional attractors in the multi-valued setting that cannot be expressed using Boolean values. For example, when $k = 2$, we obtain that $\langle 0, 0, \frac{1}{2}\rangle$ and $\langle \frac{1}{2}, 0, \frac{1}{2}\rangle$ are also attractors of this network under the synchronous update rule, since in this case, $f(\langle 0, 0, \frac{1}{2}\rangle) = \langle 0, 0, \frac{1}{2}\rangle$ and $f(\langle \frac{1}{2}, 0, \frac{1}{2}\rangle) = \langle \frac{1}{2}, 0, \frac{1}{2}\rangle$.

A. Modeling multi-valued networks using FASP

In this section, we describe our FASP encoding to model the dynamics of multi-valued networks and to compute the steady-states and attractors of the networks.

1) Finding steady states: We first tackle the easy case of finding the single state attractors – also called steady states – of a multi-valued network. Recall that the steady states are identical for the synchronous and asynchronous update rules.

Let $G = \langle X, F, k\rangle$ be a multi-valued network. First, for every node $x \in X$ in the network, we consider two fuzzy propositional atoms p_x and n_x, and write the following FASP rules

$$p_x \oplus n_x \leftarrow$$
$$\overline{0} \leftarrow p_x \otimes n_x$$

Define *GU(G)* as the set of all such rules. Intuitively, the rules in *GU(G)* are used to generate the state space of the multi-valued network. Specifically, each p_x encodes the activation level of variable $x \in X$, while the atom n_x is needed in the encoding to generate all possible values for p_x.

We then encode the interaction between nodes by creating a rule for every node x, where the head of the rule is the propositional atom p'_x associated with the node, while the body corresponds to the direct translation of the fuzzy logic function for the update rule of x, replacing the occurrences of the negation symbol $\neg$ with FASP's default negation **not** . Formally, let f_i be the update function in a node x_i of the network. The corresponding FASP *node update rule* of that node, denoted by $NU(f_i)$ is a FASP rule defined as follows:

$$NU(f_i) \equiv p'_{x_i} \leftarrow BU(f_i)$$

where $BU(f_i)$ is a FASP expression defined recursively as follows:

- $BU(f_i) = val$ if $f_i(x_i) = val$ and $val \in [0, 1]$
- $BU(f_i) = p_x$ if $f_i(x_i) = x$ for a node x
- $BU(f_i) = BU(exp_1) \circ BU(exp_2)$ if $f_i(x_i) = exp_1 \circ exp_2$ for some expressions exp_1, exp_2 and $\circ \in \{\oplus, \otimes, \min, \max\}$
- $BU(f_i) =$ **not** p_x if $f_i(x_i) = \neg x$

Define $NU(G)$ as the set of rules created in this step (i.e., $NU(G) = \{NU(f_i) \mid f_i \in F\}$). Intuitively, the atom p'_{x_i} holds the activation value of the node x_i after the update function is applied. To ensure that we have a steady-state, we need to enforce the condition that the activation level of each node is the same after the update. This can be done by using the following rules $CS(i)$ for each node x_i

$$\overline{0} \leftarrow p_{x_i} \otimes \textbf{not } p'_{x_i}$$
$$\overline{0} \leftarrow p'_{x_i} \otimes \textbf{not } p_{x_i}$$

Define $CS(G)$ as the set of all rules combined from $CS(i)$ for all $i = 1, \ldots, |X|$. We have the following results.

Proposition 1. *The program $P(G) = GU(G) \cup NU(G) \cup CS(G)$ captures all the steady states of the multi-valued network G, i.e., for every k-answer set I of $P(G)$, the state S s.t. $S(x) = I(p_x)$ for every $x \in X$ is a steady state of G, and for every steady state S of G, there is a corresponding k-answer set I of G s.t. $S(x) = I(p_x)$ for every $x \in X$.*

