



HAL
open science

A framework for composing with Boolean automata networks

Kévin Perrot, Pacôme Perrotin, Sylvain Sené

► **To cite this version:**

Kévin Perrot, Pacôme Perrotin, Sylvain Sené. A framework for composing with Boolean automata networks. 2017. hal-01654221v1

HAL Id: hal-01654221

<https://hal.science/hal-01654221v1>

Preprint submitted on 3 Dec 2017 (v1), last revised 10 Apr 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A framework for composing with Boolean automata networks

Kévin Perrot¹, Pacôme Perrotin^{*1}, and Sylvain Sené¹

Université d'Aix-Marseille, CNRS, LIF, Marseille, France

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 two results: the first one shows that our modules/wirings definition is complete; the second one uses modules/wirings to prove simulation results amongst BANs.

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

1 Introduction

Boolean automata networks (BANs) can be described 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 [6,10,15] in biology.

Many efforts to characterise the dynamics of BANs have already been put forward. For example, some studies [1,12] 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 [2,11], and a functional one [4,5,14]. In this paper, we explore a compositional approach to BANs that allows to decompose a BAN into subnetworks called modules, and to

* Corresponding author: pacome.perrotin@lif.univ-mrs.fr.

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 [7], and modules can be seen as a generalisation of circuits over any update mode.

Section 2 introduces BANs and update modes. Sections 3 and 4 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 5, which leads to a second theorem stating that composing local simulations is sufficient to simulate a BAN.

2 Boolean automata networks

2.1 Preliminary notations

For the sake of clarity, let us first describe some of the notations used throughout the paper. Let $f : A \rightarrow B$ 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 $f|_S$ the restriction of f to the domain S , $f|_S : S \rightarrow B$ such that $f|_S(a) = f(a)$. Let $\text{dom}(f)$ 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 :

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

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 $K_{[i,j]}$.

2.2 Definitions

A BAN is based upon a set of automata. Each automaton is defined as a Boolean function, with arity the size of the network. Each variable of the function of each automaton is meant to correspond to an automaton in the network. By considering a configuration of Boolean values over this network, we can compute the Boolean function of each automaton and obtain a Boolean value for each automaton (*i.e.* a local state). These values can be used to update the global state of the network, that we call a configuration. If we decide to update the value of each automaton at once, the update mode parallel. Otherwise, if only one automaton is updated at each time step, the update mode is sequential [8,13].

Definition 1. *For S a set, a Boolean automata network (BAN) F is a function from the set S to the set of all the functions of arity $|S|$ from the set of all the configurations over S to the set \mathbb{B} .*

For each $s \in S$, we denote $f_s = F(s)$ the local function of automaton s . A configuration is itself a function as defined below.

Definition 2. *A configuration on a set S is a function $x : S \rightarrow \mathbb{B}$.*

For $s \in S$ we denote $x_s = x(s)$. A function x is a configuration at a given time over the network. Thus, we can define our function f_s to be part of the set $(S \rightarrow \mathbb{B}) \rightarrow \mathbb{B}$. This way, a BAN F can be defined as a function from the set S to the set $(S \rightarrow \mathbb{B}) \rightarrow \mathbb{B}$. To go a step further, the set of all BANs over a given set S is $S \rightarrow ((S \rightarrow \mathbb{B}) \rightarrow \mathbb{B})$. For any BAN F and configuration x , we can define the configuration which is computed by F from x . A naive way to do so would be to define $x' = F(x)$ such that $x'_s = f_s(x)$ for every s ; this definition however is very limiting: it only allows parallel updates of our system. In a general definition of BANs, a computation of a BAN should allow updates of only a subset of the functions of the network. Slight changes to the update mode of a BAN can deeply change its computational capabilities [3,9]. Most results that assume a parallel update mode cannot be applied to a sequential network; the reciprocal is also true. We set the following definition of an update over our BAN to be as general as possible.

Definition 3. *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. We define now the union operator over such updates.

