# Turning block-sequential automata networks into smaller parallel networks with isomorphic limit dynamics

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

[1] Université publique, Marseille, France
[2] Aix-Marseille Univ., Univ. de Toulon, CNRS, LIS, UMR 7020, Marseille, France

**Abstract.** We state an algorithm that, given an automata network and a block-sequential update schedule, produces an automata network of the same size or smaller with the same limit dynamics under the parallel update schedule. Then, we focus on the family of automata cycles which share a unique path of automata, called tangential cycles, and show that a restriction of our algorithm allows to reduce any instance of it into a smaller parallel network of the family and to characterize the number of reductions operated while conserving the limit dynamics. We also show that any tangential cycles reduced by our main algorithm is transformed into a network whose size is that of the largest cycle of the initial network. We end by showing that the restricted algorithm allows the direct characterization of block-sequential double cycles as parallel ones.

## 1 Introduction

Automata networks are classically used to model gene regulatory networks [9,16] [10,2,4]. In these applications the dynamics of automata networks help to understand how the biological systems might evolve. As such, there is motivation in improving our computation and characterisation of automata network dynamics. This problem is a difficult one to approach considering the vast diversity of network structures, local functions and update schedules that are studied. Rather than considering the problem in general, we look for families or properties which allow for simpler dynamics that we might be able to characterise [7,8].

We are interested in studying the limit dynamics of automata networks, that is, the limit cycles and fixed points that they adopt over time, notably since these asymptotic behaviors of the underlying dynamical systems may correspond to real biological phenomenologies such as the genetic expression patterns of cellular types, tissues, or paces. More precisely, we are less interested in the possible configurations themselves than in the information that is being transfered and computed in networks over time. As such, given families of networks, one of our objectives is to count the fixed points and limit cycles they possess.

In this paper, we provide an algorithm that, given an automata network and a block-sequential update schedule, produces an automata network of the

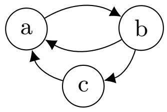



**Fig. 1.** Interaction digraph of the AN detailed in Example 1.

same size or smaller with isomorphic limit dynamics under the parallel update schedule. After definitions in Section 2, this algorithm is detailed in Section 3. In Section 4, the feasibility of the algorithm on certain types of ANs is studied. The demonstrations of all results are available in the appendix.

## 2 Definitions

Let $\Sigma$ be a finite alphabet. An *automata network (AN)* is a function $F : \Sigma^n \to \Sigma^n$, where $n$ is the size of the network. The global function $F$ can be divided into functions that are local to each automaton: $\forall k, f_k : \Sigma^n \to \Sigma$, and the global function can be redefined as the parallel application of every local function: $\forall 1 \leq i \leq n, F(x)_i = f_i(x)$. For convenience, the set of automata $\{1, \ldots, n\}$ is denoted $S$, and will sometimes be considered as a set of letters rather than numbers. For questions of complexity, we consider that *local functions are always encoded as circuits.*

For $(i, j)$ any pair of automata, $i$ is said to *influence* $j$ if and only if there exists a configuration $x \in \Sigma^n$ in which there exists a state change of $i$ that changes the state of $f_j(x)$. More formally, $i$ influences $j$ if and only if there exists $x, x' \in \Sigma^n$ such that $\forall k, x_k = x'_k \Leftrightarrow k \neq i$ and $f_j(x) \neq f_j(x')$.

It is common to represent an automata network $F$ as the digraph with its automata as nodes so that $(i, j)$ is an edge if and only if $i$ influences $j$. This digraph is called the *interaction digraph* and is denoted by $G_I(F) = (S, E)$, with $E$ the set of edges. The automata network described in Example 1 is illustrated as an interaction digraph in Figure 1.

*Example 1.* Let $F : \mathbb{B}^3 \to \mathbb{B}^3$ be an AN with local functions

$$f_a(x) = \neg x_b \vee x_c$$
$$f_b(x) = x_a$$
$$f_c(x) = \neg x_b$$

An *update schedule* is an infinite sequence of non-empty subsets of $S$, called blocks. Such a sequence describes in which order the local functions are to be applied to update the network, and there are uncountably infinitely many of them. A *periodic update schedule* is an infinite periodic sequence of non-empty subsets of $S$, which we directly define by its period. The application of an update schedule on a configuration of a network is the parallel application of the local functions of the subsets in the sequence, each subset being applied one after the other.

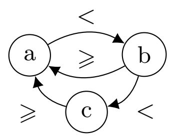



**Fig. 2.** Update digraph of the AN detailed in Example 1, for $\Delta = (\{a\}, \{b\}, \{c\})$.

For example, the sequence $\pi = (S)$ is the parallel update schedule. It is periodic, and its application on a configuration is undistinguishable from the application of $F$. The sequence $(\{1\}, \ldots, \{n\})$ is also a periodic update schedule, and implies the application of every local function in order, one at a time.