Proof. First, it can be easily seen that in any answer set I of $P(G)$, we have that $I(p'_x) = I(p_x)$, due the rules in $CS(G)$. Suppose that S is a steady-state of the multi-valued network G. By definition, we have that $f_i(X) = S(x_i)$ for every $x_i \in X$. We will show that the interpretation I s.t. $I(p_x) = S(x)$ and $I(n_x) = 1 - S(x)$ for every $x \in X$ is a k-answer set of the program $P(G)$. First, by the definition of $GU(G)$, it is clear that I is a model of $GU(G)$. For every rule r in $NU(G)$ corresponding to the update function f_i, from the fact that $I(p_x) = I(p'_x) = S(x)$ for every $x \in X$, it can be shown that the recursive definition of $BU(f_i)$ entails that $I(Body(r)) = f_i(X)$. Since we have $f_i(X) = S(x_i)$, we also have that $I(Body(r)) = S(x_i)$. This means that $I(Head(r)) = I(p'_{x_i}) = S(x_i) = I(Body(r))$, which means that I is also a model of the rule r. Consequently, I is a model of $NU(G)$, and thus also of $P(G) = GU(G) \cup NU(G)$. It is easy to see that I is a minimal k-model of $GU(G)$, since any k-model $J < I$ will not satisfy at least one rule in $GU(G)$.

Conversely, if we have a k-answer set I of $P(G)$, we can show that the state S s.t. $S(x) = I(p_x)$ for every $x \in X$ is a steady state of the network. It is sufficient to show that $f_i(X) = S(x_i) = I(p_x) = I(p'_x)$ for every $x_i \in X$. Since I is a model of the rule $NU(f_i)$, we have that $I(p'_x) \geq I(Body(NU(f_i)) = I(BU(f_i))$. From the definition of $BU(f_i)$ it can be shown that $I(BU(f_i)) = f_i(X)$. Hence we have that $I(p'_{x_i}) \geq f_i(X)$. Suppose that $I(p'_{x_i}) > f_i(X)$, for some $x_i \in X$. Consider the k-interpretation J such that $J(p'_a) = I(p'_a)$ for every $a \in X$ s.t. $a \neq x_i$, and $J(p'_{x_i}) = f_i(X)$. We have that $J < I$, and it can be seen that J is also a k-model of $P(G)$ (since it satisfies all the rules in $P(G)$), contradicting the minimality of I. Hence, we must have that $I(p'_{x_i}) = f_i(X)$ for every $x_i \in X$. □

Example 5. *Consider the multi-valued network G_2 given in*

Example 4. In this case, $GU(G_2)$ is given as

$$\begin{aligned} p_a \oplus n_a \leftarrow & \\ \quad \overline{0} \leftarrow p_a \otimes n_a & \\ p_b \oplus n_b \leftarrow & \\ \quad \overline{0} \leftarrow p_b \otimes n_b & \\ p_c \oplus n_c \leftarrow & \\ \quad \overline{0} \leftarrow p_c \otimes n_c & \end{aligned}$$

while the node update rules $NU(G_2)$ are given as follows:

$$\begin{aligned} p_a &\leftarrow p_b \oplus p_c \\ p_b &\leftarrow p_b \otimes p_a \\ p_c &\leftarrow p_c \otimes \textbf{\textit{not}}\ p_b \end{aligned}$$

One can check that the k-answer sets of the program coincide with the steady states of the multi-valued network with $k+1$ activation levels. For example, for $k=2$, the answer set $\{(p_a,0),(p_b,0),(p_c,0),(n_a,1),(n_b,1),(n_c,1)\}$ corresponds to the attractor $\langle 0,0,0\rangle$, while the answer set $\{(p_a,\frac{1}{2}),(p_b,0),(p_c,\frac{1}{2}),(n_a,\frac{1}{2}),(n_b,1),(n_c,\frac{1}{2})\}$ corresponds to the attractor $\langle \frac{1}{2},0,\frac{1}{2}\rangle$.

2) Finding fixed-size cyclic attractors: For finding the attractors that consist of more than one state, it is clear that the previous approach will not work, since the proposed encoding does not represent different values of each node at different update times, which would be needed to check for cyclicity. Furthermore, it is clear that we will need to explicitly take into account the time dimension to be able to distinguish between the different update rules. This can be achieved by adding a parameter t, representing time, to each of the fuzzy propositional atom p_x and n_x. The initial guessing rules (we call them $GU_0(G)$) are now written as

$$\begin{aligned} p_x(0) \oplus n_x(0) \leftarrow & \\ \quad \overline{0} \leftarrow p_x(0) \otimes n_x(0) & \end{aligned}$$

where the parameter 0 encodes the fact that we are guessing at the initial time point $t=0$.

