Synthesis of Boolean Networks from Biological Dynamical Constraints using Answer-Set Programming

Stéphanie Chevalier, Christine Froidevaux, Loïc Paulevé, Andrei Zinovyev

Synthesis of Boolean Networks from Biological Dynamical Constraints using Answer-Set Programming

Stéphanie Chevalier
LRI, CNRS, U. Paris-Sud
U. Paris-Saclay, France
stephanie.chevalier@lri.fr

Christine Froidevaux
LRI, CNRS, U. Paris-Sud
U. Paris-Saclay, France
christine.froidevaux@lri.fr

Loïc Paulevé
LaBRI, CNRS, U. Bordeaux
Bordeaux INP, France
loic.pauleve@labri.fr

Andrei Zinovyev
Institut Curie, INSERM
U. PSL, Mines ParisTech, France
andrei.zinovyev@curie.fr

***Abstract*—Boolean networks model finite discrete dynamical systems with complex behaviours. The state of each component is determined by a Boolean function of the state of (a subset of) the components of the network.**

This paper addresses the synthesis of these Boolean functions from constraints on their domain and emerging dynamical properties of the resulting network. The dynamical properties relate to the existence and absence of trajectories between partially observed configurations, and to the stable behaviours (fixpoints and cyclic attractors). The synthesis is expressed as a Boolean satisfiability problem relying on Answer-Set Programming with a parametrized complexity, and leads to a complete non-redundant characterization of the set of solutions.

Considered constraints are particularly suited to address the synthesis of models of cellular differentiation processes, as illustrated on a case study. The scalability of the approach is demonstrated on random networks with scale-free structures up to 100 to 1,000 nodes depending on the type of constraints.

***Index Terms*—model synthesis, discrete dynamical systems, reachability, attractors, systems biology**

I. Introduction

The modelling of complex dynamical systems usually requires extensive knowledge on their functioning to be able to reproduce their observed behaviours. For most physical and biological systems, such a knowledge is out of reach. In systems biology, the vast majority of (if not all) models involved trial error approaches with arbitrary choices for specifying the rules of the model, until its dynamics fits with the desired behaviour.

The synthesis of dynamical models aims at providing an automatic way of designing models that satisfy constraints derived from knowledge on the structure and on the behaviour of the system, and potentially gives insight into the diversity of such models.

In this paper, we address the synthesis of Boolean Networks (BNs) from dynamical properties derived from partial and discrete-time observations of the system. BNs model the dynamics of a finite set of nodes having binary states. The possible evolution of these configurations are computed according to a collection of Boolean functions and an update semantics. BNs are close to 1-bounded Petri nets [1], and are extensively applied to model the complex dynamics of biological networks. We consider positive and negative reachability properties, i.e., the ability (or impossibility) for the model to evolve from one configuration to another; and long-run properties, i.e., on configurations that are eventually reached after an infinite amount of time. The domain of Boolean functions composing the candidate BNs is typically delimited by a given influence graph (often called *Prior Knowledge Network*), which specifies for each node the variables that can be used in its Boolean function.

These properties are motived by the modelling of cellular differentiation processes. Starting from a multi-potent (stem) state, cells progressively specialize into specific types. Various biological experimentation techniques measure the activities of certain genes during the differentiation processes (at different times). From these observations can then be derived positive reachability properties to reproduce the sequence of observed states; but also attractor properties when observations have been performed in stabilized cells. Finally, negative reachability properties enable to model bifurcations inherent in the differentiation process: once a cell enters a particular branch of differentiation, it is impossible for it to reach cell types related to the other branches.

In the literature, the synthesis of BNs subject to static and dynamical properties derived from partial and discrete-time observations essentially splits into either evolutionary optimization algorithms, or satisfiablility problems. Methods of the former category, such as [2], [3], couple genetic algorithms to explore the model space together with simulations to assess positive reachability and attractor properties. In practice, they allow addressing networks between 20-40 nodes. Such approaches do not guarantee terminating, nor finding a globally optimal model. Moreover, they offer a very limited access to the space of solutions of the synthesis problem. On the other hand, [4] uses Answer-Set Programming, and [5] Satisfiability Modulo Theory (SMT), to express the synthesis problem. Such approaches enable the exhaustive enumeration of all the solutions, potentially subject to optimization criteria.

The authors ackowledge the support from ITMO Cancer and from the French Agence Nationale pourla Recherche (ANR), in the context of ANR-FNR project AlgoReCell ANR-16-CE12-0034

$$f_1(x) := \neg x_2$$
$$f_2(x) := \neg x_1$$
$$f_3(x) := \neg x_1 \wedge x_2$$

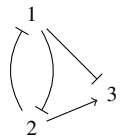

Fig. 1. Example of Boolean network f and its influence graph $G(f)$ where positive edges are with normal tip and negative edges are with bar tip.

In [4], only positive reachability properties are considered using model-checking, and have been applied to network up to 80 nodes. In [5], both positive reachability and fixpoint properties are supported, but only a particular subset of candidate Boolean functions are explored. Applications show scalability up to 20-40 nodes, with the synchronous semantics.

In this paper, we consider the logical synthesis of BNs from attractors, positive, and *negative* reachability properties using Answer-Set Programming (ASP), giving a complete characterization of the solutions. The considered dynamical constraints can be typically derived from the observation of cellular differentiation processes. We rely on the most permissive semantics of BNs, which offers both a correct abstraction of non-Boolean systems (as for biological systems), and a high scalability for the verification of dynamical properties.

II. Background

In this section, we formally define BNs, their influence (causal) graphs, and dynamical properties related to stability (trap spaces, attractors) and trajectories (reachability). Finally, we give a short introduction to Answer-Set Programming.