Definition 4. *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 $\Delta_k = \emptyset$ 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 5. *The execution of F over x , under the update mode Δ , is an endomorphism over the set of all configurations, denoted F_Δ , and defined as $F_{\Delta[1,k]}(x) = F_{\Delta_k}(F_{\Delta[1,k-1]}(x))$, with $F_{\Delta[1,1]}(x) = F_{\Delta_1}(x)$.*

3 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

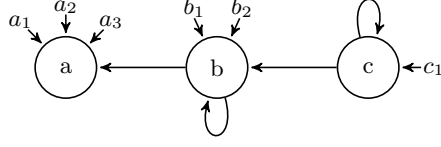


Fig. 1. Interaction graph of the module detailed in Example 1.

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 6. Let S and E be two distinct sets. An input declaration over S and E is a function $\alpha : S \rightarrow \mathcal{P}(E)$ 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 property makes all inputs linked to an automaton of S so that no input is connected to two distinct automata. To simplify notations, we sometimes denote $E_s = \alpha(s)$. Now, let us explicit the concept of a module.

Definition 7. A module M over (S, E, α) is a function such that $M(s)$ is a function from the set of all configurations over $S \cup E_s$ to the set \mathbb{B} .

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

Definition 8. An input configuration over E is a function $i : E \rightarrow \mathbb{B}$.

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 9. 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 $M_\delta(x \sqcup i)(s) = f_s(x \sqcup i|_{E_s})$ 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 Boolean 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 $M_\delta(x \sqcup i)(a) = f_a(x \sqcup i|_{E_a}) = 0$, $M_\delta(x \sqcup i)(b) = f_b(x \sqcup i|_{E_b}) = 1$, 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 10. Let $t > 1$. Let $I = (i_1, i_2, \dots, i_{t-1})$ be a sequence of input configurations over E , $X = (x_1, x_2, \dots, x_t)$ 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 can raise concerns about the computational power of modules as models of computation. We can indeed force a trivial module to compute any infinite Boolean sequence, even non-periodic ones, simply by encoding them in the sequence of input configurations. For this reason, and because we do not need such computational power to obtain the results presented thereafter, we will limit the notion of an execution to constant input configurations over time. Such a limitation is expressed by the following recursive definition.

Definition 11. Let i be an input configuration over E . The execution of M over $x \cup i$ with update mode Δ is an endomorphism over the set of all configurations, denoted M_Δ . It is defined as $M_{\Delta[1,k]}(x \cup i) = M_{\Delta_k}(M_{\Delta[1,k-1]}(x \cup i) \cup i)$, with $M_{\Delta[1,1]}(x \cup i) = M_{\Delta_1}(x \cup i)$.

4 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 β linking some inputs to automata. Let us first define non-recursive wirings.

Definition 12. 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 distinct. A non-recursive wiring from M to M' is a partial map β from E' to S .

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

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

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 β . For any $s \in S \cup S'$, the function $M \mapsto_{\beta} M'(s)$ (denoted f_s^{β}) is defined over $(S \cup S' \cup \alpha_{\beta}(s)) \rightarrow \mathbb{B}$. 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 $E'_s \setminus \text{dom}(\beta)$), and partly provided by $(x \circ \beta)$ (on $\text{dom}(\beta) \cap E'_s$).

Definition 13. *Let M be a module over (S, E) . A recursive wiring of M is a partial map β from E to S .*

With β 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 β on M is denoted $\circlearrowleft_{\beta} M$ and is defined over $(S, E \setminus \text{dom}(\beta), \alpha_{\beta})$ with the input declaration defined as, for any $s \in S$, $\alpha_{\beta}(s) = \alpha(s) \setminus \text{dom}(\beta)$. Given $s \in S$, x and i , the local function $\circlearrowleft_{\beta} M(s)$ is denoted f_s^{β} and is evaluated to $f_s^{\beta}(x \sqcup i) = f_s(x \sqcup i|_{E_s \setminus \text{dom}(\beta)} \sqcup (x \circ \beta|_{E_s}))$.

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

Property 1. The following statements hold.