We then define a new encoding of the node update rule that incorporates a time parameter t. In particular, we consider the rule $TNU(f_i,t)$ defined as

$$p_{x_i}(t+1) \leftarrow up_i(t) \otimes TBU(f_i,t)$$

where $TBU(f_i,t)$ is defined as follows:

- $TBU(f_i,t) = val$ if $f_i(x_i) = val$ and $val \in \{0,1\}$
- $TBU(f_i,t) = p_x(t)$ if $f_i(x_i) = x$ for a node x
- $TBU(f_i,t) = TBU(exp_1,t) \circ TBU(exp_2,t)$ if $f_i(x_i) = exp_1 \circ exp_2$ for some expressions exp_1, exp_2 and $\circ \in \{\oplus, \otimes, \min, \max\}$
- $TBU(f_i,t) = \textbf{not}\ p_x(t)$ if $f_i(x_i) = \neg x$

The literal $up_i(t)$ intuitively denotes a *flag* that determines whether node x_i should be updated at time t. To select which update rule to be used, we define a set of rules, $SEL_u(s)$ (where u indicates the type of update rule used), as follows. For the synchronous update rule, $SEL_s(s)$ simply consists of the rules $up_i(t) \leftarrow 1$ for $t = 1,\ldots,s$. For the 1-synchronous update rule, $SEL_1(s)$ contains the following rules for each t:

$$\begin{aligned} up_1(t) \oplus up_2(t) \ldots \oplus up_n(t) \leftarrow 1 & \\ up_1(t) \leftarrow up_1(t) \oplus up_1(t) & \\ \vdots & \\ up_n(t) \leftarrow up_n(t) \oplus up_n(t) & \end{aligned}$$

where $n = |X|$.

For the *-asynchronous update rule, $SEL_*(s)$ consists of the following set of rules for each i,t

$$\begin{aligned} up_i(t) \oplus \neg up_i(t) \leftarrow 1 & \\ up_i(t) \leftarrow up_i(t) \oplus up_i(t) & \\ \overline{0} \leftarrow \textbf{not}\ up_i(t) \otimes \textbf{not}\ up_2(t) \otimes \ldots \otimes \textbf{not}\ up_n(t) & \end{aligned}$$

Lemma 1. *For any programs R not containing any rules having propositions of the form $up_i(t)$ in the head, we have that for any k-answer set I of the program $R \cup SEL_u(s)$:*

- *$I(up_i(t)) = 1$ for any i,t, if $u = s$. Intuitively, in such a case, all rules will be selected for updates at each time step.*
- *$I(up_i(t)) = 1$ for exactly one i for every t, if $u = 1$. Intuitively, in such a case, exactly one rule will be selected for updates at each time step.*
- *$I(up_i(t)) = 1$ for at least one i for every t, if $u = *$.*

Proof. For $u = s$, it is trivial that $I(up_i(t)) = 1$ for any i,t. It is easy to see that in any FASP program P, if there is a rule $a \leftarrow a \oplus a$, then in any (k-)answer set J of P, either $J(a) = 0$ or $J(a) = 1$. Thus, we can see that in any answer set I, either $I(up_i(t)) = 0$ or $I(up_i(t)) = 1$. For $u = 1$, the first rule in $SEL_s(G)$ forces that there should be at least one among $up_i(t)$ d.t. $I(up_i(t)) > 0$. For this, we should conclude that $I(up_i(t)) = 1$. For $u = *$, we can see that any number of i can satisfy $I(up_i(t)) > 0$. Due to the last rule of $SEL_*(s)$ however, for at least one i this should be true. □

To find a cyclic attractor of size s, consider a FASP program that contains the rules $TNU(f_i,t)$ for each i and $t = 0,1,\ldots,s$, as well as the following rules for each $x_i \in X$

$$\begin{aligned} 0 &\leftarrow p_{x_i}(s) \otimes \textbf{not}\ p_{x_i}(0) \\ 0 &\leftarrow p_{x_i}(0) \otimes \textbf{not}\ p_{x_i}(s) \end{aligned}$$

These constraints verify that after s steps, each node is back to its initial state. Call the set of all update rules generated for network G up to time step s as $TNU(G)$ (i.e., $TNU(G) = \{TNU(f_i,t) \mid x_i \in X, t = 0,1,\ldots,s\}$), and the set of rules to check the cyclic attractor of size s as $CC(G,s)$. We can show the following result.