A. Boolean Networks

A *Boolean network* (BN) of dimension n is a function

$$f : \mathbb{B}^n \to \mathbb{B}^n \tag{1}$$

where $\mathbb{B} := \{0, 1\}$. For all $i \in \{1, \dots, n\}$, $f_i : \mathbb{B}^n \to \mathbb{B}$ denotes the *local function* of the i-th component. A vector $x \in \mathbb{B}^n$ is called a *configuration* of the BN f. The set of components which differ between two configurations $x, y \in \mathbb{B}^n$ is denoted by $\Delta(x, y) := \{i \in \{1, \dots, n\} \mid x_i \neq y_i\}$.

A BN f is said *locally monotonic* whenever each of its local functions is monotonic (this does not imply f monotonicity). Intuitively, when expressing the local functions using propositional logic, local monotonicity imposes that a variable appears always with the same sign in a minimal normal form.

Fig. 1 is an example of locally-monotonic BN with $n = 3$.

B. Influence Graph

For each component $i \in \{1, \dots, n\}$, f_i typically depends only on a subset of components of the BN. The *influence graph* (also called interaction or causal graph) summarizes these dependencies by having an edge from node j to i if f_i depends on the value of j. Formally, f_i depends on j if there exists a configuration $x \in \mathbb{B}^n$ such that $f_i(x)$ is different from $f_i(x')$ where x' is x having solely the component j different ($x'_j = \neg x_j$). Moreover, assuming $x_j = 0$ (therefore $x'_j = 1$), we say that j has a positive influence on i (in configuration x) if $f_i(x) < f_i(x')$, and a negative influence if $f_i(x) > f_i(x')$. It is possible that a node has different signs of influence on i in different configurations (leading to non-monotonic f_i). Remark that different BNs can have the same influence graph.

Definition 1. Given a BN f of dimension n, its *influence graph* $G(f)$ is a directed graph $(\{1, \dots, n\}, E_+, E_-)$ with *positive* and *negative* edges such that $(j, i) \in E_+$ (resp. $(j, i) \in E_-$) iff $\exists x, y \in \mathbb{B}^n$ s.t. $\Delta(x, y) = \{j\}$, $x_j < y_j$, and $f_i(x) < f_i(y)$ (resp. $f_i(x) > f_i(y)$).

Given two influence graphs $\mathcal{G} = (\{1, \dots, n\}, E_+, E_-)$ and $\mathcal{G}' = (\{1, \dots, n\}, E'_+, E'_-)$, we say that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$, denoted by $\mathcal{G} \subseteq \mathcal{G}'$ iff $E_+ \subseteq E'_+$ and $E_- \subseteq E'_-$.

Fig. 1 (right) shows the influence graph of the BN example.

C. Meta-configurations and trap spaces

The results presented in this paper extensively use the notion of meta-configurations, which denote hypercubes within $\mathbb{B}^n$, i.e., a set of components being fixed to a Boolean state, and the others being free (noted with $*$).

Definition 2. A *meta-configuration* h of dimension n is a vector in $(\mathbb{B} \cup \{*\})^n$. The set of its associated configurations is denoted by $c(h) := \{x \in \mathbb{B}^n \mid \forall i \in \{1, \dots, n\}, h_i \neq * \Rightarrow x_i = h_i\}$.

Given two meta-configurations $h, h' \in (\mathbb{B} \cup \{*\})^n$, h is *smaller* than h' iff $\forall i \in \{1, \dots, n\}, h'_i \neq * \Rightarrow h_i = h'_i$.

Trap spaces of a BN f are special cases of meta-configurations which are closed by f:

Definition 3. A *trap space* of a BN f of dimension n is a meta-configuration $t \in (\mathbb{B} \cup \{*\})^n$ such that $\forall x \in c(t), f(x) \in c(t)$.

A trap space is *minimal* if there is no smaller trap space. Remark that if $x \in \mathbb{B}^n$ is a fixpoint of f, i.e., $f(x) = x$, then x is a (minimal) trap space (hypercube of dimension 0).

Finally, given a BN f of dimension n and a set of components $L \subseteq \{1, \dots, n\}$, a L-constrained trap space is defined similarly, except that the closure is ensured only for the components *not* in L:

Definition 4. A *L-constrained trap space* of a BN f of dimension n with $L \subseteq \{1, \dots, n\}$ is a meta-configuration $w \in (\mathbb{B} \cup \{*\})^n$ such that $\forall i \in \{1, \dots, n\} \setminus L$, either $w_i = *$, or $\forall x \in c(w)$, $w_i = f_i(x)$.

Remark that $\emptyset$-constrained trap spaces are equivalent to trap spaces.

In the following, we will often rely on *smallest* (constrained) trap spaces *containing* a configuration x. These smallest meta-configurations can be obtained by transfinite iterations of functions $(\mathbb{B} \cup \{*\})^n \to (\mathbb{B} \cup \{*\})^n$ enlarging meta-configurations to satisfy the trap conditions, initially applied to x. For instance, the smallest L-constrained trap space containing x can be obtained by the transfinite iteration of e initially applied to x, where $e(h) = h'$ verifies $\forall i \in \{1, \dots, n\}, h'_i = *$ if $i \notin L$ and $\exists x \in c(h) : f_i(x) \neq x_i$, otherwise $h'_i = h_i$.

Example. The meta-configuration $01*$ is a trap space of the BN f of Fig. 1; $c(01*) = \{010, 011\}$. The meta-configuration $1*0$ is a $\{1\}$-constrained trap space of f, it is the smallest $\{1\}$-constrained trap space containing 110, and it is not a trap space, nor the smallest $\{1\}$-constrained containing 100.

D. Reachability

Given two configurations $x, y \in \mathbb{B}^n$, y is *reachable* from x, noted $x \to^* y$, if there exists a possible evolution of the configuration x, according to the BN f, which leads to y.