Formally, the application of a periodic update schedule $\Delta$ to a configuration $x \in \Sigma^n$ is denoted as the function $F_\Delta$, and is defined as the composition of the applications of each element of $\Delta$, in order. For any subset $X \subseteq S$, updating $X$ into $x$ is denoted by $F_X(x)$ and is defined as

$$\forall i \in S,\ F_X(x)_i = \begin{cases} f_i(x) & \text{if } i \in X \\ x_i & \text{otherwise} \end{cases}.$$

Example 2 provides an example of the execution of the network detailed in Example 1 under some non-trivial update schedule.

*Example 2.* Let $\Delta = (\{b, c\}, \{a\}, \{a, b\})$ be a periodic update schedule, and let $x = 000$ be an initial configuration. For $F$ the AN detailed in Example 1, we have that:

$$\begin{aligned} F_\Delta(000) &= (F_{\{a,b\}} \circ F_{\{a\}} \circ F_{\{b,c\}})(000) \\ &= (F_{\{a,b\}} \circ F_{\{a\}})(001) \\ &= F_{\{a,b\}}(101) = 111. \end{aligned}$$

A *block-sequential update schedule* is a periodic update schedule where all the subsets in the sequence form a partition of $S$; that is, every automaton is updated exactly once in the sequence. For any AN with automata $S$, both the parallel update schedule and the $|S|!$ sequential update schedules are block-sequential. Block-sequential update schedules are *fair* update schedules, in the sense that applying it updates each automaton the same amount of times.

The application of a block-sequential update schedule on an AN can be otherwise represented as an update digraph, introduced in [15,1]. For $F$ an AN and $\Delta$ a block-sequential update schedule, the *update digraph* of $F_\Delta$, denoted $G_U(F_\Delta)$, is an annotation of the network's interaction digraph, where any edge $(u, v)$ is annotated with $<$ if $u$ is updated strictly before $v$ in $\Delta$, and with $\geqslant$ otherwise. An update digraph of the AN detailed in Example 1 is illustrated in Figure 2.

Given an automata network $F$ and a periodic update schedule $\Delta$, we define the *dynamics* of $F_\Delta$ as the digraph with all configurations $x \in \Sigma^n$ as nodes, so that $(x, y)$ is an edge of the dynamics if and only if $F_\Delta(x) = y$. We call *limit*

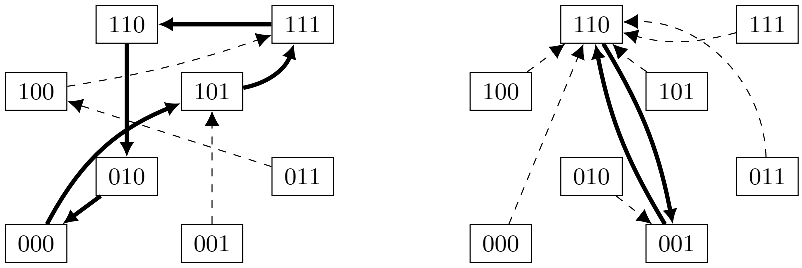



**Fig. 3.** Two dynamics of the AN $F$ detailed in Example 1. On the left, the dynamics of $F$ under the parallel update schedule. On the right, the dynamics of $F$ under the update schedule $\Delta = (\{a\}, \{b\}, \{c\})$. The limit dynamics are depicted with bold arrows.

*cycle of length* $k$ any sequence of unique configurations $(x_1, x_2, \ldots, x_k)$ such that $F_\Delta(x_i) = x_{i+1}$ for all $1 \leq i < k$, and $F_\Delta(x_k) = x_1$. A limit cycle of length one is called a *fixed point*. The *limit dynamics* of $F_\Delta$ is the subgraph which contains only the limit cycles and the fixed points of the dynamics. The limit dynamics of the network defined in Example 1 are emphasized in Figure 3.

Since the dynamics of a network is a graph that is exponential in size relative to the number of its automaton, naively computing the limit dynamics of a family of network is a computationally expensive process.

## 3 The algorithm

In this section, we look at an algorithm that can turn any automata network $F$ with a block-sequential update schedule $\Delta$ into another automata network $F'$, such that the limit dynamics of $F_\Delta$ stays isomorphic to the limit dynamics of $F'$ under the parallel update schedule $\pi$. Furthermore, the size of $F'$ will always be the size of $F$, *or less.*

This algorithm is built from two parts: first, we parallelize the network thanks to a known algorithm in the folklore of automata networks. Second, we remove automata from the networks based on redundancies created in the first step.

First, let us state the usual algorithm that, given an automata network $F$ and a block-sequential update schedule $\Delta$, provides a new automata network $F'$ defined on the same set of automata, such that $F_\Delta$ and $F'_\pi$ have the same exact dynamics.

Algorithm 1 proceeds with two waves of subsitutions. First, for every $<$-edge $(u, <, v)$, the influencing automaton $u$ is replaced in the local function of $v$ by a token symbol $\theta_u$. All of these token symbols are then replaced by the corresponding local functions (in this case, $f_u$) in the correct order: that is, no function is ever used in a substitution if it contains a token character. This way, even if the network contains a complex tree of $<$-edges, the substitutions will be applied in the correct order.