Proposition 2. *The program $P(G,s) = GU_0(G) \cup TNU(G) \cup CC(G,s)$ captures the cyclic attractors of the network G whose size is exactly s, i.e., that any interpretation I of $P(G,s)$ is a k-answer set of $P(G,s)$ iff the states S_t, $t = 1,\ldots,s$ s.t. $S_t(x_i) = I(p_{x_i}(t))$ and $1 - S_t(x_i) = I(n_{x_i}(t))$ form a cyclic attractors, with $S_s = S_0$.*

Proof Sketch. With the help of Lemma 1, one can show by induction in t that in any k-answer set I of $P(G, s)$, $I(p_{x_i}(t)) = S_t(x_i)$ for every $x_i \in X$ for some states S_t reachable after t transition by the appropriate update rule (either 1-asynchronous or *-asynchronous). The rules in $CC(G, s)$ then rule out the models for which $I(p_{x_i}(s)) \neq I(p_{x_i}(0))$, i.e., when $S_s(x_i) \neq S_0(x_i)$ for some $x_i \in X$. This means that $\{S_t \mid t = 0, 1, \ldots, s-1\}$ is a cyclic attractor of size s. □

Example 6. *For the example network G_2 given in Example 4, the following is the set of time-respecting node update rules:*

$$a(t+1) \leftarrow up_1(t) \otimes b(t) \otimes c(t)$$
$$b(t+1) \leftarrow up_2(t) \otimes b(t) \otimes a(t)$$
$$c(t+1) \leftarrow up_3(t) \otimes c(t) \otimes \neg b(t)$$

To find attractors of size 3, for example, we need to instantiate that set of rules for $t = 0, 1, 2, 3$, and add the appropriate selection rules for $up_i(t)$.

V. Benchmark and Experiments

To validate our method, we applied it to known logic-based biological network models obtained from the literature. Table 1 represents the summary of the data collected. In the referred paper for each of these networks, the node update functions have been specified for a certain k (either $k = 2$ or $k = 3$). Furthermore, each node is modelled as either Boolean-valued, 3-valued or 4-valued, depending the on the available biological data supporting such models. In encoding the regulatory relationships between the nodes in the network, we assign values from $\mathbb{Q}_k$ to any $(k+1)$-valued nodes. Consequently, in these network models, we only consider attractors reached from the set of states where the Boolean-valued nodes are assigned either 0 or 1, and 3-valued and 4-valued nodes are assigned values from $\mathbb{Q}_2$ and $\mathbb{Q}_3$, respectively. To do this, we use the appropriate k value required (either 2 or 3), and then enforce Boolean interpretation for the Boolean nodes, by adding the saturation rule $p_x \leftarrow p_x \oplus p_x$ The steady-states of the network are then computed, and compared to the ones reported in their respective reference(s).

For the *A. thaliana* flowering network, the network update functions are listed in [17] as name-values pairs indicating the input-output pairs of the update function on each node. Such a representation can always be written into a Łukasiewicz function through the use of combinations of the operators $\veebar, \barwedge, \otimes$ and $\oplus$. For the Th cell regulatory network, [40] proposed different versions of the network. For our purpose, we use the logical rules presented in Equation 2 in that paper, and evaluate them as 3-valued Łukasiewicz functions (i.e., treating $\vee$ and $\wedge$ as $\oplus$ and $\otimes$, respectively), which is equivalent to the 1-hot encoding used in [16]. Nodes that do not have an explicit update function are given a "default" update function of $p'_x \leftarrow p_x$. For the *P. aeruginosa* mucus development network and the *D. melanogaster* segmentation network, the network update functions are represented using the notation used in [29]. By ignoring the time-delay parameter of this representation and assumpting the basal-expression levels of the genes to be 0 (as also done in [26]), we can faithfully represent each of the update functions given using Łukasiewicz logic.

The benchmark is conducted on an Apple MacBook Pro with 2.4GHz Intel i5 processor and 4GB of memory, running OS X Yosemite. We used the latest version of our solver ffasp version 0.8 [8] available at https://github.com/mushthofa/ffasp, and clingo v4.5.3 as the back-end ASP solver. From the result given in Table 1, we can see that the proposed method finds the steady-states in an efficient manner.

VI. Conclusion

Boolean networks are a popular modeling technique to analyze the dynamic behaviour of GRNs. Using Boolean networks, we can capture the steady states/attractors of the network, which are often useful to understand the biological function of such networks. Many tools, including approaches based on ASP, have been devised to model such dynamics. Multi-valued networks extend Boolean networks by allowing the representation of different levels of activation for the nodes. In this paper, we have proposed the use of FASP, an extension of ASP in the continuous domain, as the language for encoding the dynamics of multi-valued networks. We showed that any network model represented as multi-valued networks in any k can be faithfully encoded in FASP. Furthermore, our encoding in FASP reasonably captures the different assumptions usually required in the modeling of biological networks. To the best of our knowledge, this is the first real-world application of FASP that goes beyond small toy examples and synthetic FASP programs. We showed the correctness of our encoding, and we evaluated its efficiency in computing the steady-states of real biological networks found in the literature.