- (i) $\forall M, \quad \circlearrowleft_{\emptyset} M = M.$
- (ii) $\forall M, M', \quad M \mapsto_{\emptyset} M' = M' \mapsto_{\emptyset} M.$
- (iii) $\forall M, M', M'', \quad M \mapsto_{\emptyset} (M' \mapsto_{\emptyset} M'') = (M \mapsto_{\emptyset} M') \mapsto_{\emptyset} M''.$

Proof.

$$(i) \quad \forall M, M', M \mapsto_{\emptyset} M' = M' \mapsto_{\emptyset} M.$$

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

$$(ii) \quad \forall M, \quad \circlearrowleft_{\emptyset} M = M.$$

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

$$(iii) \quad \forall M, M', M'', M \mapsto_{\emptyset} (M' \mapsto_{\emptyset} M'') = (M \mapsto_{\emptyset} M') \mapsto_{\emptyset} M''.$$

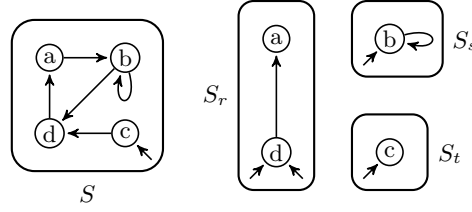


Fig. 2. Interaction graphs related to Example 2. The interaction graph of the original module is on the left and the interaction graphs of the partition of M are on the right. Notice that we did not represent the input sets E , Q_r , Q_s and Q_t .

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: $M \cup M' = M \mapsto_{\emptyset} M'$.

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 14. Let (S, E, α) . Let P be a set such that $|P| \leq |S|$, and let $\{S_p \mid p \in P\}$ be a partition of S . We define the corresponding partition of E as $\{E_p = \bigcup_{s \in S_p} \alpha(s) \mid p \in P\}$.

We can develop the corresponding partition of the input declaration.

Definition 15. For every $p \in P$, we define $\alpha_p = \alpha|_{S_p}$ over S_p and E_p .

Let us now define the partition of M itself.

Definition 16. For every $p \in P$, let Q_p verify $Q_p \cap S = \emptyset$ and $|Q_p| = |S|$, and let $\tau_p : S \rightarrow Q_p$ 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, $\forall x : S \rightarrow \mathbb{B}$ and $\forall i : E \rightarrow \mathbb{B}$,

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

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 distinct 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 , $\alpha_r(a) = \emptyset$ 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 distinct 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 β such that $M = \circlearrowleft_\beta \left(\bigcup_{p \in P} M_p \right)$.*

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 $s \in S$, $x : S \rightarrow \mathbb{B}$ and $i : E \rightarrow \mathbb{B}$ verifies

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

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 β 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 $\beta(q) = \tau_p^{-1}(q)$ given p such that $q \in Q_p$.

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

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

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

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

Therefore for any s :

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

which concludes the proof. \square

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 \emptyset), 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}^β the closure of A by the set of wiring operators $\bigcup_\beta \{\rightarrow_\beta, \circlearrowleft_\beta\}$. 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,*

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

Proof. Trivially, $\overline{\mathcal{M}_1}^\beta \subseteq \mathcal{M}$. 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 $\mathcal{M} = \overline{\mathcal{M}_1}^\beta$. \square

In our opinion, this corollary is enough to demonstrate that our definition of modules and wirings is sound.

5 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 with 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 property of simulating any BAN can be reduced in some cases to the property of locally simulating any Boolean function. Locally simulating a function means that a module reproduces any computation of that function, given that the parameters of the function are encoded in the module's 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.

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 $U_{a,b}$. By definition we always have $\phi_{a,b}(x|_{U_{a,b}}) = \phi_a(x|_{T_a})$. This means that if a module encodes an information, the same information is sent from that module to the other modules.

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 $I_{a,b} : E_b \rightarrow U_{a,b}$ which we call interface between a and b . By definition:

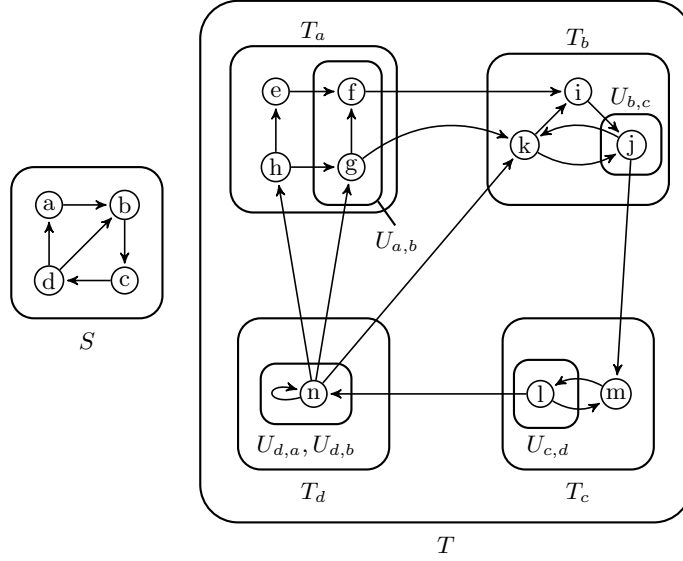


Fig. 3. Interaction graphs of the modules detailed in Example 3. The interaction graph of the original BAN is on the left and the interaction graph of the simulating BAN is on the right. The simulating BAN is decomposed into four sub-modules, one for each node in S . Notice that we did not represent 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.

- 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).

Let us now give the definitions mentioned in the previous explanation.

Definition 17. Let A be a set. A Boolean encoding over A is a function $\phi : (A \rightarrow \mathbb{B}) \rightarrow (\{0, 1, \bullet\})$, such that there exists at least one x such that $\phi(x) = 0$ and one x such that $\phi(x) = 1$.

For $x : A \rightarrow \mathbb{B}$, $\phi(x) = 1$ 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 $U_{a,b}$.

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 $x : S \rightarrow \mathbb{B}$, all the configurations $x' : T_a \rightarrow \mathbb{B}$ and input configurations $i' : E_a \rightarrow \mathbb{B}$ 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: $\phi_a(x') = x_a$. To express that i' encodes the state of all $b \neq a$ in x requires an additional notation. On the one hand we have $\phi_{b,a} : (U_{b,a} \rightarrow \mathbb{B}) \rightarrow (\{0, 1, \bullet\})$, and on the other hand we have $i' : E_a \rightarrow \mathbb{B}$ describing the input-configuration of module M_a , and $I_{b,a} : E_a \rightarrow U_{b,a}$ 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 $i' \circ I_{b,a}^{-1}$ the configuration over $U_{b,a}$ such that $i' \circ I_{b,a}^{-1}(s) = i'(e)$ for any e such that $I_{b,a}(e) = s$. By our hypothesis this configuration is well defined.

Definition 18. 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 $M_a \sim_{\Delta} f_a$, if for all $x : S \rightarrow \mathbb{B}$,

1. and for all $x' : T_a \rightarrow \mathbb{B}$ such that $\phi_a(x') = x_a$,
2. and for all $i' : E_a \rightarrow \mathbb{B}$ such that for all $b \neq a$ we have $\phi_{b,a}(i' \circ I_{b,a}^{-1}) = x_b$,
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 19. 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 $\alpha(s) = \emptyset$.

Definition 20. We define M able to input-first simulate f if there exists an input-first Δ such that $M \sim_{\Delta} f$.

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 $\Phi : (S \rightarrow \mathbb{B}) \rightarrow (S' \rightarrow \mathbb{B}) \cup \{\bullet\}$ which always verifies that for all $x' : S' \rightarrow \mathbb{B}$, there exists $x : S \rightarrow \mathbb{B}$ such that $\Phi(x) = x'$.

Definition 21. *Let F and F' be two Boolean automata networks over S and S' respectively. We define that F simulates F' , denoted by $F \sim F'$, 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) , alongside 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 β over $T = \bigcup_{a \in S} T_a$ such that*

$$\circlearrowleft_\beta \left(\bigcup_{a \in S} M_a \right) \sim F.$$

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 $\beta = \bigcup_{a,b \in S, a \neq b} I_{a,b}$. By definition of $I_{a,b}$, we can easily see that the module $M = \circlearrowleft_\beta (\bigcup_{a \in S} M_a)$ is defined over $(T, \emptyset, s \mapsto \emptyset)$ 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 $x : T \rightarrow \mathbb{B}$, the following equation holds:

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

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

Let us define $A = \{s \in T_a \mid \alpha(s) \neq \emptyset\}$ 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:

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

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 $M_{\delta \cup \delta'}(x)|_A = M_{\Delta \cup \Delta'}(x)|_A$, and that $M_{a\delta}(x|_{T_a} \sqcup (x \circ \sqcup_b I_{b,a}))|_A = M_{a\Delta}(x|_{T_a} \sqcup (x \circ \sqcup_b I_{b,a}))|_A$. In conclusion of this A part, $M_{\Delta \cup \Delta'}(x)|_A = M_{a\Delta}(x|_{T_a} \sqcup (x \circ \sqcup_b I_{b,a}))|_A$.

