On Boolean automata networks (de)composition*

Kévin Perrot[1], Pacôme Perrotin[2], and Sylvain Sené[1]

[1]Université publique, France
[2]Aix-Marseille Univ., Toulon Univ., CNRS, LIS, Marseille, France

March 9, 2020

Abstract

Boolean automata networks (BANs) are a generalisation of Boolean cellular automata. In such, any theorem describing the way BANs compute information is a strong tool that can be applied to a wide range of models of computation. In this paper we explore a way of working with BANs which involves adding external inputs to the base model (via modules), and more importantly, a way to link networks together using the above mentioned inputs (via wirings). Our aim is to develop a powerful formalism for BAN (de)composition. We formulate three results: the first one shows that our modules/wirings definition is complete; the second one uses modules/wirings to prove simulation results amongst BANs; the final one expresses the complexity of the relation between modularity and the dynamics of modules.

Keywords: Boolean automata networks, modules, wirings, simulation.

1 Introduction

Boolean automata networks (BANs) can be seen as a generalisation of cellular automata that enables the creation of systems composed of Boolean functions over any graph, while cellular automata only operate over lattices of any dimension. The study of the dynamics of a BAN, that describes the set of all computations possible in such a system, is a wide and complex subject. From very simple networks computing simple Boolean functions to possibly infinite networks able to simulate any Turing machine, the number of configurations always grows exponentially with the size of the network, making any exhaustive examination of its dynamics impractical. The study of such dynamics is nevertheless an important topic which can impact other fields. BANs are for example used in the study of the dynamics of gene regulatory networks [8, 12, 17] in biology.

*This article is an extended version of [15].

Many efforts to characterise the dynamics of BANs have already been put forward. For example, some studies [1, 14] examine the behaviour of networks composed of interconnected cycles. The modularity of BANs has been studied from multiple perspectives. In particular from a static point of view [2, 13], and a functional one [4, 7, 16]. In this paper, we explore a compositional approach to BANs that allows to decompose a BAN into subnetworks called modules, and to compose modules together in order to form larger networks. We define a module as a BAN on which we add external inputs. These inputs are used to manipulate the result of the network computation by adding extra information. They can also be used to interconnect multiple modules, making more complex networks. Those constructions resemble the circuits described in Feder's thesis [10], and modules can be seen as a generalisation of circuits over any update mode.

Section 2 discusses the possible motivations for a (de)compositional study of BANs. Section 3 introduces BANs and update modes, and Sections 4 and 5 develop a formalism for the modular study of BANs, justified by a first theorem showing that any network can be created with modules and wirings. We also present an application of our definitions to BAN simulation in Section 6, leading to a second theorem stating that composing with local simulations is sufficient to (globally) simulate a BAN. Section 7 presents and analyses two illustrations of the principles presented in Section 2. Finally, Section 8 proposes an algebraic interpretation of the dynamics of modules, and leads to properties on said dynamics. It also contains the last theorem of this paper, which states the complexity of a decision problem over modules which concerns wirings, and their effect on the dynamics of said modules.

2 Motivations

BANs, despite being very simply defined locally, become complex to analyse as the representation of their dynamics grows exponentially in the size of their networks. BANs have been proven to be Turing-complete [5] and as most Turing-complete systems are able to show complex and emergent properties.

Yet, an important number of networks can be partially understood when viewed through the lens of functionality (what an object is meant to achieve). Functionality enables to use abstraction to reduce the considered network (or some part of it) to the computation of a function or the simulation of a dynamical system. Assuming a functionality of the parts of a network can let us conclude on the functionality of the network itself, at the cost of letting aside an absolute characterisation of its dynamics (which is often practically impossible). Such a functional interpretation aims at offering the possibility to make verifiable predictions in a short amount of time.

It is not known if every Boolean automata network can be cut into a reasonable amount of parts to which one can easily affect a functionality. We will justify our present argument by illustrating it in Section 7.

3 Boolean automata networks

3.1 Preliminary notations

Let us first describe some of the notations used throughout the paper. Let $f : A \to B$ be a mapping from set A to set B. For $S \subseteq A$ we denote $f(S) = \{b \in B \mid \exists a \in S, f(a) = b\}$. We denote $f\big|_S$ the restriction of f to the domain S, $f\big|_S \colon S \to B$ such that $f\big|_S(a) = f(a)$ for all $a \in S$. Let $\mathrm{dom}(f)$ be the domain of f, and $g \circ f$ the composition of f then g. For f and g two functions with disjoint domains of definition, we define $f \sqcup g$ as the function defined such that:

$$f \sqcup g(x) = \begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f) \\ g(x) & \text{if } x \in \mathrm{dom}(h) \end{cases} .$$

We denote $\mathbb{B} = \{0, 1\}$ the set of Booleans. For K a sequence of m elements, the sub-sequence from the i-th element to the j-th element is denoted $K_{[i,j]}$. We sometimes define functions without naming them with the notation $a \mapsto b$, signifying that for any input a the function will return b. For example, the function $n \mapsto 2 \times n$ is a function that takes a number n and returns the value of n multiplied by 2.

3.2 Definitions

A BAN is based upon a set S of automata. Each automaton in S, or node, is at any time in a state in $\mathbb{B}$. A *configuration* of the network is defined as a function $S \to \mathbb{B}$. The size of the network is the cardinal of S.

The state of every automaton is bound to evolve as a function of the configuration of the entire network. Each node has a unique function, called a local function that is predefined and does not change over time. A *local function* is thus a function f defined over $f : (S \to \mathbb{B}) \to \mathbb{B}$.

A *BAN* F defined over S is formally described as a function that assigns a local function to every node in a set S. As $(S \to \mathbb{B}) \to \mathbb{B}$ is the set of all possible local functions over S, it follows that F is defined as $F : S \to (S \to \mathbb{B}) \to \mathbb{B}$. For each $s \in S$, we denote $f_s = F(s)$ the local function of automaton s. Similarly, for x a configuration, we denote $x_s = x(s)$.

We can now define a naive way to update a BAN. From a configuration x, construct an iteration x' such that x'_s is obtained by the application of the local function f_s over x, or $x'_s = f_s(x)$, for every $s \in S$. This definition however is very limiting: it only allows so called parallel updates of our system. Some might imagine updating only some of the automata of the network, before using the resulting configuration to update the rest of the automata.

In general, the computation of a BAN should allow updates of automaton of the network by any order, and with any proportion of parallelism or sequentialisation. We set the following definition of an update over our BAN to be as general as possible.

Definition 1. *Any $\delta \subseteq S$ is an* update *over S.*

One can apply multiple consecutive updates to a BAN to effectively execute the BAN over an update mode. An *update mode* is simply a sequence of updates that is denoted Δ, where Δ_k is the k^{th} update of the sequence.

Slight changes to the update mode of a BAN can deeply change its computational capabilities [3, 11]. Most results that assume a parallel update mode cannot be applied to a sequential network; the reciprocal is also true.

We define the union operator between updates modes as it will be useful for the proof of our last theorem.

Definition 2. *Let Δ, Δ' be two update modes over a set S. The* union *of Δ and Δ' denoted $\Delta \cup \Delta'$ is the update mode defined as $(\Delta \cup \Delta')_k = \Delta_k \cup \Delta'_k$. The size of $\Delta \cup \Delta'$ is the maximum among the sizes of Δ and Δ'.*

We assume that $\Delta_k = \varnothing$ if k is greater than the size of Δ. Given an update δ, we can define the endomorphism F_δ over the set of all configurations. For every configuration x, we set $F_\delta(x)(s) = f_s(x)$ if $s \in \delta$, and $F_\delta(x)(s) = x(s)$ if $s \notin \delta$. In other words, the value of s in the new configuration is set to $f_s(x)$ only if $s \in \delta$, otherwise the Boolean affectation of s remains x_s. Now, we can define the execution of F in a recursive way.

Definition 3. *The* execution *of F over x, under the update mode Δ, is the function $F_\Delta : (S \to \mathbb{B}) \to (S \to \mathbb{B})$ defined as $F_{\Delta[1,k]}(x) = F_{\Delta_k}(F_{\Delta[1,k-1]}(x))$, with $F_{\Delta[1,1]}(x) = F_{\Delta_1}(x)$.*

Throughout this paper we represent BANs as graphs called interaction graphs. Interaction graphs are a classical tool in the study of BANs. For a BAN F defined over S, the interaction graph of F is the oriented graph $G = (S, \epsilon)$, where $(s, s') \in \epsilon$ if and only if the variable x_s influences the computation of the function $F(s')$.

4 Modules

Modules are BANs with external inputs. Such inputs can be added to any local function of a module, and any local function of a module can have multiple inputs. When a local function has n inputs, the arity of this function is increased by n. These new parameters are referred to by elements in a new set E: the elements of E describe the inputs of the module; those of S describe the internal elements of the module. To declare which input $e \in E$ is affected to each function f_s, we use function α.

Definition 4. *Let S and E be two disjoint sets. An* input declaration *over S and E is a function $\alpha : S \to \mathcal{P}(E)$ such that $\{\alpha(s) \mid s \in S\}$ is a partition of E.*

For each s, $\alpha(s)$ is the set of all external inputs of function f_s. The partition proposition is important because without it, some input could be assigned to multiple nodes, or to no node at all, which is contrary to our vision of input. To simplify notations, we sometimes denote $E_s = \alpha(s)$. Now, let us explicit the concept of a module.

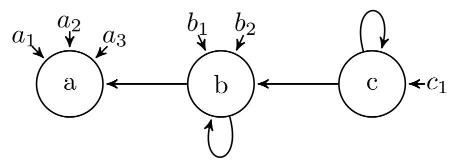

Figure 1: Interaction graph of the module detailed in Example 1.

Definition 5. *A* module *M over* (S, E, α) *is defined such that, for each* $s \in S$, $M(s)$ *is a function* $M(s) : (S \cup E_s) \to \mathbb{B}$.

If M is a module defined over $(S, \varnothing, s \mapsto \varnothing)$, M is also a BAN. To compute anything over this new system, we need a configuration $x : S \to \mathbb{B}$ and a configuration over the elements of E.

Definition 6. *An* input configuration *over* E *is a function* $i : E \to \mathbb{B}$.