Numerous semantics of BNs have been defined in the literature [6]–[8], the most prominent being the *synchronous update mode*, where $\to^*$ is the transitive closure of the binary relation $\to_s \subseteq \mathbb{B}^n \times \mathbb{B}^n$ with $x \to_s y$ iff $f(x) = y$, i.e., all components get updated simultaneously in one step; and the *asynchronous update mode*, where $\to^*$ is the transitive closure of the binary relation $\to_a \subseteq \mathbb{B}^n \times \mathbb{B}^n$ with $x \to_a y$ iff $\forall i \in \Delta(x, y), y_i = f_i(x)$, i.e., any number of components gets updated (non-deterministically) in one step.

However, all the update modes of BNs are inconsistent abstractions of non-Boolean systems dynamics [9], i.e., they both introduce spurious reachability properties and miss reachability properties actually verified in more concrete quantitative specifications. This constitutes a prime issue for BN synthesis as it may lead to reject valid models.

The *most permissive* semantics of BNs has been recently introduced to address this issue [1], [10]. This semantics is currently the only one known which guarantees that its reachability properties are a correct over-approximation of reachability properties in any quantitative refinement of the BN, with any update mode.

In this paper, we focus on most permissive BNs. The reachability property $x \to^* y$ can then be characterized with the smallest constrained trap spaces containing x: y has to be contained in one of such meta-configurations w, and in the case a component i is free ($w_i = *$) whereas $x_i = y_i$, then there should exist a configuration $z \in c(w)$ such that $f_i(z) = y_i$. The most permissive reachability is formally defined as follows.

Definition 5. Given a BN f of dimension n and two configurations $x, y \in \mathbb{B}^n$, $x \to^* y$ if and only if there exists $L \subseteq \{1, \ldots, n\}$ such that the smallest L-constrained trap space w containing x verifies (1) $y \in c(w)$, and (2) $\forall i \in \{1, \ldots, n\} \setminus L$ where $x_i = y_i$ and $w_i = *$, $\exists z \in c(w)$ s.t $f_i(z) = y_i$.

Deciding $x \to^* y$ in locally-monotonic BNs of dimension n is in PTIME – NP-complete for general BNs – instead of PSPACE-complete with classical update modes [1], [10].

Example. In the BN f of Fig. 1, $000 \to^* 111$, $110 \to^* 000 \to^* 110$ ($L = \emptyset$), but $010 \not\to^* 100$ ($w = 01*$ with $L = \emptyset$). In the BN $g : \mathbb{B}^3 \to \mathbb{B}^3$ with $g_1(x) := 1$, $g_2(x) := x_1 \wedge x_3$ and $g_3(x) := \neg x_2$, $011 \to^* 000$ ($L = \{1\}, w = 0**$), but $001 \not\to^* 010$ (either $1 \notin L$, then $\nexists z \in c(w) : f_1(z) = 0$, or $1 \in L$, then $w = 001$).

E. Attractors

The long-run behaviour of BNs is characterized by so-called *attractors*, which are the smallest sets of configurations closed by the reachability relation:

Definition 6. An *attractor* of a BN f of dimension n is a set of configurations $A \subseteq \mathbb{B}^n$ such that $\forall x, y \in A, x \to^* y$ and $y \to^* x$, and $\forall x \in A, z \in \mathbb{B}^n$, $x \to^* z \Rightarrow z \in A$.

The set of attractors of f is denoted by $\mathcal{A}(f)$.

We usually distinguish two kinds of attractors: the singleton attractors $\{x\}$ corresponding to the fixpoints of the BN ($f(x) = x$); and the cyclic attractors.

With the most permissive semantics, attractors match exactly with the *minimal* trap spaces of f [10].

Example. The BN f of Fig. 1 has two attractors, being, in this particular case, fixpoints: 011 and 100. The BN g illustrating Def. 5 has a single cyclic attractor, being all the configurations $\{100, 101, 110, 111\}$, i.e., the minimal trap space $1**$.

It is worth noticing that, due to the non-determinism of BN semantics, one configuration can reach several attractors; it is the case in the BN f of Fig. 1, where the configuration 000 can reach the two fixpoints. This is an important feature of BNs for the modelling of biological differentiation processes.

F. Answer-Set Programming

Answer Set Programming (ASP; [11], [12]) is a declarative approach to solving combinatorial satisfaction problems. It is close to SAT (propositional satisfiability) [13] and known to be efficient for enumerating solutions of NP problems comprising up to tens of millions of variables, while providing a convenient language for specifying the problem. We give a very brief overview of ASP syntax and semantics that we use in the next sections; see [12] for more details.

An ASP program is a Logic Program (LP) being a set of logical rules with first order logic predicates of the form:

```
1 a_0 ← a_1, ..., a_n, not a_{n+1}, ..., not a_{n+k}.
```

where a_i are (variable-free) atoms, i.e., elements of the Herbrand base, which is built from all the possible predicates of the LP. The Herbrand base is built by instantiating the LP predicates with the LP terms (constants or elements of the Herbrand universe).

Essentially, such a logical rule states that when all $a_1, \ldots, a_n$ are true and none of $a_{n+1}, \ldots, a_{n+k}$ can be proven to be true, then a_0 has to be true as well. Whenever a_0 is $\bot$ (false), the rule, also called integrity constraint, becomes:

```
2 ←a_1, ..., a_n, not a_{n+1}, ..., not a_{n+k}.
```

Such a rule is satisfied only if the right hand side of the rule is false (at least one of $a_1, \ldots, a_n$ is false or at least one of $a_{n+1}, \ldots, a_{n+k}$ is true). On the other hand, $a_0 \leftarrow \top$ (a_0 is always true) is abbreviated as a_0. A solution (answer set) is a *stable* Herbrand model, that is, a minimal set of true atoms where all the logical rules are satisfied.