**Algorithm 1** Parallelization algorithm of $F_\Delta$

**Input**
- $F$ — local functions of a network over $S$, encoded as circuits
- $\Delta$ — update schedule over $S$
- $G_U(F_\Delta)$ — update digraph of $F_\Delta$

**Output**
- $F$ — local functions of a parallel network over $S$, encoded as circuits

```
for (u, <, v) ∈ E(G_U(F_Δ)) do
    apply the substitution x_u ↦ θ_u in f_v          ▷ θ is a temporary symbol
let X ← S
while |X| > 0 do
    let s ∈ X such that f_s contains no θ symbol
    X ← X \ {s}
    for s' ∈ X do
        if s' contains θ_s then
            apply the substitution θ_s ↦ f_s(x) in f_s'
return F
```

Let us prove for completeness that this algorithm always returns, and runs in polynomial time. The update digraph of $F_\Delta$ is considered given as part of the input.

*Property 1.* Algorithm 1 always returns, and does so in polynomial time.

*Sketch of proof.* There are no $<$-edge loop in the update digraph by definition, and so the algorithm always ends. Encoding local functions as circuits lets us do all needed substitutions in a straightforward way without increasing the size of the resulting circuits beyond the size of the input.

*Remark 1.* This algorithm is not polynomial if the local functions are encoded as formulæ, which is a detail often overlooked in the literature where this parallelisation algorithm is always assumed to be polynomial.

**Theorem 1.** *For any $F_\Delta$ Algorithm 1 returns a network $F'$ such that the dynamics of $F_\Delta$ is equal to that of $F'_\pi$.*

*Sketch of proof.* Substitution of the form $x_u \mapsto f_u(x)$ in the local function $f_v$ is equivalent to the presence of a $<$-edge $(u, <, v)$ in the update digraph of the network; in both cases, $v$ is updated using the next value of $x_u$ instead of the previous one. Altogether this means that both $F_\Delta$ and $F'_\pi$ are the same function.

Algorithm 2 is our contribution to this process, and removes automata that are not necessary for the limit dynamics of the network. It proceeds in two steps: first, the algorithm identifies pairs of automata with equivalent local functions, up to some function. In other terms, if one automaton $u$ can be computed as a function $g$ of the local function of another automaton $v$, then $u$ is not necessary and all references to $x_u$ in the network can be replaced by $g(x_v)$ for an identical

**Algorithm 2** Parallelization algorithm of $F_\Delta$, with a possible reduction in size

**Input**
- $F$ local functions of a network over $S$, encoded as circuits
- $\Delta$ update schedule over $S$
- $G_U(F_\Delta)$ update digraph of $F_\Delta$

**Output**
- $F'$ local functions of a parallel network over a subset of $S$, encoded as circuits

let $F' \leftarrow$ apply Algorithm 1 to $F_\Delta$, $G_U$
let $G_I(F') \leftarrow$ the interaction digraph of $F'$
**for** $(u, v) \in S^2$ **do**
  **if** $\forall x \in \Sigma^n, f_u(x) = g(f_v(x))$ **then** ▷ for some $g : \Sigma \to \Sigma$
    **for** $(u, w) \in E(G_I(F'))$ **do**
      apply the substitution $x_u \mapsto g(x_v)$ in $f_w$
**while** $\exists u \in S$ such that $u$ has no accessible neighbor in $G'_I$ **do**
  $S \leftarrow S \setminus \{u\}$ ▷ $u$ is removed from the network
return $F'$

result. Of course, this only works under the hypothesis that $u$ and $v$ are updated synchronously, which is the case after the application of Algorithm 1. Second, the algorithm iteratively removes any automaton that has no influence in the network, that is, that has no accessible neighbor in the interaction graph of the network. These automata are not part of cycles and do not lead to cycles, and as such have no impact on the attractors.

Algorithm 2 is non-deterministic, and when the local functions of any pair of automata $(u, v)$ are shown to be equivalent up to some reversible function $g : \Sigma \to \Sigma$, either automata could replace the influence of the other without preference. As such, more than one result network is possible, but all are equivalent in their limit dynamics, as will be shown later.

While it is clear that Algorithm 2 always terminates, its complexity is out of the deterministic polynomial range, as applying it implies solving the coNP-complete decision problem of testing if two Boolean formulæ are equal for all possible pairs of automata and for every possible function $g : \Sigma \to \Sigma$. As such, a polynomial implementation of this algorithm would (at least) imply P = NP. This drastic conclusion is softened when looking at restricted classes of networks where redundancies can be easily pointed out, which is the case for the rest of the paper.

**Theorem 2.** *For any $F_\Delta$, Algorithm 2 returns a network $F'$ such that the limit dynamics of $F_\Delta$ and $F'_\pi$ are isomorphic.*

*Sketch of proof.* Local transformations operated by the algorithm preserve the limit dynamics of the network, from which the result naturally follows.