Let x be a configuration over S, and i an input configuration over E. As x and i are defined over disjoint sets, we define $x \sqcup i$ as their union. Such an union, coupled with an update over S, is enough information to perform a computation over this new model.

Definition 7. *Let x be a configuration over S and i an input over E. Let δ be an update over S. The* computation *of M over x, i and δ, denoted $M_\delta(x \sqcup i)$, is the configuration over S such that $M_\delta(x \sqcup i)(s) = f_s(x \sqcup i\big|_{E_s})$ for each $s \in \delta$, and $M_\delta(x \sqcup i)(s) = x(s)$ for every $s \in S \setminus \delta$.*

In the following example, we assume a total order over $S \cup E$, allowing us to intuitively write configurations as binary words. For example, $x = 101$ means $x(a) = 1$, $x(b) = 0$ and $x(c) = 1$.

Example 1. *$S = \{a, b, c\}$, and $E = \{a_1, a_2, a_3, b_1, b_2, c_1\}$. We define α such that $\alpha(a) = \{a_1, a_2, a_3\}$, $\alpha(b) = \{b_1, b_2\}$ and $\alpha(c) = \{c_1\}$. Let M be a module over (S, E, α), such that $M(a) = x_b \vee a_1 \vee a_2 \vee a_3$, $M(b) = \neg x_b \vee x_c \vee \neg b_1 \wedge b_2$, and $M(c) = \neg c_1$. Let $x = 101$, $i = 000010$ and $\delta = \{a, b\}$. We get that $M_\delta(x \sqcup i) = M_{\{a,b\}}(101 \sqcup 000010)$ is such that $M_\delta(x \sqcup i)(a) = f_a(x \sqcup i\big|_{E_a}) = 0$, $M_\delta(x \sqcup i)(b) = f_b(x \sqcup i\big|_{E_b}) = 1$, and $M_\delta(x \sqcup i)(c) = x(c) = 1$. Therefore $M_\delta(x \sqcup i) = 011$. A representation of this module is pictured in Figure 1.*

Let us now define executions, while considering that the input configuration can change over time.

Definition 8. *Let $t > 1$. Let $I = (i_1, i_2, \ldots, i_{t-1})$ be a sequence of input configurations over E, $X = (x_1, x_2, \ldots, x_t)$ a sequence of configurations over S, and Δ an update mode over S of size t. (X, I, Δ) is an* execution *of M if for all $1 \leq k < t$, $x_{k+1} = M_{\Delta_k}(x_k \cup i_k)$.*

This definition allows for variation over the inputs over time. As this particular feature is not needed throughout this paper, we also propose a simpler definition of executions over modules which only allows fixed input values over time.

Definition 9. *Let i be an input configuration over E. The* execution *of M over $x \sqcup i$ with update mode Δ is an endomorphism over the set of all configurations, denoted M_Δ. It is defined as $M_{\Delta[1,k]}(x \sqcup i) = M_{\Delta_k}(M_{\Delta[1,k-1]}(x \sqcup i) \sqcup i)$, with $M_{\Delta[1,1]}(x \sqcup i) = M_{\Delta_1}(x \sqcup i)$.*

Similarly to a BAN, we can represent a module with an interaction graph. The definition is the same as the interaction graph of a BAN, to which is added smalls arrows which represent the inputs of the network, pointed on the nodes they are attached to.

5 Wirings

The external inputs of a module can be used to encode any information. For instance, we could encode any periodic (or non-periodic) sequence of Boolean words into the inputs of a given module. We could also encode the output of a given BAN or module, combining in some way the computational power of both networks. Such a composition of modules is captured by our definition of wirings. A wiring is an operation that links together different inputs and automata from one more or modules, thus forming bigger and more complex modules.

We decompose this compositional process into two different families of operators: the non-recursive and the recursive wirings. The first ones connect the automata of one module to the inputs of another; the second ones connect the automata of a module to its own inputs. A wiring, recursive or not, is defined by a partial map ω linking some inputs to automata. Let us first define non-recursive wirings.

Definition 10. *Let M, M' be modules defined over (S, E, α) and (S', E', α') respectively, such that S, S' and E, E' are two by two disjoint. A* non-recursive wiring *from M to M' is a partial map ω from E' to S.*

The new module result of the non-recursive wiring ω is denoted $M \rightarrowtail_\omega M'$ and is defined over $(S \cup S', E \cup E' \setminus \mathrm{dom}(\omega), \alpha_\omega)$. The input declaration of $M \rightarrowtail_\omega M'$ is $\alpha_\omega(s) = \alpha(s) \setminus \mathrm{dom}(\omega)$ (in particular, $\alpha_\omega(s) = \alpha(s)$ if $s \in S$). Given $s \in S \cup S'$, the local function $M \rightarrowtail_\omega M'(s)$, denoted f_s^ω, is defined as

$$f_s^\omega(x \sqcup i) = \begin{cases} f_s(x\big|_S \sqcup i\big|_{E_s}) & \text{if } s \in S \\ f'_s(x\big|_{S'} \sqcup i\big|_{E'_s \setminus \mathrm{dom}(\omega)} \sqcup (x \circ \omega\big|_{E'_s})) & \text{if } s \in S' \end{cases}.$$

In this new module, some inputs of M' have been assigned to the values of some elements of M. Such assignments are defined in the wiring ω. For any $s \in S \cup S'$, the function $M \rightarrowtail_\omega M'(s)$ (denoted f_s^ω) is defined over $(S \cup S' \cup \alpha_\omega(s)) \to \mathbb{B}$. In the case $s \in S'$, the image of $x \sqcup i$ is given by f'_s which expects a configuration on $S' \cup E'_s$: the configuration on S' is provided by x, and the configuration on E' is partly provided by i (on $E'_s \setminus \mathrm{dom}(\omega)$), and partly provided by $(x \circ \omega)$ (on $\mathrm{dom}(\omega) \cap E'_s$).

Definition 11. *Let M be a module over (S, E). A* recursive wiring *of M is a partial map ω from E to S.*

With ω defining now a recursive wiring over a module M, the result is similar if not simpler than in the definition of non-recursive wirings. The new module obtained from a recursive wiring ω on M is denoted $\circlearrowright_\omega M$ and is defined over $(S, E \setminus \mathrm{dom}(\omega), \alpha_\omega)$ with the input declaration defined as, for any $s \in S$, $\alpha_\omega(s) = \alpha(s) \setminus \mathrm{dom}(\omega)$. Given $s \in S$, x and i, the local function $\circlearrowright_\omega M(s)$ is denoted f^ω_s and is evaluated to $f^\omega_s(x \sqcup i) = f_s(x \sqcup i\big|_{E_s \setminus \mathrm{dom}(\omega)} \sqcup (x \circ \omega\big|_{E_s}))$.

Recursive and non-recursive wirings can be seen as unary and binary operators respectively, over the set of all modules. For any ω, we can define the operators $\rightarrowtail_\omega$ and $\circlearrowright_\omega$. For simplicity we define that $M \rightarrowtail_\omega M' = \varnothing$ and $\circlearrowright_\omega M = \varnothing$ if the wiring ω is not defined over the same sets as M or M'. Notice that both the recursive and non-recursive wirings defined by $\omega = \varnothing$ are well defined wiring. They define two operators, $\circlearrowright_\varnothing$ and $\rightarrowtail_\varnothing$, that will be useful later on.

Proposition 1. *The following statements hold.*

(i) $\quad \forall M, \quad \circlearrowright_\varnothing M = M.$

(ii) $\quad \forall M, M', \quad M \rightarrowtail_\varnothing M' = M' \rightarrowtail_\varnothing M.$

(iii) $\quad \forall M, M', M'', \quad M \rightarrowtail_\varnothing (M' \rightarrowtail_\varnothing M'') = (M \rightarrowtail_\varnothing M') \rightarrowtail_\varnothing M''.$

Proof.

$$\forall M, M', M \rightarrowtail_\varnothing M' = M' \rightarrowtail_\varnothing M.$$

By definition, $M \rightarrowtail_\varnothing M'$ and $M' \rightarrowtail_\varnothing M$ are both defined on $(S \cup S', E \cup E', \alpha \sqcup \alpha')$. For any $s \in S$, $M \rightarrowtail_\varnothing M'(s) = M' \rightarrowtail_\varnothing M(s)$ and for $s' \in S'$, $M \rightarrowtail_\varnothing M'(s') = M' \rightarrowtail_\varnothing M(s')$.

$$\forall M, \circlearrowright_\varnothing M = M.$$

By a similar argument, $\circlearrowright_\varnothing M$ is by definition defined on (S, E, α) such that $\circlearrowright_\varnothing M(s) = M(s)$ for any $s \in S$.

$$\forall M, M', M", M \rightarrowtail_\varnothing (M' \rightarrowtail_\varnothing M") = (M \rightarrowtail_\varnothing M') \rightarrowtail_\varnothing M".$$

By definition, the left side of this equation is defined over $(S \cup S' \cup S", E \cup E' \cup E", \alpha \sqcup \alpha' \sqcup \alpha")$ as is the right side of this equation. The two modules defining the same functions, we obtain the result. $\square$

For simplicity of notations, we will denote the empty non-recursive wiring as the union operator over modules: $M \cup M' = M \rightarrowtail_{\varnothing} M'$.

It is quite natural to want to put two modules together, by linking the input of the first to states of the second, and conversely. Our formalism allows this operation in two steps: first, use a non-recursive wiring to connect all of the desired inputs of the first module to states of the second module. Then, use a recursive wiring to connect back all of the desired inputs of the second module to states of the first module.

We now express that recursive and non-recursive wirings are expressive enough to construct any BAN or module, in Theorem 1. Our aim is to show that for any division of a module into smaller parts (partitioning), there is a way to get back to the initial module using only recursive and non-recursive wirings.