ASP allows using variables (starting with an upper-case) instead of terms/predicates: these *template* declarations will be

expanded to the corresponding propositional logic rules prior to the solving. For instance, the following ASP program

```
c(X) ← b(X).
b(1).
b(2).
```

has as unique solution {`b(1), b(2), c(1), c(2)`}.

We also use the notations `a((x;y))` which is expanded to `a(x), a(y)`; `#count {X: a(X)}` which is the number of distinct `X` for which `a(X)` is true; n `{a(X): b(X)}` m which is satisfied when at least n and at most m `a(X)` are true where `X` ranges over the true `b(X)`; and `a(X): b(X)` which is satisfied when for each `b(X)` true, `a(X)` is true. If any term follows such a condition, it is separated with `;`. Finally, rules of form

```
{a} ← body.
```

leave the choice to make `a` true whenever the body is satisfied.

III. Synthesis Problem

This paper focuses on the synthesis of BNs from constraints on its influence graph and on its dynamics, with reachability and attractors properties.

The nature of the constraints is inspired by the modelling of cellular differentiation processes. In this biological context, a cell population evolves towards various phenotypes, and this behaviour covers interesting properties both in healthy and pathological context (respectively for studying embryogenesis and cancer for instance). Typical experimental data provide partial discrete-time observations of genes and proteins activity along bifurcating trajectories. These data can be further statistically processed to provide binary interpretation of the activity of components at the collected time points and classify them along differentiation branches. Then, putative components and influences of interest can be extracted from databases and completed by causal learning from the experimental data.

A (partial) observation o of a configuration of dimension n is specified by a set of couples associating a component to a Boolean value: $o \subseteq \{1,\dots,n\} \times \mathbb{B}$, assuming there is no $i \in \{1,\dots,n\}$ such that $\{(i,0),(i,1)\} \subseteq o$.

Formally, the synthesis problem we tackle is the following. Given

- an influence graph $\mathcal{G} = (\{1,\dots,n\}, E_+, E_-)$,
- p partial observations $o^1,\dots,o^p$,
- sets PR and NR of couples of indices of observations: $\mathsf{PR},\mathsf{NR} \subseteq \{1,\dots,p\}^2$,
- subset FP of indices of observations: $\mathsf{FP} \subseteq \{1,\dots,p\}$,
- a set TP associating indices of observations with components: $\mathsf{TP} \subseteq \{1,\dots,p\} \times \{1,\dots,n\}$,

find a BN f of dimension n such that

- $G(f) \subseteq \mathcal{G}$,
- there exist p configurations $x^1,\dots,x^p$ such that:
  - (observations) $\forall m \in \{1,\dots,p\}, \forall (i,v) \in o^m, x^m_i = v$,
  - (positive reachability) $\forall (m,m') \in \mathsf{PR}, x^m \to^* x^{m'}$,
  - (negative reachability) $\forall (m,m') \in \mathsf{NR}, x^m \not\to^* x^{m'}$,
  - (fixpoints) $\forall m \in \mathsf{FP}, f(x^m) = x^m$,
  - (trap space) $\forall (m,i) \in \mathsf{TP}, \exists t \in (\mathbb{B} \cup \{*\})^n : t$ is the smallest trap space containing x^m, and $t_i = x^m_i$.

Remark that such a problem can be non-satisfiable depending on the input influence graph and dynamical properties. Besides the scalability challenge of such a synthesis problem, desired features include the *complete* and *non-redundant* characterization of the satisfying BNs. Completeness is possible as there is a finite number of BNs f such that $G(f) \subseteq \mathcal{G}$. Non-redundancy implies that the method should enumerate only among non-equivalent BNs (i.e., where their values differ for at least one configuration).

IV. Answer-Set Programming Encoding

This section details the ASP encoding of the BN synthesis from constraints on its influence graph and its dynamics.

The constraints on dynamics relate to the *existence* of configurations which match their *partial* observations and verify given reachability and stability properties. A partial observation of configuration `X` is specified by `obs(X,N,V)` predicates, where `N` and `V` denote the component and its observed Boolean value. Boolean values are encoded as -1 for false, and 1 for true. The configuration `X` is encoded by a set of predicates `cfg(X,N,V)`. If the node `N` has been observed, `V` is equal to the observed value; otherwise, its value is chosen:

```
cfg(X,N,V) ← obs(X,N,V).
1 {cfg(X,N,(-1;1))} 1 ← obs(X,_,_), node(N),
    not obs(X,N,_).
```

A. Canonical Domain of Boolean Networks

The ASP encoding of locally-monotonic BNs compatible with an influence graph faces two difficulties. First, two different solutions should correspond to two non-equivalent BNs f and f', i.e., there exists $x \in \mathbb{B}^n$ such that $f(x) \neq f'(x)$. This requires ensuring that solutions match with canonical representations of BNs. Second, the worst size of the specification of a Boolean function is exponential in the number of its variables. Therefore, the encoding should allow specifying a bound on the size of the Boolean function specification, ideally without bounding the number of variables.

We represent the Boolean functions composing a BN under their Disjunctive Normal Form (DNF), i.e., a set of clauses, where clauses are sets of literals, and two distinct clauses have no subset relation (antichain). In ASP, we have to encode DNF as lists of clauses, and therefore give an index to each clause. The canonicity is then ensured by enforcing a total ordering between the clauses. The maximum number of clauses for a DNF with d variables is $\binom{d}{\lfloor d/2 \rfloor}$, and our encoding allows specifying a lower number to restrict the set of DNFs to consider, without limiting the number of variables to consider.