References

[1] C. Baral, *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[2] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 107–124, 2011.

[3] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, "The DLV system for knowledge representation and reasoning," *ACM Transactions on Computational Logic*, vol. 7, no. 3, pp. 499–562, 2006.

[4] T. Eiter, G. Ianni, and T. Krennwallner, "Answer set programming: A primer," in *Reasoning Web. Semantic Technologies for Information Systems*, ser. Lecture Notes in Computer Science, S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. Schmidt, Eds. Springer Berlin Heidelberg, 2009, vol. 5689, pp. 40–110.

[5] D. Van Nieuwenborgh, M. De Cock, and D. Vermeir, "Fuzzy answer set programming," in *Proceedings of the 10th European Conference on Logics in Artificial Intelligence*, 2006, pp. 359–372.

[6] M. Blondeel, S. Schockaert, D. Vermeir, and M. De Cock, "Complexity of fuzzy answer set programming under Łukasiewicz semantics," *International Journal of Approximate Reasoning*, vol. 55, no. 9, pp. 1971–2003, 2014.

[7] M. Mushthofa, S. Schockaert, and M. De Cock, "A finite-valued solver for disjunctive fuzzy answer set programs," in *Proceedings of European Conference in Artificial Intelligence 2014*, 2014, pp. 645–650.

[8] ——, "Solving disjunctive fuzzy answer set programs," in *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2015, pp. 453–466.

TABLE I
BENCHMARK RESULT

No.	A	B	C	D	E	F
1	*P. aeruginosa* mucus development network [38], [39]	2	1	3	2	0.3
2	*A. thaliana* flowering network [17]	15	7	3	10	5.6
3	Th cell regulatory network [16], [40]	23	9	3	4	8.1
4	*D. melanogaster* segmentation network [26]	7	4	4	4	4.2

A = name and reference for the network model
B = number of nodes
C = number of Boolean nodes
D = number of levels of activation ($k + 1$)
E = number of attractors
F = computation time (in seconds)

[9] M. Alviano and R. Peñaloza, "Fuzzy answer sets approximations," *Theory and Practice of Logic Programming*, vol. 13, no. 4-5, pp. 753–767, 2013.

[10] ——, "Fuzzy answer set computation via satisfiability modulo theories," *Theory and Practice of Logic Programming*, vol. 15, pp. 588–603, 7 2015. [Online]. Available: http://journals.cambridge.org/article_S1471068415000241

[11] T. Fayruzov, M. De Cock, C. Cornelis, and D. Vermeir, "Modeling protein interaction networks with answer set programming," in *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine, 2009*, 2009, pp. 99–104.

[12] M. Gebser, A. Konig, T. Schaub, S. Thiele, and P. Veber, "The BioASP library: ASP solutions for systems biology," in *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2010*, vol. 1. IEEE, 2010, pp. 383–389.

[13] S. Dworschak, S. Grell, V. J. Nikiforova, T. Schaub, and J. Selbig, "Modeling biological networks by action languages via answer set programming," *Constraints*, vol. 13, no. 1-2, pp. 21–65, 2008.

[14] S. A. Kauffman, *The origins of order: Self-organization and selection in evolution*. Oxford university press, 1993.

[15] H. De Jong and M. Page, "Search for steady states of piecewise-linear differential equation models of genetic regulatory networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 2, pp. 208–222, 2008.

[16] L. Mendoza, "A network model for the control of the differentiation process in Th cells," *Biosystems*, vol. 84, no. 2, pp. 101–114, 2006.

[17] C. Espinosa-Soto, P. Padilla-Longoria, and E. R. Alvarez-Buylla, "A gene regulatory network model for cell-fate determination during arabidopsis thaliana flower development that is robust and recovers experimental gene expression profiles," *The Plant Cell Online*, vol. 16, no. 11, pp. 2923–2939, 2004.

[18] A. Garg, I. Xenarios, L. Mendoza, and G. DeMicheli, "An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments," in *Research in Computational Molecular Biology*. Springer, 2007, pp. 62–76.