Definition 12. *Let (S, E, α). Let P be a set such that $\{S_p \mid p \in P\}$ is a partition of S. We define the* corresponding partition *of E as $\{E_p = \bigcup_{s \in S_p} \alpha(s) \mid p \in P\}$.*

Definition 13. *We can now develop the corresponding partition of the input declaration, and define the partition of M itself. For every $p \in P$, we define $\alpha_p = \alpha\big|_{S_p}$ over S_p and E_p.*

Definition 14. *For every $p \in P$, let Q_p verify $Q_p \cap S = \varnothing$ and $|Q_p| = |S|$, and let $\tau_p : S \to Q_p$ be a bijection. For any $p \in P$, the* sub-module *M_p over $(S_p, E_p \cup \tau_p(S \setminus S_p), \alpha_p)$ is defined for $s \in S_p$ as, for all $x : S \to \mathbb{B}$ and for all $i : E \to \mathbb{B}$,*

$$M_p(s)(x\big|_{S_p} \sqcup i_p) = M(s)(x \sqcup i),$$

where $i_p(e) = i(e)$ if $e \in E_p$ and $i_p(e) = x(\tau_p^{-1}(e))$ if $e \in \tau_p(S \setminus S_p)$.

In the previous definition, the purpose of each Q_p is to work as a representation of the set S for every sub-module M_p. Without it, every module M_p would have used the set $(S \setminus S_p) \cup E_p$ as input set. However our definition of wiring requires the input sets of the wired modules to be disjoint from each other. The sets Q_p are a workaround to bypass this technical point.

Example 2. *Let $S = \{a, b, c, d\}$, $E = \{e\}$, $P = \{r, s, t\}$ and $S_r = \{a, d\}$, $S_s = \{b\}$ and $S_t = \{c\}$. For each $p \in P$, we define $Q_p = \{a_p, b_p, c_p, d_p\}$. In the module M_r, $\alpha_r(a) = \varnothing$ and $\alpha_r(d) = \{b_r, c_r\}$. In the module M_s, $\alpha_s(b) = \{a_s\}$. In the module M_t, $\alpha_t(c) = \{e\}$. The modules M_r, M_s and M_t are defined over disjoint sets and can be wired (see Figure 2 for an illustration).*

As a reminder, the union operator over modules is defined to be the result of an empty non-recursive wiring.

Theorem 1. *Let M be a module and $\{M_p \mid p \in P\}$ any partition of that module, then there exists a recursive wiring ω such that $M = \circlearrowleft_{\omega} \left(\bigcup_{p \in P} M_p\right)$.*

Sketch of Proof: We construct ω to wire every link lost in partition P.

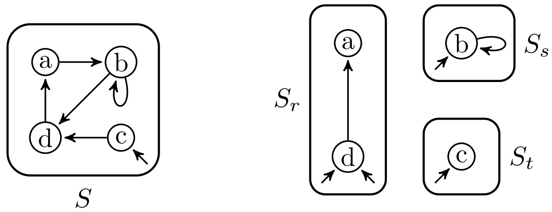

Figure 2: Interaction graphs related to Example 2. The interaction graph of the original module is on the left and the interaction graphs of the partition of M are on the right. Notice that we did not represent the input sets E, Q_r, Q_s and Q_t.

Proof. By definition of the empty wiring, the module $\bigcup_{p\in P} M_p$ is defined over $(S, E \cup \bigcup_{p\in P} \tau_p(S \setminus S_p), \bigsqcup_{p\in P} \alpha_p)$ and for all $s \in S$, $x : S \to \mathbb{B}$ and $i : E \to \mathbb{B}$ verifies

$$\left(\bigcup_{p\in P} M_p\right)(s)(x \sqcup i') = M(s)(x \sqcup i). \tag{1}$$

Knowing that $i'(e) = i(e)$ for $e \in E_s$, and $i'(s) = x(\tau_p^{-1}(s))$ for $s \in Q_p$. Let ω be the recursive wiring over $\bigcup_{p\in P} M_p$ with domain $\bigcup_{p\in P} \tau_p(S \setminus S_p)$ such that $\omega(q) = \tau_p^{-1}(q)$ given p such that $q \in Q_p$.

By definition of the recursive wiring, the module $\circlearrowright_\omega (\bigcup_{p\in P} M_p)$ is defined over the set (S, E, α). For all s, x, i, we now have that

$$\circlearrowright_\omega \left(\bigcup_{p\in P} M_p\right)(s)(x \sqcup i) = \left(\bigcup_{p\in P} M_p\right)(s)(x \sqcup i\big|_{E_s} \sqcup (x \circ \omega\big|_{\tau_p(S\setminus S_p)})). \tag{2}$$

By our definitions of ω and i', we have that $i' = i\big|_{E_s} \sqcup (x \circ \omega\big|_{\tau_p(S\setminus S_p)})$. From that, and Equations 1 and 2, we infer that for all s, x, i:

$$\circlearrowright_\omega \left(\bigcup_{p\in P} M_p\right)(s)(x \sqcup i) = M(s)(x \sqcup i).$$

Therefore for any s:

$$\circlearrowright_\omega \left(\bigcup_{p\in P} M_p\right)(s) = M(s),$$

which concludes the proof. □

Theorem 1 allows to say that our definition of wiring is complete: any BAN or module can be assembled with wirings. It can be reworked more algebraically. Let $\mathcal{M}$ denote the set of all modules (which includes $\varnothing$), and for any $n \in \mathbb{N}$, let

$\mathcal{M}_n$ denote the set of all modules of size n (we have $\mathcal{M} = \bigcup_{n\in\mathbb{N}} \mathcal{M}_n$). For any subset $A \subseteq \mathcal{M}$ we denote $\overline{A}^{\omega}$ the closure of A by the set of wiring operators $\bigcup_{\omega}\{\rightarrowtail_{\omega}, \circlearrowleft_{\omega}\}$. The following result is a direct corollary of Theorem 1.

Corollary 1. *The set of all modules is equal to the closure by any wiring of the set of modules of size* 1,

$$\mathcal{M} = \overline{\mathcal{M}_1}^{\omega}.$$

Proof. Trivially, $\overline{\mathcal{M}_1}^{\omega} \subseteq \mathcal{M}$. For any $M \in \mathcal{M}$ of size n, we know by Theorem 1 that in particular the n-partition of M into sub-modules of size 1 can be wired into the original module M. Therefore $\mathcal{M} = \overline{\mathcal{M}_1}^{\omega}$. □

Every module in $\mathcal{M}_1$ is of size 1, but as the set of inputs E of a module is not bounded, the set $\mathcal{M}_1$ is infinite. In our opinion, this corollary is enough to demonstrate that our definition of modules and wirings is sound.

6 Simulation

BANs are by nature complex systems and sometimes, we like to understand the computational power of a subset of them by demonstrating that they are able to simulate (or be simulated by) another subset of BANs. By *simulation*, we generally mean that a BAN is able to reproduce, according to some encoding, all the possible computations of another BAN.

Simulation is a powerful way to understand the limitations and possibilities of BANs. It is still difficult to prove if any two BANs simulate each other. In the present paper our aim is to prove that the proposition of simulating any BAN can be reduced in some cases to the proposition of locally simulating any Boolean function. Locally simulating a function means that a module reproduces any computation of that function, when the parameters of the function are encoded in the module inputs. Our claim is that if we can locally simulate every function of a BAN, in a way such that the simulating modules are able to communicate with each other, then we can simulate the same BAN with a bigger module which is obtained by a wiring over the locally simulating modules. In this context, modules become a strong tool to reduce the complexity of simulation (which is a global phenomena) to a local scale, which is more tractable.

Let us go into further details. For F a BAN over the set S, our aim is to simulate F. For this purpose, for each $a \in S$, we create M_a, a module which is defined over some sets (T_a, E_a, α_a) and locally simulates the function f_a. To assert this local simulation we need to define a Boolean encoding ϕ_a over the configurations of M_a. We also need to define how these modules communicate with each other, and in the end how they will be wired together. For any couple $a, b \in S$ such that $a \neq b$, we define the set $U_{a,b}$ as a subset of T_a. This set represents all the automata of M_a that are planned to be connected to inputs of M_b. We can say that the elements of $U_{a,b}$ are the only way for the module M_a to send information to the module M_b. We define which information is sent from M_a to M_b at any time with a Boolean encoding $\phi_{a,b}$ over the set of configurations on $U_{a,b}$. By definition we always have that if $U_{a,b} \neq \varnothing$, then $\phi_a(x\big|_{T_a}) \neq \bullet \Rightarrow \phi_{a,b}(x\big|_{U_{a,b}}) = \phi_a(x\big|_{T_a})$. This means that if a module encodes an information ($\bullet$ being the absence of information, *i.e.* in this case $\phi_a(x\big|_{T_a})$ equals 0 or 1), the same information is sent from that module to each module that is meant to receive information from it. In other words, all encodings are coherent.

Now that our modules are set to communicate with each other, we only need to wire them to each other. The precise nature of this wiring is defined, for every pair $a, b \in S$ such that $a \neq b$, by the function $I_{a,b} : E_b \to U_{a,b}$ which we call interface between a and b. By definition:

- for every $s \in U_{a,b}$, there exists $e \in E_b$ such that $I_{a,b}(e) = s$ (surjectivity);
- for every $b \in S$, $\bigsqcup_a I_{a,b}$ is a total map from E_b to $\bigcup_a U_{a,b}$.

With such an interface defined for every pair (a, b), the final wiring connecting all modules together is decomposed in two steps. The first one empty-wires every module together, the second one applies a recursive wiring which is defined as the union of every interface $I_{a,b}$. The last condition that we have stated over the definition of an interface lets us know that the obtained module has no remaining inputs; it can be considered as a BAN, defined over $T = \bigcup_{a\in S} T_a$. All these sets are illustrated in Figure 3.