Overall, our encoding of canonical Boolean functions with d variables generates $O(ndk^2)$ predicates and $O(nd^2k^2)$ rules where k is the fixed upper bound on the number of DNF clauses per local function, the maximum being $\binom{d}{\lfloor d/2 \rfloor}$. With

this maximum value, the number of solutions matches with the number of distinct monotonic Boolean functions, the Dedekind number [14], currently known up to $d = 8$ [15][1]. Whenever the specified k is lower than the maximum, Boolean functions are not captured by the encoding. The constraints on canonicity are necessary to obtain efficient enumeration of solutions. Whenever checking only for the existence of at least one solution, these constraints can be relaxed, reducing the number of predicates and rules to $O(ndk)$.

We detail the encoding hereby. We use a predicate template `clause(N,C,L,S)` to specify that the literal `L` with sign `S` is included in the `C`-th clause of the DNF of f_{N}. For instance, the two-clauses DNF $f_a(x) = (\neg x_a \wedge x_b) \vee x_c$ is encoded by the three following predicates: `clause(a,1,a,-1)`, `clause(a,1,b,1)` and `clause(a,2,c,1)`.

The domain of arguments `N`, `L`, and `S` is fully determined by the input influence graph (V, E_+, E_-); `C` ranges from 1 to k. The influence graph is encoded with `node/1` predicates with `node(`i`)` if and only if $i \in V$, and `in/3` predicates such that `in(`j`,`i`,1)` if and only if $(j,i) \in E_+$ and `in(`j`,`i`,-1)` if and only if $(j,i) \in E_-$. The bound on the number of clauses is set by `maxC(N,`k`)`:

```
3 {clause(N,1..C,L,S): in(L,N,S), maxC(N,C)}.
```

The local monotonicity is ensured by denying a literal appearing with both signs in the DNF of each component `N`:

```
4 ← clause(N,_,L,S), clause(N,_,L,-S).
```

DNFs without clauses result in constant functions, specified with the predicate `constant/2`:

```
5 1 {constant(N,(-1;1))} 1 ← node(N),
    not clause(N,_,_,_).
```

The canonicity is obtained by ensuring the clauses are ordered by size and then lexicographically, and without subset relation. The ordering by size is guaranteed by the following integrity constraints. The first line ensures that clauses identifiers increase continuously from 1.

```
6 ← clause(N,C,_,_), not clause(N,C-1,_,_),
    C > 1.
7 size(N,C,X) ← clause(N,C,_,_),
    X = #count{L,S: clause(N,C,L,S)}.
8 ← size(N,C1,X1), size(N,C2,X2), X1 < X2,
    C1 > C2.
```

The lexicographic ordering between clauses of the same size is enforced as follows, where `clausediff(N,C1,C2,L)` indicates that `L` is present in the `C1`-th clause but not in the `C2`-th; and `mindiff(N,C1,C2,L)` indicates that `L` is the smallest literal such that `clausediff(N,C1,C2,L)`.

```
9 ← size(N,C1,X), size(N,C2,X), C1 > C2,
    mindiff(N,C1,C2,L1), mindiff(N,C2,C1,L2),
    L1 < L2.
10 clausediff(N,C1,C2,L) ← clause(N,C1,L,_),
    not clause(N,C2,L,_), clause(N,C2,_,_).
11 mindiff(N,C1,C2,L) ← clausediff(N,C1,C2,L),
    L <= L' : clausediff(N,C1,C2,L');
    clause(N,C1,L',_).
```

Finally, the absence of subset relation is guaranteed by the following integrity constraint:

```
12 ← size(N,C1,X1), size(N,C2,X2), X1 <= X2,
    clause(N,C2,L,S): clause(N,C1,L,S);
    C1 != C2.
```

B. Evaluation of Boolean functions

We define generic rules to evaluate Boolean functions on meta-configurations. A meta-configuration is specified similarly to configurations, with predictates `mcfg(H,N,V)`, where `V` in $\{-1, 1\}$, but with potentially two predicates `mcfg(`h`,`i`,-1)` `mcfg(`h`,`i`,1)` indicating that the component i is free in the meta-configuration h, i.e., $h_i = *$. The encoding of dynamical constraints takes care about instantiating their related `mcfg/3`.

The rules ensure that `eval(`h`,`i`,1)` (resp. `eval(`h`,`i`,-1)`) if and only if there exists a configuration $x \in c(h)$ such that $f_i(x)$ is true (resp. false). A clause is evaluated to false whenever one of its literal evaluates to false (l.13); and to true whevener all its literals evaluate to true (l.14). Then, either the function is a constant and its evaluation follows the constant value (l.17), or the function is evaluated to true if all its clauses have been evaluated true (l.15); and to false whenever one ot its clauses is evaluated false (l.16).

```
13 eval(H,N,C,-1) ← clause(N,C,L,-V),
    mcfg(H,L,V).
14 eval(H,N,C,1) ← clause(N,C,_,_),
    mcfg(H,_,_), mcfg(H,L,V): clause(N,C,L,V).
15 eval(H,N,1) ← eval(H,N,C,1); clause(N,C,_,_).
16 eval(H,N,-1) ← clause(N,_,_,_), mcfg(H,_,_),
    eval(H,N,C,-1): clause(N,C,_,_).
17 eval(H,N,V) ← constant(N,V), mcfg(H,_,_).
```

For each meta-configuration, this encoding generates $O(nk)$ predicates and $O(ndk)$ rules.

C. Positive Reachability

Each $(m, m') \in \mathsf{PR}$ is translated as a predicate `reach(`m`,`m'`)`, specifying that the configuration x^m has to be able to reach the configuration $x^{m'}$.