[19] G. Arellano, J. Argil, E. Azpeitia, M. Benitez, M. Carrillo, P. Gongora, D. Rosenblueth, and E. Alvarez-Buylla, ""antelope": a hybrid-logic model checker for branching-time boolean grn analysis," *BMC Bioinformatics*, vol. 12, no. 1, p. 490, 2011. [Online]. Available: http://www.biomedcentral.com/1471-2105/12/490

[20] F. Ay, F. Xu, and T. Kahveci, "Scalable steady state analysis of boolean biological regulatory networks," *PLoS ONE*, vol. 4, no. 12, p. e7992, 12 2009.

[21] E. Dubrova and M. Teslenko, "A SAT-based algorithm for finding attractors in synchronous boolean networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 8, no. 5, pp. 1393–1399, 2011.

[22] D. Zheng, G. Yang, X. Li, Z. Wang, F. Liu, and L. He, "An efficient algorithm for computing attractors of synchronous and asynchronous boolean networks," *PloS one*, vol. 8, no. 4, p. e60593, 2013.

[23] M. Mushthofa, G. Torres, Y. Van de Peer, K. Marchal, and M. De Cock, "ASP-G: an ASP-based method for finding attractors in genetic regulatory networks," *Bioinformatics*, p. btu481, 2014.

[24] G. Didier, E. Remy, and C. Chaouiya, "Mapping multivalued onto boolean dynamics," *Journal of Theoretical Biology*, vol. 270, no. 1, pp. 177 – 184, 2011.

[25] A. Garg, L. Mendoza, I. Xenarios, and G. DeMicheli, "Modeling of multiple valued gene regulatory networks," in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2007)*. IEEE, 2007, pp. 1398–1404.

[26] L. Sanchez and D. Thieffry, "Segmenting the fly embryo:: a logical analysis of the pair-rule cross-regulatory module," *Journal of Theoretical Biology*, vol. 224, no. 4, pp. 517–537, 2003.

[27] A. Bockmayr and H. Siebert, *Programming Logics: Essays in Memory of Harald Ganzinger*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ch. Bio-Logics: Logical Analysis of Bioregulatory Networks, pp. 19–34.

[28] R. Thomas, "Boolean formalization of genetic control circuits," *Journal of Theoretical Biology*, vol. 42, no. 3, pp. 563–585, 1973.

[29] ——, "Regulatory networks seen as asynchronous automata: a logical description," *Journal of Theoretical Biology*, vol. 153, no. 1, pp. 1–23, 1991.

[30] E. Dubrova, M. Teslenko, and A. Martinelli, "Kauffman networks: Analysis and applications," in *Proceedings of the 2005 IEEE/ACM International Conference on Computer-aided Design (ICCAD 2005)*, Washington, DC, USA, 2005, pp. 479–484. [Online]. Available: http://dl.acm.org/citation.cfm?id=1129601.1129670

[31] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, vol. 88, 1988, pp. 1070–1080.

[32] E. Erdem, "Theory and applications of answer set programming," Ph.D. dissertation, The University of Texas at Austin, 2002, the University of Texas at Austin.

[33] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "Planning under incomplete knowledge," in *Proceedings of the 1st International Conference in Computational Logic (CL 2000)*, 2000, pp. 807–821.

[34] S. Schockaert, J. Janssen, and D. Vermeir, "Fuzzy equilibrium logic: Declarative problem solving in continuous domains," *ACM Transactions on Computational Logic*, vol. 13, no. 4, pp. 33:1–33:39, 2012.

[35] T. Lukasiewicz and U. Straccia, "Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web," in *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems*, 2007, pp. 289–298.

[36] S. Aguzzoli and A. Ciabattoni, "Finiteness in infinite-valued Łukasiewicz logic," *Journal of Logic, Language and Information*, vol. 9, no. 1, pp. 5–29, 2000.

[37] K. Inoue, "Logic programming for boolean networks," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 2011, pp. 924–930.

[38] J. Guespin-Michel and M. Kaufman, "Positive feedback circuits and adaptive regulations in bacteria," *Acta Biotheoretica*, vol. 49, no. 4, pp. 207–218, 2001.

[39] S. Peres and J.-P. Comet, "Contribution of computational tree logic to biological regulatory networks: Example from pseudomonas aeruginosa," in *Proceedings of the First International Workshop on Computational Methods in Systems Biology (CMSB 2003)*, 2003, pp. 47–56.

[40] L. Mendoza and I. Xenarios, "A method for the generation of standardized qualitative dynamical systems of regulatory networks," *Theoretical Biology and Medical Modelling*, vol. 3, no. 1, p. 13, 2006.