Example 3. *Let $S = \{a, b, c, d\}$. Let $T_a = \{e, f, g, h\}$, $T_b = \{i, j, k\}$, $T_c = \{l, m\}$ and $T_d = \{n\}$. Let $T = T_a \cup T_b \cup T_c \cup T_d$. Let $E_a = \{e_g, e_h\}$, $E_b = \{e_i, e_k, e'_k\}$, $E_c = \{e_m\}$ and $E_d = \{e_n\}$. Let $U_{a,b} = \{f, g\}$, $U_{b,c} = \{j\}$, $U_{c,d} = \{l\}$, $U_{d,a} = U_{d,b} = \{n\}$, and any other U set empty. We will define interfaces as the following: $I_{a,b}(e_i) = f$, $I_{a,b}(e_k) = g$, $I_{b,c}(e_m) = j$, $I_{c,d}(e_n) = l$, $I_{d,a}(e_h) = n$, $I_{d,a}(e_g) = n$ and $I_{d,b}(e'_k) = n$ (see Figure 3).*

Definition 15. *Let A be a set. A* Boolean encoding *over A is a function $\phi : (A \to \mathbb{B}) \to (\{0, 1, \bullet\})$, such that there exists at least one x such that $\phi(x) = 0$ and one x such that $\phi(x) = 1$.*

For $x : A \to \mathbb{B}$ (a Boolean configuration over a set A), $\phi(x) = 1$ means that x encodes a 1, $\phi(x) = 0$ means that x encodes a 0, and $\phi(x) = \bullet$ means that x does not encode any value. Each ϕ_a is defined as an encoding over T_a, and each $\phi_{a,b}$ as an encoding over $U_{a,b}$.

By definition we enforce that

$$\text{if } U_{a,b} \neq \varnothing, \text{ then } \phi_a(x\big|_{T_a}) \neq \bullet \Rightarrow \phi_{a,b}(x\big|_{U_{a,b}}) = \phi_a(x\big|_{T_a}).$$

Given a BAN on S and some $a \in S$, let us now define the local simulation of function f_a by a module M_a. We want to express that given any configuration $x : S \to \mathbb{B}$, all the configurations $x' : T_a \to \mathbb{B}$ and input configurations $i' : E_a \to \mathbb{B}$ such that x', i' encode the same information as x, the result of the dynamics on x', i' in the simulating module must encode the result of the dynamics on x in the simulated automaton. To express that x' encodes the state of a in x is easy:

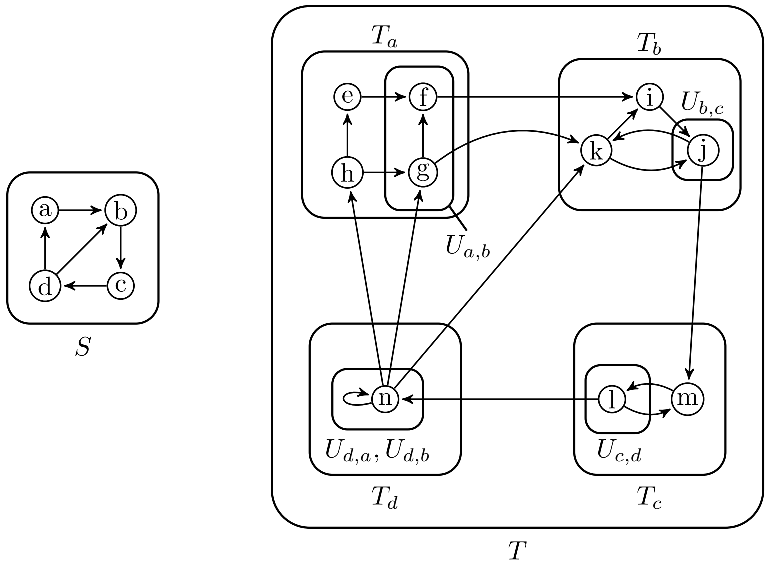

Figure 3: Interaction graphs of the modules detailed in Example 3. The interaction graph of the original BAN is on the left and the interaction graph of the simulating BAN is on the right. The simulating BAN is decomposed into four sub-modules, one for each node in S. Notice that we did not represent the input sets E_a, E_b, E_c and E_d. The connections between the sets T_a, T_b, T_c and T_d are based upon the interfaces defined in the example.

$\phi_a(x') = x_a$. To express that i' encodes the state of all $b \neq a$ in x requires an additional notation. On the one hand we have $\phi_{b,a} : (U_{b,a} \to \mathbb{B}) \to (\{0, 1, \bullet\})$, and on the other hand we have $i' : E_a \to \mathbb{B}$ describing the input-configuration of module M_a, and $I_{b,a} : E_a \to U_{b,a}$ describing the interface from b to a. To plug these objects together, we put forward the hypothesis that if $I_{b,a}(e) = I_{b,a}(e')$, then $i'(e) = i'(e')$ for any $e, e' \in E_a$. This hypothesis is justified by the fact that the wiring applied by $I_{b,a}$ enforces the value of two inputs connected to the same element to be the same. Now, we define $i' \circ I^{-1}_{b,a}$ the configuration over $U_{b,a}$ such that $i' \circ I^{-1}_{b,a}(s) = i'(e)$ for any e such that $I_{b,a}(e) = s$. By our hypothesis this configuration is well defined.

Definition 16. *Let $a \in S$, f_a be a Boolean function over S and M_a a module over (T_a, E_a, α_a), with ϕ_a (resp. $\phi_{b,a}$) a Boolean encoding over T_a (resp. $U_{b,a}$). Given a finite update mode Δ over T_a, M_a* locally simulates *f_a, denoted by $M_a \prec_\Delta f_a$, if for all $x : S \to \mathbb{B}$,*

1. *and for all $x' : T_a \to \mathbb{B}$ such that $\phi_a(x') = x_a$,*
2. *and for all $i' : E_a \to \mathbb{B}$ such that for all $b \neq a$ we have $\phi_{b,a}(i' \circ I^{-1}_{b,a}) = x_b$,*

3. we have:

$$\phi_a(M_{a\Delta}(x' \sqcup i')) = f_a(x).$$

This local simulation can be defined on a wide range of update modes Δ. To ensure that the simulation works as planned at the global scale, we restrict the range of update modes Δ used for the local simulations, to those where no automata with input(s) are updated later than the first update.

Definition 17. *An update mode Δ over a module M is defined to be* input-first *if for all $k > 1$ and all $s \in \Delta_k$, we have $\alpha(s) = \varnothing$.*

Definition 18. *We define that M is able to* input-first simulate *f if there exists an input-first Δ such that $M \prec_\Delta f$.*

Intuitively, such update modes let us make parallel the computation of modules; all information between modules is communicated simultaneously at the first frame of computation (update), followed by isolated updates in each module. To define global simulation, we introduce the global encoding $\Phi : (S \to \mathbb{B}) \to (S' \to \mathbb{B}) \cup \{\bullet\}$ which always verifies that for all $x' : S' \to \mathbb{B}$, there exists $x : S \to \mathbb{B}$ such that $\Phi(x) = x'$.

Definition 19. *Let F and F' be two Boolean automata networks over S and S' respectively. We define that F* simulates *F', denoted by $F \prec F'$, if there exists a global encoding Φ such that for all x', x such that $\Phi(x) = x'$, and for all $\delta' \subseteq S'$, there exists a finite update mode Δ over S such that $\Phi(F_\Delta(x)) = F'_{\delta'}(x')$.*

Given the definitions of local and global simulation, for any BAN F over a set S, we define each module M_a as earlier, each defined over (T_a, E_a, α_a), along side each set $U_{a,b}, I_{a,b}$ and each encoding $\phi_a, \phi_{a,b}$.

Theorem 2. *Let F be a BAN over S. For each $a \in S$, let M_a be a module over (T_a, E_a, α_a) that locally simulates $F(a)$ in an input-first way. There exists a recursive wiring ω over $T = \bigcup_{a \in S} T_a$ such that*

$$\circlearrowright_\omega \left(\bigcup_{a \in S} M_a \right) \prec F.$$

Sketch of Proof: We prove that the execution of the module M obtained from the wiring ω can be built from the execution of each M_a. We apply the hypothesis of local simulation on each M_a, and obtain a global simulation.

Proof. By definition of the empty wiring, $\bigcup_{a \in S} M_a$ is defined over $(T, \bigcup_{a \in S} E_a, \bigsqcup_{a \in S} \alpha_a)$. Let $\omega = \bigcup_{a,b \in S, a \neq b} I_{a,b}$. By definition of $I_{a,b}$, we can easily see that the module $M = \circlearrowright_\omega \left(\bigcup_{a \in S} M_a\right)$ is defined over $(T, \varnothing, s \mapsto \varnothing)$ and can be seen as a Boolean automata network. Let us prove that, for all $a \in S$, for all input-first simulating update mode Δ for the module M_a, for any Δ' update mode over $T \setminus T_a$, and for any $x : T \to \mathbb{B}$, the following equation holds:

$$M_{\Delta \cup \Delta'}(x)\big|_{T_a} = M_{a\Delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a})). \tag{3}$$

At the first step of the execution, the wiring ω implies that for any $s \in T_a$, for any x, $M(s)(x) = \left(\bigcup_{a \in S} M_a\right)(s)(x \sqcup (x \circ \omega))$. From the definition of the empty wiring, we can deduce in particular that $M(s)(x) = M_a(s)(x\big|_{T_a} \sqcup (x \circ \omega\big|_{E_a}))$. By definition of the interfaces, this notation is equivalent to $\forall s \in T_a, M(s)(x) = M_a(s)(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))$.

Let us define $A = \{s \in T_a \mid \alpha(s) \neq \varnothing\}$ and $B = T_a \setminus A$. By the definition of Δ, we know that $s \in \Delta_k$ with $k > 0$ implies $s \in B$.

Let us look at the A part of this problem. Let $\delta = \Delta_0$ and $\delta' = \Delta'_0$. We can trivially deduce from the previous statement that:

$$M_{\delta \cup \delta'}(x)\big|_A = M_{a\delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))\big|_A.$$

Furthermore, there is no $s \in A$ such that $s \in \Delta_k$ for any $k > 0$. We can simply conclude since no update is made to any function of A in the rest of the execution that $M_{\delta \cup \delta'}(x)\big|_A = M_{\Delta \cup \Delta'}(x)\big|_A$, and that $M_{a\delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))\big|_A = M_{a\Delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))\big|_A$. In conclusion of this A part, $M_{\Delta \cup \Delta'}(x)\big|_A = M_{a\Delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))\big|_A$.

Let us now consider the B part of the problem. For $s \in B$, we have $M(s)(x) = M_a(s)(x\big|_{T_a} \sqcup (x \circ \omega\big|_{E_s}))$. By definition of B, $s \in B$ implies $E_s = \varnothing$. We can conclude that $\forall s \in B, M(s)(x) = M_a(s)(x\big|_{T_a})$. We deduce, for any $\delta \subseteq T_a$ and $\delta' \subseteq T \setminus T_a$, that $M_{\delta \cup \delta'}(x)\big|_B = M_{a\delta}(x\big|_{T_a} \sqcup i)\big|_B$, for i any input configuration over E_a. By a simple recursive demonstration, we can easily show that $M_{\Delta \cup \Delta'}(x)\big|_B = M_{a\Delta}(x\big|_{T_a} \sqcup i)\big|_B$.