Following Def. 5, reachability properties in most permissive BNs can be assessed with particular meta-configurations. The rule below declares a meta-configuration dedicated to the positive reachability constraint, initially being equal to the initial configuration.

```
18 mcfg((pr,X,Y),N,V) ← reach(X,Y), cfg(X,N,V).
```

Then, the meta-configuration has to be extended to satisfy the (constrained) trap space property (Def. 4). The extensions of meta-configurations are encoded with `ext(H,N,V)` predicates, and their application is encoded by the generic rule in l.19. Whenever the function of the component `N` of the meta-configuration can be evaluated to its value in the target configuration `Y`, the meta-configuration is extended to include

[1]for $0 \leq d \leq 8$: 2, 3, 6, 20, 168, 7581, 7828354, 2414682040998, 56130437228687557907788

this value (l.20). Whenever the function can be evaluated to the opposite value of the target configuration, its inclusion in the meta-configuration is a choice (l.21).

```
19 mcfg(H,N,V) ← ext(H,N,V).
20 ext((pr,X,Y),N,V) ← reach(X,Y),
     eval((pr,X,Y),N,V), cfg(Y,N,V).
21 {ext((pr,X,Y),N,V)} ← reach(X,Y),
     eval((pr,X,Y),N,V), cfg(Y,N,-V).
```

The resulting meta-configuration is a L-constrained trap space, where L is the set of components where the extensions of l.21 have been skipped, provided the opposite value is not already in the initial configuration.

Finally, the two properties that the constrained trap space has to verify (Def. 5) lead to the following rules. The first rejects models where the target configuration is not included in the meta-configuration; the second rejects models where a component is free in the meta-configuration (therefore not in L), but its target value can not be obtained with its function in the scope of the constrained trap space.

```
22 ← cfg(Y,N,V), not mcfg((pr,X,Y),N,V),
     reach(X,Y).
23 ← cfg(Y,N,V), not ext((pr,X,Y),N,V),
     ext((pr,X,Y),N,-V), reach(X,Y).
```

Accounting for `eval`-related rules, for each `reach(X,Y)` predicate, $O(nk)$ predicates and $O(ndk)$ rules are generated.

D. Negative Reachability

Each $(m, m') \in \mathsf{NR}$ is translated as a predicate `nonreach(`m,m'`)`, specifying that it is impossible to reach the configuration $x^{m'}$ from configuration x^m.

The most permissive reachability property recalled in Def. 5 relies on the *existence* of subset of components $L \subseteq \{1, \ldots, n\}$ so that the smallest L-constrained trap space w containing the initial configuration (1) contains the target configurations, and (2) for each component i not in L, there exists a configuration $z \in c(w)$ such that $f_i(z) = y_i$.

Proving the absence of reachability would require that these conditions are verified by none of these subsets of components L. In [10], it has been demonstrated that its is sufficient to consider at most n particular subsets of components L to conclude on the absence of reachability. Essentially, we start verifying the conditions with $L = \emptyset$ and then iteratively add in L the components which do not satisfy the condition (2). With this procedure, it is sufficient to check the condition (1) in the L obtained at the n^{th} iteration.

To assess the non-reachability of configuration y from x, our encoding generates n meta-configurations, initially being equal to x (l.24-25). Then predicates `locked(X,Y,I+1,N)` specify that the component `N` is in the `I`+1$^{\text{th}}$ iteration of L. Such a predicate has to be true if `N` does not verify condition (2) at iteration `I` (l.26), or if it is already in L at the preceding iteration (l.27). The extension of the meta-configuration at iteration `I` is then constrained by components in L (l.28). Finally, if there exists a component `N` such that $y_{\texttt{N}}$ is not the meta-configuration of the last iteration, the predicate `nr(`x,y`)` is true, indicating the absence of reachability (l.29). A model is rejected if such a predicate cannot be proven true (l.30).

```
24 iter(1..K) ← nbnode(K).
25 mcfg((nr,X,Y,I),N,V) ← nonreach(X,Y),
     cfg(X,N,V), iter(I).
26 locked(X,Y,I+1,N) ← cfg(X,N,V), cfg(Y,N,V),
     not ext((nr,X,Y,I),N,V),
     ext((nr,X,Y,I),N,-V), iter(I+1).
27 locked(X,Y,I+1,N) ← locked(X,Y,I,N),
     iter(I+1).
28 ext((nr,X,Y,I),N,V) ← not locked(X,Y,I,N),
     eval((nr,X,Y,I),N,V).
29 nr(X,Y) ← not mcfg((nr,X,Y,K),N,V),
     nbnode(K), cfg(Y,N,V), nonreach(X,Y).
30 ← not nr(X,Y), nonreach(X,Y).
```

Accounting for `eval`-related rules, for each `nonreach(X,Y)` predicate, this encoding generates $O(n^2k)$ predicates and $O(n^2dk)$ rules.

E. Attractors

As indicated in Sect. III, we consider two different properties related to the attractors of the BN f: fixpoints properties, where specified configurations have to be fixpoints of f; and trap space properties, where specified configurations have to belong to trap spaces where a subset of their components have a fixed value.

Accounting for `eval`-related rules, the encoding of each of the following properties generates $O(nk)$ predicates and $O(ndk)$ rules.