# 4 Reductions in size of tangential cycles

In this section, we characterize the reduction in size that our algorithm provides on a specific family of networks. We call *tangential cycles* (TC) any AN composed of any number of cycles $\{C_1, C_2, \ldots, C_k\}$ such that every cycle shares a unique path of automata, called the *tangent*. The first automaton of the tangent is the only automaton with more than one in-neighbor, and is called the *central automaton*. Two TCs are represented as part of Figure 4, that contain three cycles and a tangent of length 0 (only one node is shared between the cycles).

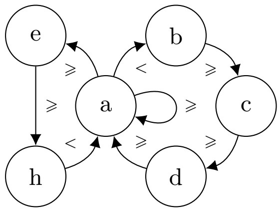
*Why focusing on TCs?* Cycles are fundamental retroactive patterns that are necessary to observe complex dynamics [14]. They are present in many biological regulation networks [17] and are perfectly understood in isolation [6,12]. In theory, cycles generate an exponential amount of limit cycles, which is incoherent with the observed behavior of biological systems [9]. The only way to reduce the amount of limit cycles is to constrain the degrees of freedom induced by isolated cycles, which can only be done by intersecting cycles from the purely structural standpoint. This leads us naturally to TCs, as a simple intersection case. Double cycles in particular are the largest family of intersecting cycles for which a complete caracterisation exists [12,5]; the present paper generalizes this result to block-sequential update schedules. Moreover, from the biological standpoint, double cycles are also observed in biological regulation networks, in which they seem to serve as inhibitors of their limit behavior [3].

## 4.1 Reducing block-sequential TCs

The reduction in size provided by Algorithm 2 can be quite large on TCs, as even parallel TCs can be reduced in size by merging the different cycles as much as possible. As such, the reduction power of this algorithm is greater than just removing the redundancies inherent to the block-sequential to parallel update translation. Indeed, Figure 4 provides an example of a parallel TC, the size of which is greatly reduced by the application of Algorithm 2. But, by this process, the final result of Algorithm 2 is no longer a TC.

As explained above, TCs are studied as the next simplest cases of complex ANs that make biological sense, after automata cycles. Both automata cycles and automata double cycles are examples of TCs. To show that the study of TCs under block-sequential update schedules can be directly reduced to the study of TCs under the parallel update schedule, we provide a last algorithm that transforms any TC under a block-sequential update schedule into a TC under the parallel update schedule, such that their limit dynamics are isomorphic, and the local functions of their central automaton equivalent. This is done by simply stopping the process of Algorithm 2 earlier to preserve the TC shape of the network.

The only difference between Algorithms 2 and 3 is that the latter restricts the reductions it operates. If two local functions are found to be equivalent up to some function $g$, the algorithm removes a node if and only if these local functions are duplicates of the previous local function of the central automaton




$f_a(x) = x_a \lor x_d \lor \neg x_h$
$f_b(x) = \neg x_a$
$f_c(x) = x_b$
$f_d(x) = x_c$
$f_e(x) = x_a$
$f_h(x) = x_e$

Initial network, $\Delta = (\{h\}, \{a, c, d, e\}, \{b\})$

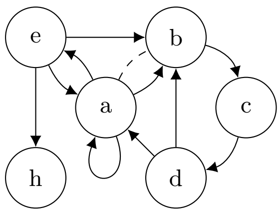
---

$f_a(x) = x_a \lor x_d \lor \neg \theta_h$
$f_b(x) = \neg \theta_a$

...

Algorithm 1, first loop (c, d, e and h are unchanged)

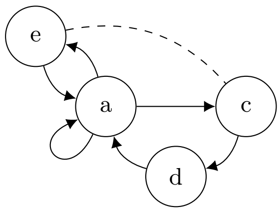
---




$f'_a(x) = x_a \lor x_d \lor \neg x_e$
$f'_b(x) = \neg(x_a \lor x_d \lor \neg x_e)$
$f'_c(x) = x_b$
$f'_d(x) = x_c$
$f'_e(x) = x_a$
$f'_h(x) = x_e$

Algorithm 1, second loop

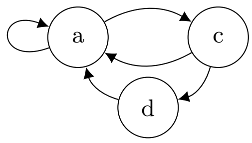
---




$f'_a(x) = x_a \lor x_d \lor \neg x_e$
$f'_c(x) = \neg x_a$
$f'_d(x) = x_c$
$f'_e(x) = x_a$

Algorithm 2 after the deletion of $h$ and the merge of $b$ into $a$. This is where Algorithm 3 ends

---




$f'_a(x) = x_a \lor x_d \lor \neg x_c$
$f'_c(x) = x_a$
$f'_d(x) = \neg x_c$

End of Algorithm 2, after the merge of $e$ into $c$

**Fig. 4.** Application of Algorithm 2 and 3 on an example network. Different steps of the algorithm are represented and separated using horizontal lines. At each step, the interaction graph or update graph and the local functions are the result of the operations precised on the right. As the initial network is a TC, the fourth step represents the result returned by Algorithm 3, which is a TC of smaller size. The fifth step represents the result returned by Algorithm 2, which is not a TC. Dashed lines in the interaction digraph connect automata the local function of which are equivalent up to a negation. Only the first graph is represented as an update digraph, as all the other networks are updated in parallel.