Reuniting the A and B parts of this demonstration, we obtain that $M_{\Delta \cup \Delta'}(x) = M_{a\Delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))\big|_A \cup M_{a\Delta}(x\big|_{T_a} \sqcup i)\big|_B$. Assuming $i = x \circ \bigsqcup_b I_{b,a}$, we obtain $M_{\Delta \cup \Delta'}(x) = M_{a\Delta}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))$, and prove the lemma described in Equation 3.

Let us now define $\Phi : (T \to \mathbb{B}) \to (S \to \mathbb{B}) \cup \{\varnothing\}$ such that, for any $x : T \to \mathbb{B}$, $\Phi(x) = \varnothing$ if there exists $a \in S$ such that $\phi_a(x\big|_{T_a}) = \varnothing$, and $\Phi(x)(a) = \phi_a(x\big|_{T_a})$ otherwise. Let x and x' such that $\Phi(x) = x'$, and $x' \neq \varnothing$. Let $\delta \subseteq S$ be an update over F. Let us define, for any $a \in \delta$, the update mode Δ_a such that Δ_a is an input-first update mode upon which M_a simulates the function $F(a)$; by hypothesis such an update mode can always be found.

Let us define the update mode Δ over T such that $\Delta = \bigcup\{\Delta_a \mid a \in \delta\}$. We will now prove that $\Phi(M_\Delta(x)) = F_\delta(x')$. First, we can clearly see that $M_\Delta(x) = \bigsqcup\{M_\Delta(x)\big|_{T_a} \mid a \in S\}$, which can be developed into $M_\Delta(x) = \bigsqcup\{M_\Delta(x)\big|_{T_a} \mid a \in \delta\} \sqcup \bigsqcup\{x\big|_{T_a} \mid a \in S \setminus \delta\}$, from which we infer:

$$M_\Delta(x) = \bigsqcup\{M_{\Delta_a \cup \bigcup_{b \in \delta, b \neq a} \Delta_b}(x)\big|_{T_a} \mid a \in \delta\} \sqcup \bigsqcup\{x\big|_{T_a} \mid a \in S \setminus \delta\}.$$

Using the lemma formulated in Equation 3, this can be rewritten into:

$$M_\Delta(x) = \bigsqcup_{a\in\delta} M_{a\Delta_a}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a})) \sqcup \bigsqcup_{a\in S\setminus\delta} x\big|_{T_a}.$$

As the result of an execution of the module M_a is always defined as a configuration over T_a, we can infer the following encoding of $M_\Delta(x)$ by Φ :

$$\Phi(M_\Delta(x))(a) = \begin{cases} \phi_a(M_{a\Delta_a}(x\big|_{T_a} \sqcup (x \circ \bigsqcup_b I_{b,a}))) & \text{if } a \in \delta \\ \phi_a(x\big|_{T_a}) & \text{if } a \in S \setminus \delta \end{cases}.$$

We know by definition of x and x' that $\phi_a(x\big|_{T_a}) = x'_a$ and that $\phi_{b,a}(x \circ I_{b,a} \circ I_{b,a}^{-1}) = \phi_{b,a}(x\big|_{U_{b,a}}) = \phi_b(x\big|_{T_b}) = x'_b$ by definition of $\phi_{b,a}$. From this we can apply the local simulation definition and obtain:

$$\Phi(M_\Delta(x))(a) = \begin{cases} f_a(\Phi(x)) & \text{if } a \in \delta \\ \phi_a(x\big|_{T_a}) & \text{if } a \in S \setminus \delta \end{cases} = \begin{cases} f_a(\Phi(x)) & \text{if } a \in \delta \\ \Phi(x)(a) & \text{if } a \in S \setminus \delta. \end{cases}.$$

Futhermore, by the definition of an update over F, we can write that:

$$F_\delta(x')(a) = \begin{cases} f_a(x') & \text{if } a \in \delta \\ x'(a) & \text{if } a \in S \setminus \delta \end{cases}.$$

Finally, by definition of $x' = \Phi(x)$:

$$F_\delta(x')(a) = \begin{cases} f_a(\Phi(x)) & \text{if } a \in \delta \\ \Phi(x)(a) & \text{if } a \in S \setminus \delta \end{cases},$$

which implies $\Phi(M_\Delta(x)) = F_\delta(x')$, and concludes the proof. □

This theorem helps us investigate if every BAN can be simulated by a BAN with a given proposition, hence justifying that theoretical studies can impose some restrictions without loss of generality. If every function f can be locally simulated by a given module with a proposition $\mathcal{P}$, and if proposition $\mathcal{P}$ is preserved over wirings, then we know that any BAN can be simulated by another BAN with the proposition $\mathcal{P}$. This is formally proven for the two following cases, involving disjunctive clauses and monotony respectivly:

Corollary 2. *Let F be a BAN. There exists F' such that $F' \prec F$ and every function of F' is a disjunctive clause.*

Proof. With Theorem 2 in mind, we only need to demonstrate that for any function f, there exists a module locally simulating it in a input-first way, in which every function is a disjunctive clause.

Let us consider F a BAN set over S. Let $a \in S$. We decompose f_a into a set of disjunctive clauses C_a such that $f_a(x) = \bigwedge_{c\in C_a} c(x)$.

Let $M_a = (T_a, E_a, \alpha_a)$ be a module with $T_a = \{u_c \mid c \in C\} \cup \{r_a\}$, $E_a = \{e_{b,c,a} \mid a \neq b$, and the variable x_b is included in clause $c\}$. For all b, c, $e_{b,c,a} \in$

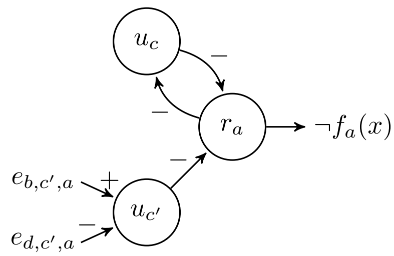

Figure 4: Interaction graph of the locally disjunctive module for the example function $f_a(x) = x_a \wedge (\neg x_b \vee x_d)$. We name the clauses of f_a as $c = x_a$ and $c' = \neg x_b \vee x_d$. Notice that most of the signs are inversed to simulate a AND gate.

$\alpha(u_c)$ if and only if x_b is included in clause c. For $c \neq c'$, $e_{b,c,a} \notin \alpha(u'_c)$ and $\alpha(r_a) = \varnothing$.

For $c \in C_a$, x a configuration over T_a and e a configuration over E_a, $M_a(u_c)$ is the function described by $f_{u_c}(x, e) = c(x_a \mapsto \neg x(r_a) \sqcup x_b \mapsto \neg e(e_{b,c,a}))$. The function $M(r)$ is the function $f_{r_a}(x, e) = \bigvee_{c \in C_a} \neg x(u_c)$.

This local module is shaped as a pyramid where the base is constitued of one node for every disjunctive clause of the simulated function, and the top of exactly one node that represents the result of the function. It follows from this definition that every function of this module is a disjunctive clause. An illustrated example of such a local module is presented in Figure 4.

We define $U_{b,a}$ such that $U_{b,a} = \{r_a\}$ if the variable x_b is included in one of the clauses of the function f_a, and $U_{b,a} = \varnothing$ otherwise.

The encodings ϕ_a and $\phi_{b,a}$ for every b such that $U_{b,a} \neq \varnothing$ are defined such that $\phi_a(x) = \phi_{b,a}(x\big|_{U_{b,a}}) = \neg x(r_a)$. This means that the node r represents the inverse of the result of the function.

We always define $I_{b,a}(e_{b,c,a}) = r_b$. More intuitively, to resolve the value of the variable x_b in a clause of f_a, look for the value of the node r_b in the local module M_b. We reverse it back to the correct value thanks to the inversion of each input of each clause automaton.

We shall now prove that M_a locally simulates f_a in an input-first way.

Let $\Delta_a = (\{u_c \mid c \in C_a\}, \{r_a\})$ be an input-first update mode for the module M_a. We will sometimes note $\Delta_a = (\delta, \delta_r)$ in further developments.

Let x be a configuration over F. Let x' be a configuration over T_a such that $\phi_a(x') = x_a$. Let i' be an input configuration over E_a such that for any $b \neq a$, $\phi_{b,a}(i' \circ I_{b,a}^{-1}) = x_b$.

Such a x' is a configuration over T_a with $x'(r_a) = \neg x_a$. Such a i' is a configuration over E_a such that $i'(e_{b,c,a}) = \neg x_b$ for every b and c. Such configurations are well defined and can always be found.

To prove the above local simulation, we have to show that $\phi_a(M_{a\Delta_a}(x' \sqcup i')) = f_a(x)$, which can be simplified into $\neg M_{a\Delta_a}(x' \sqcup i')(r_a) = f_a(x)$. By the definition of an execution over a module, this can be developed into:

$$
\begin{aligned}
&M_{a\Delta_a}(x' \sqcup i')(r_a) = f_{r_a}(M_{a\delta}(x' \sqcup i'), i') \\
&\quad = \bigvee_{c \in C_a} \neg M_{a\delta}(x' \sqcup i')(u_c) = \bigvee_{c \in C_a} \neg c(x_a \mapsto \neg x'(r_a) \sqcup x_b \mapsto \neg i'(e_{b,c,a})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad = \neg \bigwedge_{c \in C_a} c(x_a \mapsto \neg x'(r_a) \sqcup x_b \mapsto \neg i'(e_{b,c,a})).
\end{aligned}
$$

By the above hypothesis, this can be simplified into:

$$M_{a\Delta_a}(x' \sqcup i')(r_a) = \neg \bigwedge_{c \in C_a} c(x_a \mapsto x_a \sqcup x_b \mapsto x_b),$$

which let us simply conclude that:

$$\neg M_{a\Delta_a}(x' \sqcup i')(r_a) = \bigwedge_{c \in C_a} c(x) = f_a(x)$$

wich proves the local simulation of f_a by M_a. From this result and the fact that the proposition that function are locally defined by disjunctive functions isn't broken by any wiring, we conclude the result.