1) Fixpoints: Each $m \in \mathsf{FP}$ is translated as a predicate `is_fp(`m`)`, specifying that the configuration x^m is a fixpoint of f . The constraint is ensured by rejecting models where the evaluation gives an opposite value for at least one component:

```
31 mcfg(X,N,V) ← is_fp(X), cfg(X,N,V).
32 ← is_fp(X), cfg(X,N,V), eval(X,N,-V).
```

2) Trap spaces: Each $(m, i) \in \mathsf{TP}$ is translated as a predicate `is_tp(`m,i`)`, specifying that the smallest trap space t containing the configuration x^m has to have the component i fixed, i.e., $t_i \neq *$. The initialisation and extension of the smallest trap space containing x are obtained with rules in l.33-34. The model is rejected if the resulting trap space has any free component specified as trapped.

```
33 mcfg((ts,X),N,V) ← cfg(X,N,V), is_tp(X,_).
34 mcfg((ts,X),N,V) ← eval((ts,X),N,V).
35 ← is_tp(X,N), cfg(X,N,V), mcfg((ts,X),N,-V).
```

V. Evaluation

We performed experiments to assess the scalability and illustrate potential biological applications of our encoding of BN synthesis. We used the ASP solver Clingo[2] using default solving strategies[3].

[2]version 5.3.0 available at https://potassco.org/clingo

[3]Instances available at http://www.labri.fr/perso/lpauleve/ictai19.zip

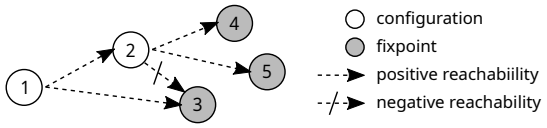

Fig. 2. Sketch of the constraints for the synthesis on random graphs

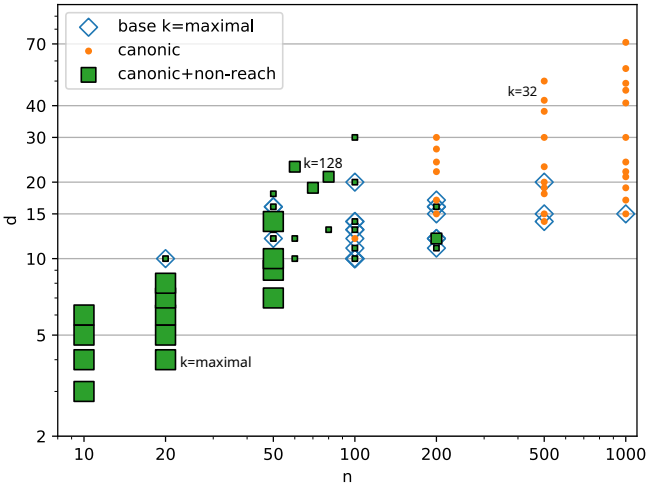

Fig. 3. Successfully solved random instances for different sets of constraints (marker shapes) and different bounds on the number of clauses (marker sizes) in function of the number of components n and maximal in-degree d.

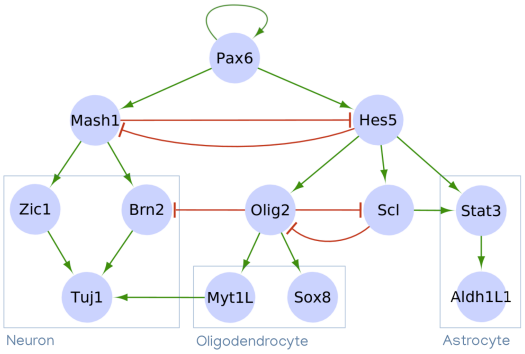

Fig. 4. Influence graph for CNS development

TABLE I
LIST OF OBSERVED NODES IN EACH OBSERVATION

obs. ID	activated genes	inactivated genes
0	*none*	*all*
iPax6	Pax6	*the 11 others*
tM	Pax6	Aldh1L1, Olig2, Scl, Sox8, Tuj1
fT	Brn2, Tuj1, Zic1	Aldh1L1, Sox8
tO	Olig2, Pax6	Aldh1L1, Scl, Sox8, Tuj1
fMS	Sox8	Aldh1L1, Brn2, Tuj1, Zic1
tS	Pax6, Scl	Aldh1L1, Olig2, Sox8, Tuj1
fA	Aldh1L1	Brn2, Sox8, Tuj1, Zic1

A. Scalability on Random Boolean Networks

We randomly generated scale-free directed graphs with different biases on the in-degree of nodes in order to obtain influence graphs similar to the usually encountered with gene and cell signalling networks.

The synthesis has then been performed with each of these networks as input influence graph, and with a generic dynamical property of a two stages differentiation processes, as illustrated in Fig. 2. The properties are specified using 5 *empty* observations $\{1,\ldots,5\}$, among which 3 should match with a distinct fixpoint ($\mathsf{FP} = \{3,4,5\}$). The first observation is supposed to reach the second and third, whereas the second is expected to reach the fourth and fifth, but not the third: $\mathsf{PR} = \{(1,2),(1,3),(2,4),(2,5)\}$, $\mathsf{NR} = \{(2,3)\}$.

Fig. 3 gives an overview of successfully solved instances within 2h of CPU time (2.5Ghz). With canonic solutions and the maximal number of clauses, it scales to networks up to 50 nodes, with maximal in-degree 15. With bounded number of clauses, instances with up to 200 nodes have been solved, provided a similar in-degree. Solving larger instances requires dropping negative reachability constraints. The main limit is the number of variables and rules generated by the encoding, which is often larger than 2^{32} with negative reachability. Almost all solved instances are satisfiable, except in a couple of cases with ≤ 20 nodes with negative reachability.

B. Application to Cell Differentiation Modelling

We illustrate our methodology on a cell differentiation context: the central nervous system (CNS) development. Neural stem cells can terminally differentiate into neurons, astrocytes and oligodendrocytes, and an influence graph gathering known gene interactions is available in the literature [16]. This graph with two differentiation stages (Fig. 4) consists of 12 genes. Despite its relatively small size, this influence graph already entails more than 226 millions of compatible BNs (the number of BNs compatible with an influence graph is given by the product of the Dedekind numbers related to each node).