**Algorithm 3** Parallelization algorithm of a TC $F$ under the block-sequential update schedule $\Delta$, with a possible reduction in size

**Input**
  $F$ local functions of a network over $S$, encoded as circuits
  $\Delta$ update schedule over $S$
  $G_U(F_\Delta)$ update digraph of $F_\Delta$
**Output**
  $F'$ local functions of a parallel network over a subset of $S$, encoded as circuits
let $F' \leftarrow$ apply Algorithm 1 to $F_\Delta$, $G_U$
let $G_I(F') \leftarrow$ the interaction digraph of $F'$
**for** $(u, v) \in S^2$, such that either $u$ or $v$ has more than one in-neighbor in $G_I(F')$ **do**
  **if** $\forall x \in \Sigma^n, f_u(x) = g(f_v(x))$ **then** ▷ for some $g : \Sigma \to \Sigma$
    **for** $(u, w) \in E(G_I(F'))$ **do**
      apply the substitution $x_u \mapsto g(x_v)$ in $f_w$
**for** $u \in S$ **do**
  **if** $u$ has no accessible neighbors in $G_I(F')$ **then**
    $S \leftarrow S \setminus \{u\}$ ▷ $u$ is removed from the network
return $F'$

of the network. Removing duplications of any function that is part of a cycle would merge the cycles and the network would no longer be tangential cycles, in a way that is harder to count the reductions for. Since Algorithm 3 is a variation of Algorithm 2 that only does less reductions, Theorem 2 still applies in its case. An application of Algorithms 2 and 3 is illustrated in Figure 4 and the difference between the algorithms is highlighted.

**Theorem 3.** *Let $F$ be a TC and $\Delta$ a block-sequential update schedule. The amount of reductions in size that Algorithm 3 operates on $F_\Delta$ is the number of $<$-edges in the update digraph of $F_\Delta$, and the result is a TC.*

*Sketch of proof.* We show that in a $TC$, all the local transformations operated by the algorithm result in the removal of exactly one automaton, and that those transformations locally preserve the structure of the TC.

If Algorithm 2 cannot be polynomial in the worst case under the hypothesis that P $\neq$ NP, Algorithm 3 can be reduced to a very simple rule of thumb: taking a TC with a block-sequential update schedule, we obtain the equivalent parallel TC by reducing each cycle by the number of $<$-edges that its update digraph contains. This process is linear when the interaction digraph is provided as part of the input, since what is required is to count the number of $<$-edges along the various paths.

### 4.2 Reducing parallel Boolean TCs further

Applying Algorithm 2 to its full extent to a Boolean TC may result in a larger reduction in size. As any automaton that is not the central one has a unary function as its local function, any pair of non-central local functions will be equivalent up to some $g : \Sigma \to \Sigma$ if they are influenced by the same automaton. For example, if the central automaton influences three other automata that represent the start of three chains, these three automata can be merged into one. Continuing this zipping process yields a final network only as large as the longest cycle of the initial TC.

This process is not straightforward for non-Boolean TCs, as the local functions along the chains can be non-reversible using modular arithmetics, for example. Optimizing these networks is still possible, but requires a more complex set of substitutions to do so. It has been proven in general using modules and output functions [13]. The following theorem corresponds to the Boolean case, proven with more classical means. An example of its application is illustrated in the two last steps of Figure 4.

**Theorem 4.** *Let $F$ be a Boolean TC. Applying Algorithm 2 to $F_\pi$ generates a network $F'$ whose size is that of the largest cycle in $F$.*

*Sketch of proof.* All the cycles composing $F$ are merged together in a 'zipping' transformation.

## 5 An application: disjunctive double cycles

As an application of this algorithm and as an example to the algorithm's capacities to reduce the size of the provided network, we turn to the family of disjunctive double cycles. Notice that the result still holds for conjunctive double cycles since conjuctive and disjunctive cycles have isomorphic dynamics [12,11].

In disjunctive automata networks, an edge $(u, v)$ is signed positively if the $x_u$ appears as a positive variable in $f_v$. An edge $(u, v)$ is signed negatively if $x_u$ appears as a negative variable in $f_v$. A cycle is said to be positive if it contains an even number of negative edges, and negative otherwise.

A *disjunctive double cycle* is an automata network with an interaction digraph that is composed of two automata cycles that intersect in one automata. The local function of this central automata is a disjunctive clause. This family of networks is very simple to define, and is a simple and intuitive next step after the family of Boolean automata cycles, which are composed of a single cycle.

Both families have been characterized under the parallel update schedule [12,7]; that is to say, given basic parameters concerning the size of the cycles, their sign, and any integer $k$, an explicit formula (defined as a polynomially computable function) has been given among other to count the number of limit cycles of size $k$ of such networks under the parallel update schedule. In this section, we extend this characterization to the block-sequential cases by showing how applying our algorithm reduces the network to a smaller case in the same family of networks.