□

Corollary 2 utilizes the Theorem 2 to show that any BAN can be simulated by a BAN only composed of disjunctions as local functions. This result was known, and is presented here to give an example of an application of Theorem 2.

Another example of this theorem is the demonstration that any BAN can be simulated by a BAN only composed of monotone local functions. This second corollary requires the demonstration of a lemma which follows.

Lemma 1. *Let S be a set. Let $x : S \to \mathbb{B}$. Let f be a Boolean function over S. Let $S' = \{s, s^- \mid s \in S\}$. There exists f' a monotone Boolean function over S' such that $f(x) = f'(x \sqcup s^- \mapsto \neg x(s))$.*

Proof. For reminder, we assume that $x \leq x'$ if and only if $x(s) \leq x'(s)$ for every $s \in S$, and that f' is monotone if and only if $x \leq x' \Rightarrow f'(x) \leq f'(x')$.

For x' an execution over S', and $s \in S$, we note $\mathit{code}(x', s) \Leftrightarrow x'(s) = \neg x'(s^-)$. Let f be a Boolean function over S.

We define f' over the set S' as the following:

$$f'(x') = \begin{cases} f(x'\big|_S) & \text{if for every } s \in S, \mathit{code}(x', s) \\ 1 & \text{if for every } s \in S, \neg\mathit{code}(x', s) \Rightarrow x'(s) = x'(s^-) = 1 \\ 0 & \text{otherwise} \end{cases}.$$

From this definition we clearly see that for all configurations x over S, $f(x) = f'(x \sqcup s^- \mapsto \neg x(s))$. Let us now show that f' is monotone.

Let x' and $x"$ be two configurations over S', such that $x' < x"$. This implies that for all $s' \in S'$, $x'(s') \leq x"(s')$ and that there is at least one $s' \in S'$ such that $x'(s') < x"(s')$. This clearly implies that the propositions $\forall s \in S, code(x', s)$ and $\forall s \in S, code(x", s)$ cannot both be true.

Let us suppose $\forall s \in S, code(x', s)$ and $\exists s \in S, \neg code(x", s)$. As $x' < x"$, for every $s \in S$ such that $\neg code(x", s)$, we now that $x"(s) = x"(s^-) = 1$. This implies that $f'(x") = 1$, and that $f'(x') \leq f'(x")$.

Let us now suppose that $\exists s \in S, \neg code(x', s)$ and $\forall s \in S, code(x", s)$. By a similar argument, we now suppose that for every $s \in S$ such that $\neg code(x', s)$, we have that $x'(s) = x'(s^-) = 0$. This implies that $f'(x') = 0$, and $f'(x') \leq f'(x")$.

Let us finally suppose that $\exists s \in S, \neg code(x', s)$ and $\exists s \in S, \neg code(x", s)$. In this case, we know that $f'(x') = 1 \Rightarrow f'(x") = 1$ since $x' < x"$. Assuming $f'(x') = 0$ naturally implies $f'(x') \leq f'(x")$. □

We can now demonstrate the following corollary.

Corollary 3. *Let F be a BAN. There exists F' such that $F' \prec F$ and every function of F' is monotone.*

Proof. Let F be a BAN defined over set S. For every $a \in S$, we define $M_a = (T_a, E_a, \alpha_a)$ a module with $T_a = \{u_{a,-}, u_{a,+}\}$, $E_a = \{e_{b,a,+}, e_{b,a,-} \mid x_b \text{ is included in } f_a\}$. The function α is such that $e_{b,a,+} \in E_a \Rightarrow e_{b,a,+} \in \alpha(u_{a,+})$ and $e_{b,a,-} \in E_a \Rightarrow e_{b,a,-} \in \alpha(u_{a,-})$.

Let S be a configuration over S. We define the monotone function f'_a over the set $\{s, s^- \mid s \in S\}$ that for every configuration x verifies $f_a(x) = f'_a(x \sqcup s^- \mapsto \neg x(s))$. The existence of such a function is given by Lemma 1.

For x' a configuration over T_a, and i a configuration over E_a, We define $M_a(u_{a,+})$ as a function that verifies:

$$
\begin{aligned}
&M_a(u_{a,+})(x' \sqcup i) = \\
&\quad f'_a(a \mapsto x'(u_{a,+}) \sqcup a^- \mapsto x'(u_{a,-}) \sqcup \bigsqcup_{b \neq a} \left(b \mapsto i'(e_{b,a,+}) \sqcup b^- \mapsto i'(e_{b,a,-})\right))
\end{aligned}
$$

The function $M_a(u_{a,-})$ is given by $M_a(u_{a,-})(x' \sqcup i) = \neg M_a(u_{a,+})(x' \sqcup i)$.

This local module is composed of two automata, one that computes the original function and one that computes the negation of the original function. This allows us to simulate the original network while being locally monotone. The monotony is given by the fact that the configurations used for simulation are now incomparable to each other. A representation of an example is presented in Figure 5.

We define $U_{b,a}$ such that $U_{b,a} = T_a$ if the variable x_b is included in function f_a, and $U_{b,a} = \varnothing$ otherwise.

The encodings ϕ_a and $\phi_{b,a}$ for every b such that $U_{b,a} \neq \varnothing$ are defined by:

$$
\phi_a(x') = \phi_{b,a}(x') = \begin{cases} 1 & \text{if } x'(u_{a,+}) = 1 \text{ and } x'(u_{a,-}) = 0 \\ 0 & \text{if } x'(u_{a,+}) = 0 \text{ and } x'(u_{a,-}) = 1 \\ \bullet & \text{otherwise} \end{cases} .
$$

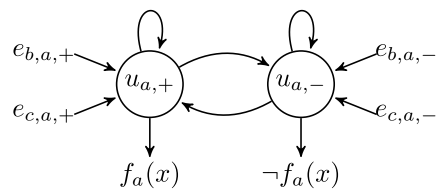

Figure 5: Interaction graph of the locally monotone module for the example function $f_a(x) = x_a \wedge (\neg x_b \vee x_c)$. As x_a is present in the local function, the two automata composing this module loop between each other and themselves.

For every b such that $U_{b,a} \neq \varnothing$, we define $I_{b,a}(e_{b,a,+}) = u_{b,+}$ and $I_{b,a}(e_{b,a,-}) = u_{b,-}$. In other words, the positive (resp. negative) value of automaton b is given by the value of the positive (resp. negative) node of the local module M_b.

Let us prove that M_a locally simulates f_a in a input-first way.

Let $\Delta_a = \{T_a\}$ be an input-first way update mode for the module M_a. Let x be a configuration over F. Let x' be a configuration over T_a such that $\phi(x') = x_a$. Let i' be an input configuration over E_a such that for any $b \neq a$, $\phi_{b,a}(i' \circ I_{b,a}^{-1}) = x_b$.

Such a x' verifies $x'(u_{a,+}) = x_a$ and $x'(u_{a,-}) = \neg x_a$. Such a i' verifies $i'(e_{b,a,+}) = x_b$ and $i'(e_{b,a,-}) = \neg x_b$ for every $b \neq a$. Theses configurations are well defined.

To prove the local simulation of f_a by M_a, we have to show that $\phi_a(M_{a\Delta_a}(x' \sqcup i')) = f_a(x)$. This is equivalent to:

$$\Leftrightarrow \begin{cases} M_a(u_{+,a})(x' \sqcup i') = f_a(x) \\ M_a(u_{-,a})(x' \sqcup i') = \neg f_a(x) \end{cases}$$

$$\Leftrightarrow \begin{cases} M_a(u_{+,a})(x' \sqcup i') = f_a(x) \\ \neg M_a(u_{+,a})(x' \sqcup i') = \neg f_a(x) \end{cases}$$

$$\Leftrightarrow M_a(u_{+,a})(x' \sqcup i') = f_a(x)$$

$$\Leftrightarrow f'_a(a \mapsto x'(u_{a,+}) \sqcup a^- \mapsto x'(u_{a,-}) \sqcup \bigsqcup_{b \neq a} \big(b \mapsto i'(e_{b,a,+}) \sqcup b^- \mapsto i'(e_{b,a,-})\big)) = f_a(x).$$

We noticed earlier that $x'(u_{a,+}) = \neg x'(u_{a,-})$ and that $i'(e_{b,a,+}) = \neg i'(e_{b,a,-})$ for every $a \neq b$. This implies that our this evaluation of f'_a can be developed as follows:

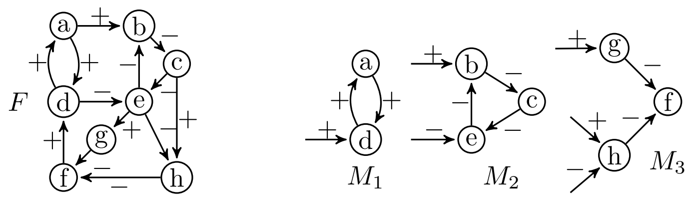

Figure 6: Representation of a Boolean automata network F next to the three different modules M_1, M_2 and M_3 that compose it. The function of each automaton is defined as a disjunctive clause with a positive literal for each incident "+" edge, and a negative literal for each incident "−" edge. For example, $f_h(x) = x_c \vee \neg x_e$.

$$
\begin{aligned}
&f'_a(a \mapsto x'(u_{a,+}) \sqcup a^- \mapsto x'(u_{a,-}) \sqcup \bigsqcup_{b \neq a} \big(b \mapsto i'(e_{b,a,+}) \sqcup b^- \mapsto i'(e_{b,a,-})\big)\big) \\
&\quad = f_a(a \mapsto x'(u_{a,+}) \sqcup \bigsqcup_{b \neq a} b \mapsto i'(e_{b,a,+})) = f_a(a \mapsto x_a \sqcup \bigsqcup_{b \neq a} b \mapsto x_b) \\
&\quad = f_a(x),
\end{aligned}
$$

which proves that M_a locally simulates f_a. Using this lemma, knowing that the Lemma 1 implies the monotony of each function in the local modules and the simple fact that local monotony is not broken by any wiring, we use Theorem 2 to conclude this proof.