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

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

Let us now define $\Phi : (T \rightarrow \mathbb{B}) \rightarrow (S \rightarrow \mathbb{B}) \cup \{\emptyset\}$ such that, for any $x : T \rightarrow \mathbb{B}$, $\Phi(x) = \emptyset$ if there exists $a \in S$ such that $\phi_a(x|_{T_a}) = \emptyset$, and $\Phi(x)(a) = \phi_a(x|_{T_a})$ otherwise. Let x and x' such that $\Phi(x) = x'$, and $x' \neq \emptyset$. 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 $M_\Delta(x) = \bigsqcup \{M_\Delta(x)|_{T_a} \mid a \in S\}$, which can be developed into $M_\Delta(x) = \bigsqcup \{M_\Delta(x)|_{T_a} \mid a \in \delta\} \sqcup \bigsqcup \{x|_{T_a} \mid a \in S \setminus \delta\}$, from which we infer:

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

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

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

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 :

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

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

the local simulation definition and obtain:

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

In the other hand, 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. \square

This theorem can help us investigate if every BAN can be simulated by a BAN with a given property. If every function f can be locally simulated by a given module with a property \mathcal{P} , and if property \mathcal{P} is preserved over wirings, then we know that any BAN can be simulated by another BAN with the property \mathcal{P} . This is observed for example when \mathcal{P} corresponds to restricting each function to be a disjunctive clause, or when it corresponds to the monotony of every function in the module or BAN.

6 Conclusion

The two 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. Let us underline that all the definitions and results can be applied to BANs and modules defined over countably infinite sets of automata and inputs. This modular strategy of considering BANs is, in our opinion, a good way to understand and characterise their dynamics. The dynamical complexity arises when the network is too large, making the number of possible configurations impractical. Cutting the network into smaller pieces (partitioning) is a promising way to understand it at a cheaper cost.

Alongside simulation, a number of different paths can be taken to develop modules. We are now interested in characterising a module with a language, the same way that the language of a finite automaton characterises it. For modules, the considered language would be the set of all executions that can be obtained given the set of all possible inputs and the set of all possible update modes on that module. Understanding how the languages of two given modules unite when they are wired together would be a strong tool to characterise the language of any BAN, namely its dynamics.

References

1. A. Alcolei, K. Perrot, and S. Sené. On the flora of asynchronous locally non-monotonic Boolean automata networks. In *Proc. 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 Appl. Math.*, 161:1–12, 2013.
4. G. Bernot and F. Tahi. Behaviour preservation of a biological regulatory network when embedded into a larger network. *Fund. Inform.*, 91:463–485, 2009.
5. F. Delaplace, H. Klaudel, T. Melliti, and S. Sené. Analysis of modular organisation of interaction networks based on asymptotic dynamics. In *Proc. of CMSB'12*, volume 7605 of *LNCS*, pages 148–165, 2012.
6. 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.
7. T. Feder. *Stable networks and product graphs*. PhD thesis, Stanford University, 1990.
8. F. Fogelman, E. Goles, and G. Weisbuch. Transient length in sequential iteration of threshold functions. *Discrete Appl. Math.*, 6:95–98, 1983.
9. E. Goles and L. Salinas. Comparison between parallel and serial dynamics of Boolean networks. *Theor. Comput. Sci.*, 396:247–253, 2008.
10. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.
11. 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.
12. M. Noual. *Updating Automata Networks*. PhD thesis, cole normale suprieure de Lyon, 2012.
13. F. Robert. *Discrete iterations: a metric study*. Springer, 1986.
14. H. Siebert. Dynamical and structural modularity of discrete regulatory networks. In *Proc. of COMPMOD'09*, volume 6 of *EPTCS*, pages 109–124, 2009.
15. R. Thomas. Boolean formalization of genetic control circuits. *J. Theor. Biol.*, 42:563–585, 1973.