Furthermore, as Boolean automata cycles and disjunctive double cycles are TCs, our method can be simplified to the simple following rules: given a TC $F$, a block-sequential update schedule $\Delta$, count the number of $<$-edges in the update digraph $G_U(F_\Delta)$; for every cycle, substract to its size the number of such edges it contains, while keeping its sign; the final network, under the parallel update schedule, and the initial network under $\Delta$ have isomorphic limit dynamics. This is a simple application of Theorem 3, and of the rule of thumb deduced from Algorithm 3.

We denote $DC(s, s', a, b)$ the disjunctive double cycles with cycle sizes $a, b$ and signs $s, s'$.

**Theorem 5.** *Let $D = DC(s, s', a, b)$ be disjunctive double cycles, $\Delta$ a block-sequential update schedule. For $A$ ($B$ respectively) the number of $<$-edges on the cycle of size $a$ ($b$ respectively) in $G_U(F_\Delta)$, the limit dynamics of $D_\Delta$ is isomorphic to that of $D'_\pi$, where $D' = DC(s, s', a - A, b - B)$.*

*Proof.* This is a straightforward application of Theorem 3. □

## 6 Conclusion

In this paper we provide a novel algorithm which allows the reduction in size of automata networks, in particular while passing the network from a block-sequential to a parallel update schedule, while keeping isomorphic limit dynamics. While this algorithm is too computationally expensive for the general case, we study the specific family of intersection of automata cycles, on which this algorithm is easily applied. This study allows the discovery that all block-sequential tangential cycles have isomorphic limit dynamics to parallel tangential cycles. Finally, we apply this fact to Boolean automata double cycles to characterize their behavior under block-sequential update schedules.

It seems now clear to us that the difference between the parallel update schedule and block-sequential update schedules is that the latter changes the timing of the information along sections of the network. In particular structures such as tangential cycles can be directly translated into an equivalent parallel network with shorter cycles. We are interested in seeing what effects this translation could have in a more general set of families of networks, and if there exists other families in which block-sequential update schedule lead to equivalent parallel networks which are still part of the family.

As a perspective, we would like to characterize more redundancies that can be removed from networks to help with the computation of their dynamics. For example, we are currently interested in more complex compositions of automata cycles, and have already found equivalences that show that many networks are equivalent in their limit dynamics where complex parts of automata networks can be moved alongside cycles without affecting the network's limit dynamics.

Isolated paths are also a strong candidate for size reduction. Isolated paths are paths that lead from cycles to other cycles but can only be crossed once. Our current algorithms conserves such paths, despite it being possible to reduce them completely without changing the limit dynamics of the network in many cases, for example when an isolated path is the only way to go from one part to another. We have to be careful when multiple isolated paths exit from and join onto the same parts, as the synchronicity of the information in the entire network must be preserved.

**Acknowledgments.** This work has been partially funded by ANR-18-CE40-0002 FANs project and ECOS-Sud CE19E02.

## References

1. J. Aracena, E. Goles, A. Moreira, and L. Salinas. On the robustness of update schedules in Boolean networks. *Biosystems*, 97:1–8, 2009.
2. M.I. Davidich and S. Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLoS One*, 3:e1672, 2008.
3. J. Demongeot, A. Elena, M. Noual, S. Sené, and F. Thuderoz. "Immunetworks", intersecting circuits and dynamics. *J. Theor. Biol.*, 280:18–33, 2011.
4. 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.
5. J. Demongeot, T. Melliti, M. Noual, D. Regnault, and S. Sené. *Automata and Complexity: Essays presented to Eric Goles on the occasion of his 70th birthday*, volume 42 of *Emergence, Complexity, Computation*, chapter On Boolean automata isolated cycles and tangential double-cycles dynamics, pages 145–178. Springer, 2022.
6. J. Demongeot, M. Noual, and S. Sené. On the number of attractors of positive and negative Boolean automata circuits. In *Proceedings of WAINA'10*, pages 782–789. IEEE Press, 2010.
7. J. Demongeot, M. Noual, and S. Sené. Combinatorics of Boolean automata circuits dynamics. *Discr. Appl. Math.*, 160:398–415, 2012.
8. Z. Gao, X. Chen, and T. Başar. Stability structures of conjunctive boolean networks. *Automatica*, 89:8–20, 2018.
9. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, 22:437–467, 1969.
10. L. Mendoza and E. R. Alvarez-Buylla. Dynamics of the genetic regulatory network for *Arabidopsis thaliana* flower morphogenesis. *J. Theor. Biol.*, 193:307–319, 1998.
11. M. Noual. Dynamics of circuits and intersecting circuits. In *Proceedings of LATA'12*, volume 7183 of *LNCS*, pages 433–444. Springer, 2011.
12. M. Noual. *Updating Automata Networks*. PhD thesis, École Normale Supérieure de Lyon, 2012.
13. P. Perrotin. *Simulation entre modèles de calcul naturel et modularité des réseaux d'automate*. PhD thesis, Aix-Marseile Université, 2021.
14. F. Robert. Itérations sur des ensembles finis et automates cellulaires contractants. *Linear Algebra and its Applications*, 29:393–412, 1980.
15. L. Salinas. *Estudio de modelos discretos: estructura y dinámica*. PhD thesis, Universidad de Chile, Santiago, Chile, 2008.
16. R. Thomas. Boolean formalization of genetic control circuits. *J. Theor. Biol.*, 42:563–585, 1973.
17. R. Thomas. On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. In *Numerical methods in the study of critical phenomena*, pages 180–193. Springer, 1981.