□

Theorem 2 and consequent corollaries only concerns BANs, and it could be expected to obtain more general results concerning modules. Such a result would need a definition of simulation between modules, and such a definition would imply an interpretation of the information provided by the simulating module's inputs. We choose not to develop this particular idea, as this theorem was only meant to apply to BANs. A generalisation of this result to modules would be a good subject for future works.

7 Examples

To illustrate and justify the notions that are presented in Section 2, we shall now present two examples of BANs that can be partially understood by cutting them into modules. The first example is a toy BAN illustrated in Figure 6. In this representation we assume the function of each automaton to be a disjunctive clause with one literal for each incident edge, the sign of which dictates the sign of the literal.

Looking at this example, it does not seem easy to express the entire behaviour of the BAN F. Its representation is a strongly connected graph with multiple interconnected positive and negative cycles. Yet, cutting this graph into multiple modules and analysing the functionality of each of them is an easy way to understand interesting parts of the dynamics of the network.

By assuming the decomposition of F as shown in Figure 6, we can start to attach a functionality to each module. Module M_1 is a positive cycle, where the configuration $x_a = x_d = 1$ is a fixed point (whatever the input). Its functionality can be identified as a "one time button" that cannot be pushed back. Module M_2 is a negative cycle, which are known for their long limit cycles. The difference here is that as M_2 has two inputs, its behaviour can be stabilised into a fixed point by a fixed input. For example, the fixed point $x_b = x_e = 1, x_c = 0$ can be obtained with the constant input $i_b = 1, i_e = 0$. Finally, the module M_3 is acyclic and thus only computes the Boolean function $\neg i_g \vee (\neg i_h \wedge i_{h'})$. It follows that M_3 stabilises to a fixed point under any constant input.

This simple analysis leads us to the following conclusion: every fair execution (meaning executing every automaton an infinite amount of time) of F which verifies $x_a = x_d = 1$ at any moment stabilises into a fixed point. This is true because $x_a = x_d = 1$ implies that the "one time button" of M_1 is pushed in, which locks the behaviour of M_2 into a fixed point, which leads M_3 to compute a Boolean function over a fixed input. This somewhat informal demonstration has led us to a conclusion that was not easily implied by the architecture of the network, showcasing the usefulness of understanding networks as composition of parts to which one can assign functionalities.

The second example is drawn from a model predicting the cell cycle sequence of fission yeast [6]. This network is represented in Figure 7, and can be decomposed into a more abstract network, where each node represents a module of the original network. This network is represented in Figure 8 and its modules are constructed as follows: $C = \{\mathrm{Rum1}, \mathrm{Ste9}\}, D = \{\mathrm{Cdc}, \mathrm{Cdc}*\}, \mathrm{F} = \{\mathrm{Cdc25}\}, G = \{\mathrm{Mik}\}, \mathrm{I} = \{\mathrm{Start}, \mathrm{SK}\}, J = \{\mathrm{PP}, \mathrm{Slp1}\}$. A quick analysis of these modules leads us to sort them into three categories: cycles (C, D), functions (F, G) and igniters (I, J). Let us now explain this organisation in an informal way.

The two cycle modules C and D are organised in a 4-cycle of negative feedback which means that if considered separately from the rest of the network, those two modules would behave as antagonists: in most cases, when the automata of C (resp. D) are evaluated to 1, the automata of D (resp. C) will be evaluated to 0. Modules F and G can be viewed as functions which help D and C respectively to be evaluated to 1; they both are influenced by J in different ways. Modules I and J are called igniters because they turn themselves to 0 every time they are evaluated to 1, but not before influencing the other nodes. Module I inhibits C when activated, and can be considered as the input of the whole network. Module J is activated by D, activates C and G, and inhibits F.

From this we can conclude that if the network stabilises, it will more likely stabilise by evaluating C to 1 and D to 0. This conclusion arises from the fact that D activates J, which in turn inhibits D directly, but also inhibits F (which

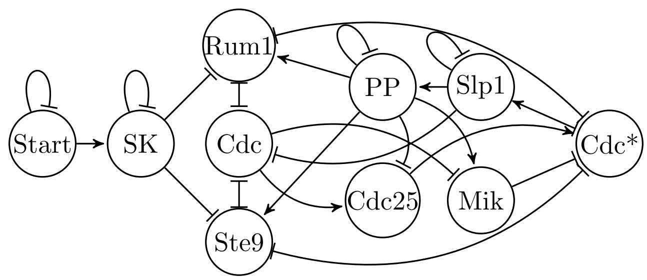

Figure 7: Representation of the network simulating the cell cycle sequence of fission yeast extracted from [6]. Activating interactions are represented by simple arrows and inhibiting interactions by flat arrows. The detail of each node's function is available in the original paper.

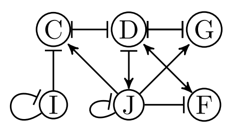

Figure 8: Abstract representation of the interactions between the modules C, D, F, G, I and J based upon the network represented in Figure 7.

activates D) and activates G (which inhibits D). This also means that F will be evaluated to 0 and G to 1. Finally, I and J will naturally be evaluated to 0 because of the natural negative feedback that compose them. This particular evaluation of the network (only C and G to 1) is actually the main fixed point of the network's dynamics put forward in [6] and is named $G1$. This shows that such a fixed point can be described without the need to compute the $2^{10} = 1024$ different configurations of the network and their dynamics.

8 Algebraic exploration of the dynamics of modules

This section takes insipiration into the work of [9] and explores the idea of an algebraic representation of a module's dynamics. Our aim is to start a characterisation of the effects of wirings upon the module's dynamics using algebra.

The dynamics of a module are meant to represent the relation between every possible state of the system in the form of a graph. The original paper [9] expresses such a representation as a couple (D, f), with D a set of states and f the next-state function which maps each state to the next one. In our approach, we will describe the dynamics of a module $M = (S, E, \alpha)$ as a couple (D, R), with D the set of all configurations $x : S \to \mathbb{B}$, and $R \subseteq D^2$ a relation such that xRx' if and only if there exists an update δ and an input i such that $M_\delta(x, i) = x'$. Starting from any configuration, the behaviour of a module for which the inputs or update mode is not choosen ahead of time is naturally non deterministic. The obtained graph over a BAN corresponds to what is commonly called a General Transition Graph.

The multiplication operator on modules' dynamics is directly taken out from the original paper [9] and reads as follow:

$$(D, R) \times (D', R') = (D \times D', R \times R')$$
$$\text{where } (x, x')(R \times R')(y, y') \Leftrightarrow (xRy \text{ and } x'R'y').$$

Considering M, M' two modules with (D, R) and (D', R') as their respective dynamics, it is easy to see that the module $M \rightarrowtail_\varnothing M'$ has $(D, R)\times(D', R')$ as its dynamics. Furthermore, the special module $M = (S = \varnothing, E = \varnothing, \alpha : \varnothing \to \varnothing)$ has dynamics $(D = \{a\}, R)$ such that aRa, which is the neutral element of the multiplication operator described above. Considering these facts we can clearly express that the category of all dynamics generated by modules is a commutative monoid over the multiplication operator. It is also a submonoid of the commutative monoid $(D, \times)$ from [9].

This monoid of dynamical systems generated from modules can be shown to have no zero like element, as the empty dynamical system cannot be generated from any module. This can clearly be checked as the size of the dynamics of a module is always of the form 2^n, where n is the size of the module.

As this paper describes a type of composition of modules in which the empty wiring is only a very specific case, let us look at some possible characterisation of wirings. As any wiring $M \rightarrowtail_\omega M'$ can always be rewritten into the form $\circlearrowleft_{\omega'} (M \rightarrowtail_\varnothing M')$, and as the algebraic effect of the empty non-recursive wiring upon the dynamics of the modules has already been characterised, we shall only discuss the algebraic effect of wirings of the recursive type.

As it is the purpose of a recursive wiring to affect an input to the value of an automaton, it is understandable that the effect of that wiring on the algebraic dynamics (D, R) would be to remove some pairs in R. To take a basic example, the module composed of only one function $f_a(x_a, i_e) = i_e$ defines a couple (D, R) where R is the relation containing every possible pair of configurations over $\{a\} \to \mathbb{B}$, *i.e.* $0R0$, $0R1$, $1R0$ and $1R1$. Applying the recursive wiring $\omega(e) = a$, we obtain the BAN containing only the function $f_a(x_a) = x_a$. The couple (D', R') of this new system would be such that $0R'0$ and $1R'1$ only. This monotonicity of recursive wirings is formalised in the following proposition.

Proposition 2. *Let $M = (S, E, \alpha)$ and $M' = (S, E', \alpha')$ be modules, with (D, R) and (D', R') their respective dynamics. If there exists a recursive wiring ω over M such that $\circlearrowleft_\omega M = M'$, then $R \supseteq R'$.*

Proof. Let us take $x, y \in D$ such that $xR'y$. This implies that there exists an input i' and an update mode δ such that $M'_\delta(x, i') = y$. Let us take an input i over M such that $i\big|_{E'} = i'$, and for all $e \in E \setminus E'$, $i(e) = \omega(e)(x)$. This way, i

reproduces the behaviour of the wiring ω. It follows that $M_\delta(x, i) = y$, which means that xRy.

□

This somewhat simple proposition offers the interesting insight that specifying the behaviour of the inputs of the network will always lead to simpler dynamics. This idea has been hinted at in the past [7, 14], and we propose here a formalisation of it in a broader context, where inputs can exist. In this interpretation those inputs could represent parameters external to the system, or unknown factors.

This proposition allows us to describe the evolution of the dynamics of a network during its creation. The first stage consists of uniting the different necessary parts of the network, and the second consists in the recursive wiring of every input that is planned to connect those parts together. Keeping those two stages in mind, we can see the evolution of the dynamics itself has two separate monotonous phases: the first one sees the dynamics of the network explode exponentialy in size, with an increasing complexity of connections between the possible states depending on the number of inputs added to the interaction graph. The second phase is also monotonous, as the number of configurations stays constant, and sees the number of edges decrease, as some of the inputs left over from the first phase are wired to the network.