The observations are given in Table I, and the positive and negative reachability constraints are set as $\mathsf{PR} = \{(\mathit{iPax6},\mathit{tM}),(\mathit{tM},\mathit{fT}),(\mathit{iPax6},\mathit{tO}),(\mathit{tO},\mathit{fMS}),(\mathit{iPax6},\mathit{tS}),(\mathit{tS},\mathit{fA})\}$, $\mathsf{NR} = \{(\mathit{0},\mathit{fT}),(\mathit{0},\mathit{fMS}),(\mathit{0},\mathit{fA})\}$.

To test the impact of various hypotheses on the stability of the phenotypes, trap spaces (with the fixation of the 4 phenotypes markers Aldh1L1, Myt1L, Sox8 and Tuj1) and fixpoints constraints are applied on the observations *fT*, *fMS* and *fA*. Their relevance depends on the assumptions and knowledge precision about the phenotypes.

To appreciate the pertinence of the method, Table II presents the number of inferred BNs given each defined constraint and combinations thereof. Each constraint complements the filtering by adding new information, and while 226 millions of BNs were candidates for modelling the CNS development, applying a relevant combination of constraints leads to select almost instantaneously the relatively small set of models respecting the observed behaviours. This huge reduction combined with the exhaustiveness of the method is twice interesting for biological studies. It first enables the analysis of variability across the models to study the significance of the components in the observed behaviours. Secondly, it offers the opportunity

TABLE II
NUMBER OF ADMISSIBLE BNS W.R.T. VARIOUS PROPERTIES

applied constraints	**# solutions**
application of a single type of constraint:	
3 negative reachability *(NR)*	224 025 280
6 positive reachability *(PR)*	24 076 416
12 trap spaces *(TP)*	17 220
3 fixpoints *(FP)*	4970
application of combination of constraints:	
PR + NR	16 050 944
PR + TP	8964
NR + TP	5667
PR + NR + TP	3735
PR + FP	3360
PR + NR + FP	1120

to quantify the data informativeness and even inform of the inconsistency of an hypothesis.

VI. DISCUSSION

Taking advantage of stable models offered by ASP, we provide a compact encoding of the BN synthesis from static and dynamical properties, with part of the complexity being parametrized. The method enables addressing scales and type of dynamical properties beyond the scope of already existing approaches.

Although not explicitly addressed in the encoding and evaluation, the use of ASP also enables efficient synthesis with optimization, e.g., finding BNs with minimal/maximal influence graph.

Negative reachability has a limited scalability due to the $O(n^2)$ variables and rules it generates (n being the dimension of the BNs). Future work will investigate SMT-like approaches to generate part of the constraints on the fly.

The considered properties are inspired by models of cellular differentiation. In such a context, having access to the complete set of candidate models enables uncovering influence motifs which are key for reproducing desired behaviours. Related to the applications, being able to account for universal properties on (reachable) attractors in the synthesis would increase the precision of inferred models, and constitutes a challenging direction.

ACKNOWLEDGEMENT

Part of the experiments was carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr).

REFERENCES

[1] T. Chatain, S. Haar, J. Kolčák, L. Paulevé, and A. Thakkar, "Concurrency in Boolean networks," *Natural Computing*, 2019.

[2] C. Terfve, T. Cokelaer, D. Henriques, A. MacNamara, E. Goncalves, M. K. Morris, M. v. Iersel, D. A. Lauffenburger, and J. Saez-Rodriguez, "CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms," *BMC Systems Biology*, vol. 6, no. 1, p. 133, 2012.

[3] J. Dorier, I. Crespo, A. Niknejad, R. Liechti, M. Ebeling, and I. Xenarios, "Boolean regulatory network reconstruction using literature based knowledge with a genetic algorithm optimization method," *BMC Bioinformatics*, vol. 17, no. 1, p. 410, 2016.

[4] M. Ostrowski, L. Paulevé, T. Schaub, A. Siegel, and C. Guziolowski, "Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming," *Biosystems*, vol. 149, pp. 139 – 153, 2016.

[5] B. Yordanov, S.-J. Dunn, H. Kugler, A. Smith, G. Martello, and S. Emmott, "A method to identify and analyze biological programs through automated reasoning," *Systems Biology and Applications*, vol. 2, 2016.

[6] S. A. Kauffman, "Metabolic stability and epigenesis in randomly connected nets," *Journal of Theoretical Biology*, vol. 22, pp. 437–467, 1969.

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

[8] J. Aracena, E. Goles, A. Moreira, and L. Salinas, "On the robustness of update schedules in Boolean networks," *Biosystems*, vol. 97, no. 1, pp. 1 – 8, 2009.

[9] T. Chatain, S. Haar, and L. Paulevé, "Boolean Networks: Beyond Generalized Asynchronicity," in *Cellular Automata and Discrete Complex Systems*, ser. LNCS, vol. 10875. Springer, 2018, pp. 29–42.

[10] T. Chatain, S. Haar, and L. Paulevé, "Most Permissive Semantics of Boolean Networks," *CoRR*, vol. abs/1808.10240, 2018.

[11] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

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

[13] F. Lin and Y. Zhao, "ASSAT: Computing answer sets of a logic program by SAT solvers," *Artificial Intelligence*, vol. 157, no. 1, pp. 115–137, 2004.

[14] D. Kleitman, "On Dedekind's problem: The number of monotone Boolean functions," *Proceedings of the American Mathematical Society*, vol. 21, no. 3, p. 677, 1969.

[15] D. Wiedemann, "A computation of the eighth dedekind number," *Order*, vol. 8, no. 1, pp. 5–6, 1991.

[16] X. Qiu, Q. Mao, Y. Tang, L. Wang, R. Chawla, H. A. Pliner, and C. Trapnell, "Reversed graph embedding resolves complex single-cell trajectories," *Nature Methods*, vol. 14, no. 10, pp. 979–982, 2017.