# A Proofs

*Property 1.* Algorithm 1 always returns, and does so in polynomial time.

*Proof.* Let us denote by $<$*-graph* the subset of the graph $G_U(F_\Delta)$ where only the $<$-edges have been preserved. The $<$-graph of $F_\Delta$ is always a tree (or multiple disconnected trees): if this wasn't the case, there would be a cycle of $<$-edges in $G_U(F_\Delta)$, which would mean a cycle of automata that are all updated strictly before their out-neighbor, which is impossible.

Algorithm 1 will place a $\theta$ symbol for every edge in the $<$-graph. In the second loop, the selected $s$ is always a leaf of one of the trees contained in the $<$-graph. The applied substitution removes that leaf from the $<$-graph. By the structure of a tree, all the $<$-edges will be removed and the algorithm terminates.

To see that this algorithm can be performed in polynomial time, consider that all of the local functions are encoded as circuits. As such, it is enough to prepare a copy of each local function into one large circuit, on which every substitution will be performed. Any substitution $x_u \mapsto \theta_u$ is performed by renaming the corresponding input gate. Any substitution $\theta_s \mapsto f_s(x)$ is performed by replacing the input gate which corresponds to $\theta_s$ by a connection to the output gate of the circuit that computes the local function $f_s$. These substitutions are performed for every $<$-edge in the update digraph of $F_\Delta$, which is part of the inputs. The resulting circuit is then duplicated for every automaton in the output network, which leads to a total size of no more than $k^2$, for $k$ the size of the input. □

**Theorem 1.** *For any $F_\Delta$ Algorithm 1 returns a network $F'$ such that the dynamics of $F_\Delta$ is equal to that of $F'_\pi$.*

*Proof.* Let us consider some configuration $x \in \Sigma^n$, and let us compute its image $x'$ in both systems. Let us consider the initial block $X_0$ in $\Delta$. For any automaton in $X_0$, its local function is untouched in $F'$, and thus $F_\Delta(x)|_{X_0} = F'(x)|_{X_0}$. Suppose that $F_\Delta(x)|_{X_0 \cup \ldots \cup X_k} = F'(x)|_{X_0 \cup \ldots \cup X_k}$ for some $k$, let us prove that is true when including the next block $X_{k+1}$.

Let $v \in X_{k+1}$. By the nature of updates in $\Delta$, $f_v$ will be updated using the values in $F_\Delta(x)$ for any $x_u$ such that $u \in X_0 \cup \ldots \cup X_k$, and in $x$ otherwise. In $F'$, in the local function $f'_v$ and for any $u \in X_0 \cup \ldots \cup X_k$ that influences $v$, a substitution has replaced $x_u$ by $f'_u(x)$, which implies that the value of $v$ will be updated using a value of $u$ in $F'(x)$. Pulling this together, we obtain that $f_v(x) = f'_v(x)$ and $F_\Delta(x)|_{X_0 \cup \ldots \cup X_{k+1}} = F'(x)|_{X_0 \cup \ldots \cup X_{k+1}}$, and the recurrence yields $F_\Delta(x) = F'(x)$ for any $x$. □

**Theorem 2.** *For any $F_\Delta$, Algorithm 2 returns a network $F'$ such that the limit dynamics of $F_\Delta$ and $F'_\pi$ are isomorphic.*

*Proof.* By Theorem 1, the network $F'$ returned by the application of Algorithm 1 to $F_\Delta$ has identical dynamics to $F_\Delta$.

Algorithm 2 operates two kinds of modifications.

The first operation is replacing the influence of any automaton $u$ by another automaton $v$ if they are found to have equivalent local function up to some $g : \Sigma \to \Sigma$, that is, $f_u = g \circ f_v$. For any configuration $x$, the value of $f_u(x)$ and $g(f_v(x))$ are always equal. Thus, substituting the variable $x_u$ by $g(x_v)$ in the local functions of every out-neighbor of $u$ will lead to an identical limit behavior. After this substitution, the automaton $u$ does not have any influence over the network. Morever, all its previous out-neighbors in $G_I(F')$ are now the out-neighbors of $v$.