This process of input wiring is key in understanding the evolution of the dynamics. In a practical sense, it means that wiring an input fixes it's value to the value of a part of the network. Understanding the influence of wirings on the dynamics of networks would be a very powerful tool in practical cases. To that effect we express a decision problem on the existence of a one input wiring that would remove a target edge in the graph dynamics of a module. We shall call this decision problem the Reductive Wiring Existence problem.

Reductive Wiring Existence problem
Instance : A module $M = (S, E, \alpha)$, and x, x' such that there exists δ, i, such that $M_\delta(x, i) = x'$.
Question : Does there exist ω a recursive wiring over M wiring exactly one input ($|\mathrm{dom}(\omega)| = 1$)
such that for all δ, i we have $\circlearrowright_\omega M_\delta(x, i) \neq x'$?

The aim of such a problem is to mesure the practicability of this approach, as well as allowing for a more precise characterisation of the evolution of the dynamics under wirings. The complexity of the above problem actually comes from the complexity of the Reductive Wiring problem.

Reductive Wiring problem
Instance : A module $M = (S, E, \alpha)$, and x, x' such that there exists δ, i, such that $M_\delta(x, i) = x'$,
and ω a recursive wiring over M.
Question : For all δ, i do we have $\circlearrowright_\omega M_\delta(x, i) \neq x'$?

Theorem 3. *The Reductive Wiring problem is* co-NP *complete.*

Proof. We will first prove that this problem is in co-NP, and then prove that it is co-NP hard.

Let us call RW the langage of all positive instances of the Reductive Wiring problem. Let $a = (M, x, x', \omega)$ be a positive or negative instance of the problem. The Reductive Wiring is in co-NP if and only if there exists a langage B in P and a polynomial $p(n)$ such that

$$a \in \mathrm{RW} \Leftrightarrow \forall b \in \{0,1\}^{\leq p(|a|)}, (a, b) \in B.$$

In the context of the Reductive Wiring Existence problem, b is defined as a pair containing an input configuration i and an update δ on M. We shall define that $(a, b) \in B$ if and only if b does not correctly encode a pair as stated above, or if $\circlearrowright_\omega M_\delta(x, i) \neq x'$, with $b = (i, \delta)$. It is clear that $B \in \mathrm{P}$ since the computation of $\circlearrowright_\omega M_\delta(x, i) \neq x'$ is only the computation of every local function in M, after the application of the wiring ω. We can also see that if $a \in \mathrm{RW}$, then for every pair (i, δ) defined as above, $(a, (i, \delta)) \in B$. It is also true that if there exists b such that $(a, b) \notin B$ then $\circlearrowright_\omega M_\delta(x, i) = x'$ with $b = (i, \delta)$, then the instance a verifies $a \notin \mathrm{RW}$. Finally, every b is in polynomial size of a since it is a map that for each of the $|S| \times |E|$ possible wirings, assigns an input configuration of size $|E|$ and an update of size $|S|$. We conclude that the Reductive Wiring problem is in co-NP.

Let us now prove that it is co-NP hard. The problem that we will reduce from is the Boolean unstatisfiability problem, or co-SAT. For any instance $\phi \in$ co-SAT with n different variables, let us construct a module M such that $S = \{s\}$, $E = \{e_1, \ldots, e_n, e_{n+1}\}$, and α such that $\alpha(e_k) = s$ for $k \leq n + 1$. The local function of the node s will be the following: let us note ψ the formula of the instance ϕ in which every instance of the variable x_k for $k \leq n$ is substituted for the input variable e_k. We then define $f_s(x, i) = i(e_{n+1}) \vee \psi(i)$, where $\psi(i)$ is the formula ψ evaluated according to i. The edge x, x' that we aim to remove in the graph of the dynamics of M is the one such that $x(s) = 0$ and $x'(s) = 1$. This edge always exists before any wiring, as setting the input e_{n+1} to 1 will always influence the value of the node s from 0 to 1. Finally, we define the recursive wiring ω as the wiring which only connects the input e_{n+1} to the node s.

Let us prove that the answer to the Reductive Wiring problem for this instance is equivalent to the unsatisfiability of ϕ. Note that only two updates $\delta = \{s\}$ and $\delta' = \varnothing$ are possible, and since the latter will never lead to any evolution of the configuration, we will use the update $\delta = \{s\}$ for the rest of the demonstration. In $\circlearrowright_\omega M$ and under the configuration $x(s) = 0$, the local function $f_s(x, i)$ is equivalent to the computation of $0 \vee \psi(i)$, which is equivalent to $\psi(i)$. As we can see, ϕ is positive in co-SAT if and only if the instance M, x, x' is positive in the Reductive Wiring Existence problem.

Finally, it is clear to see that the above proposed reduction is polynomial, as the construction itself is defined with $|S| = 1$ and $|E| = n + 1$. $\square$

Corollary 4. *The Reductive Wiring Existence problem is* co-NP *complete.*

Proof. To prove that this problem is in co-NP, we only have to see that the number of wirings ω with $|\mathrm{dom}(\omega)| = 1$ is $|S| \times |E|$. Knowing this, we can resolve this problem by resolving the Reductive Wiring problem on the same instance a polynomial amount of times.

To prove that it is co-NP hard, let us take an instance $\phi \in$ co-SAT and take the same exact construction as in the last demonstration, only here the final construction does not include a specific wiring ω to test. To show that it is nonetheless equivalent, we have to see that between all the possible $n+1$ wirings, n of them leave the input e_{n+1} unconnected, which gives a trivial positive solution to the resolution of the local function f_s. The only non trivial wiring is therefore the same as in the last demonstration, which leads to the equivalent result that the Reductive Wiring Existence problem is co-NP hard, and therefore co-NP complete. □

These results hint that the prediction of the evolution of the dynamics of a module upon wiring is a costly process. However it is interesting to note that the above mentionned complexity scales on the number of inputs of the studied modules. The co-NP completeness arises here from the exploration of all possible input configurations and possible updates: in a case where the number of inputs is logarithmic on the module size and where the update mode is restricted to a family of polynomial size, the problem would be solvable in polynomial time by a brute force algorithm. In particular, the prediction of the evolution of the dynamics of a module, when this module evolves under the parallel udpate mode, with a low number of inputs, would be easier to predict. It is however important to note that fixing the update mode of the network only does not reduce the complexity of the problem, as characterised in the following corollary.

Corollary 5. *The Reductive Wiring Existence problem, restricted to the parallel update mode, is co-NP complete.*

Proof. This corollary naturally follows from the fact that the demonstration of the Corollary 4 can be applied as long as the update mode containing only the node s is allowed. □

9 Conclusion

The three theorems formulated in this article tell us that seeing BANs as modular entities is a way to discover useful results. With the simple addition of inputs to BANs, we have expressed a general simulation structure that can be used to understand the computational nature and limits of given properties over BANs. We have also met simple but instructive results when envisioning the dynamics of modules as algebraic structures. Let us underline that all the definitions and results, except ones related to complexity classes, can be applied to BANs and modules defined over countably infinite sets of automata and inputs.

Wherever Turing-completeness is observed, complex behaviours emerge that cannot be simply or quickly formulated from the basic rules of the computation.

In such situations, the solution is either to compute every single possibility to capture the whole dynamics of the observed system, or to simplify the model. We believe that the framework developed in this paper is a strong candidate to enable us to decompose complex networks into parts with tractable functionalities, and to make conclusions about the whole network at a cheaper cost.

Adding inputs to BANs is an interesting way of studying the evolution of a network upon modification, as it offers a formalisation of the idea of unknown parameters that influence the network. We believe that improving the understanding of such inputs shall improve our comprehension of the relation between a network and its dynamics.

Acknowledgements This work was funded mainly by our salaries as a French State agent and therefore by French taxpayers' taxes, and secondarily by the ANR-18-CE40-0002 FANs project, the ECOS-Sud C16E01 project, the STIC AmSud CoDANet 19-STIC-03 (Campus France 43478PD) project, and the *PACA Region* project 2015_01134 "Fri".

References

[1] A. Alcolei, K. Perrot, and S. Sené. On the flora of asynchronous locally non-monotonic Boolean automata networks. In *Proceedings of SASB'15*, volume 326 of *ENTCS*, pages 3–25, 2016.

[2] U. Alon. Biological networks: the tinkerer as an engineer. *Science*, 301:1866–1867, 2003.

[3] J. Aracena, Luis Gómez, and L. Salinas. Limit cycles and update digraphs in Boolean networks. *Discrete Applied Mathematics*, 161:1–12, 2013.

[4] G. Bernot and F. Tahi. Behaviour preservation of a biological regulatory network when embedded into a larger network. *Fundamenta Informaticae*, 91:463–485, 2009.

[5] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15:1–40, 2004.

[6] M.I. Davidich and S. Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLoS One*, 3:e1672, 2008.

[7] F. Delaplace, H. Klaudel, T. Melliti, and S. Sené. Analysis of modular organisation of interaction networks based on asymptotic dynamics. In *Proceedings of CMSB'12*, volume 7605 of *LNCS*, pages 148–165, 2012.

[8] J. Demongeot, E. Goles, M. Morvan, M. Noual, and S. Sené. Attraction basins as gauges of robustness against boundary conditions in biological complex systems. *PLoS One*, 5:e11793, 2010.

[9] A. Dennunzio, V. Dorigatti, E. Formenti, L. Manzoni, and A. E. Porreca. Polynomial equations over finite, discrete-time dynamical systems. In *Cellular Automata*, pages 298–306, Cham, 2018. Springer International Publishing.

[10] T. Feder. *Stable networks and product graphs.* PhD thesis, Stanford University, 1990.

[11] E. Goles and L. Salinas. Comparison between parallel and serial dynamics of Boolean networks. *Theoretical Computer Science*, 396:247–253, 2008.

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

[13] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[14] M. Noual. *Updating Automata Networks*. PhD thesis, École Normale Supérieure de Lyon, 2012.

[15] K. Perrot, P. Perrotin, and S. Sené. A framework for (de)composing with Boolean automata networks. In *Proceedings of MCU'18*, volume 10881 of *LNCS*, pages 121–136, 2018.

[16] H. Siebert. Dynamical and structural modularity of discrete regulatory networks. In *Proceedings of COMPMOD'09*, volume 6 of *EPTCS*, pages 109–124, 2009.

[17] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1973.