The second operation is iteratively removing automata that do not influence any automaton. Let $u$ be such an deleted automaton. Consider a limit cycle $(x^1, x^2, \ldots, x^k)$ in $G$. By definition of a limit cycle, $G(x^i) = x^{i+1}$ for any $i$, $G(x^k) = x^1$, and $x^i = x^j \Rightarrow i = j$. Consider the component $x^i_u$ for some $i$. Since $u$ does not influence any automaton, $x^{i+1}$ is a function of $x^i|_{S\setminus\{u\}}$. As the entire sequence is aperiodic, the sequence of the subconfigurations $x^i|_{S\setminus\{u\}}$ is also aperiodic, and the attractor is preserved in $F'$. □

**Theorem 3.** *Let $F$ be a TC and $\Delta$ a block-sequential update schedule. The amount of reductions in size that Algorithm 3 operates on $F_\Delta$ is the number of $<$-edges in the update digraph of $F_\Delta$, and the result is a TC.*

*Proof.* Algorithm 1 operates a substitution for every $<$-edge in the update digraph of $F_\Delta$. In this proof, we will show that each of the possible transformations implies the removal of exactly one node from the network.

For any such edge $(u, <, v)$, there are two cases. Either $u$ is the central automaton, or not. In any case, $u \neq v$ since the contrary would imply that an automaton is updated strictly before itself.

If we suppose that $u$ is the central automaton, this means that $f_v$ is a local function that only depends on $x_u$. It can thus be written $f_v(x) = g(x_u)$ for some $g : \Sigma \to \Sigma$. After the application of Algorithm 1, we thus obtain $f_v(x) = g(f_u(x))$, which implies the removal of either $u$ or $v$ (but at this point, not both) by Algorithm 3.

If we suppose that $u$ is not the central automaton, this means that $f_v$ is an arbitrary formula which contains $x_u$, and $f_u$ is a function of the form $f_u(x) = g(x_w)$ for some $g$ and some $w \in S$. Note that $w \neq u$ by the hypothesis that $F$ is a TC, either $w$ is the previous automaton in the path, or it is the central automaton $v$. As such, applying Algorithm 1 substitutes any mention of $x_u$ in $f_v$ by $g(x_w)$. Previously, $u$ only had one accessible neighbor, as it was part of a path connecting to the central automaton. This leaves $x_u$ without any accessible neighbors in the interaction digraph of $F$, which means that it is removed by Algorithm 3. If the removed edge is part of a cycle, this means that this cycle will be reduced in size. If the edge is part of the tangent, this means that the tangent will be reduced in size.

We thus obtain that the number of reductions is at least the number of $<$-edges in the update digraph of the network. Suppose now that some extra automaton $u$ is removed on top of any $<$-edge related reduction. First observe that if $u$ has no accessible neighbor, it must have had none from before the application of Algorithm 1, since in none of the two cases are external automaton disconnected from each other. Now suppose that $f_u$ is equivalent to some $f_v$ up to some $g$. Neither $u$ nor $v$ can be the central automaton, as any duplication of that function is handled in the first case. This proves that the number of reductions is exactly the number of $<$-edges in the update digraph of $(F, \Delta)$.

Let us now show that the result of Algorithm 3 is a TC. If the initial network had a central automata, there still exists a unique central automata at the end of the algorithm, even if the original central automata was removed in a chosen reduction. Paths that exit the central automata in the previous network still exit the central automata in the result, in the same number, and still share some tangent. The paths can be smaller in size, as well as the tangent, but they still end in the central automata. □

**Theorem 4.** *Let $F$ be a Boolean TC. Applying Algorithm 2 to $F_\pi$ generates a network $F'$ whose size is that of the largest cycle in $F$.*

*Proof.* Starting from the initial TC $F$, all of the automata directly influenced by the automata at the end of the tangent $u$ (but that are not $u$) have local functions $f_v(x) = g(x_u), f_w(x) = h(x_u), \dots$ for $g, h, \dots : \Sigma = \{0, 1\} \to \Sigma$. All these functions $g, h, \dots$ are not constant functions, since the automata that they represent are influenced by an automaton by hypothesis. Thus, they can only be defined as the identity or the negation of $x_u$. As a consequence, all but one of these automata will be removed by the algorithm as they are all equivalent up to some $g$.

This same argument can be repeated by taking all the automata influenced by the only automaton resulting from the previous iteration, excluding the central automaton. At each step, all of the automata at the same distance from the central automaton are merged. Hence, at the end of this process, whatever the choices made for merging automata along the iterative process, the resulting AN will be compoesed of $k$ automata, with $k$ the length of the largest cycle of $F$. □

**Theorem 5.** *Let $D = DC(s, s', a, b)$ be disjunctive double cycles, $\Delta$ a block-sequential update schedule. For $A$ ($B$ respectively) the number of $<$-edges on the cycle of size $a$ ($b$ respectively) in $G_U(F_\Delta)$, the limit dynamics of $D_\Delta$ is isomorphic to that of $D'_\pi$, where $D' = DC(s, s', a - A, b - B)$.*

*Proof.* This is a straightforward application of Theorem 3. □