

Attractor identification in asynchronous Boolean dynamics with network reduction

Elisa Tonello¹ and Loïc Paulevé²

¹Freie Universität Berlin, Germany

²Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

May 3, 2023

Abstract

Identification of attractors, that is, stable states and sustained oscillations, is an important step in the analysis of Boolean models and exploration of potential variants. We describe an approach to the search for asynchronous cyclic attractors of Boolean networks that exploits, in a novel way, the established technique of elimination of components. Computation of attractors of simplified networks allows the identification of a limited number of candidate attractor states, which are then screened with techniques of reachability analysis combined with trap space computation. An implementation that brings together recently developed Boolean network analysis tools, tested on biological models and random benchmark networks, shows the potential to significantly reduce running times.

1 Introduction

Boolean networks are adopted as modelling tools to organise knowledge and explore possible behaviours emerging in biological processes [16, 23, 5]. From the logic describing the influence between species, dynamics are defined to express the evolution of variables at play. Update schemes that implement asynchrony are of particular interest, as they express one form of inherent stochasticity [18]. Attractors are fundamental structures that ought to capture the fate or stable behaviours of modelled systems. Recently, the topic of identification of attractors of asynchronous dynamics has seen renewed interest and several developments. To look at the last two years alone, [15] suggested an algorithm that combines different techniques such as motif detection and time reversal; [2] developed a symbolic approach that can handle large transition systems, adopting binary decision diagrams to represent Boolean networks, and supporting partially defined update functions [1]; [21, 20] described approaches based on feedback vertex set identification and model checking reachability analysis. The new techniques have enabled the handling of models of increasing complexity, as summarized in [20].

With this work we want to investigate the usefulness of network reduction in the investigation of attractors of asynchronous state transition graphs. We refer specifically to the popular reduction method that consists in the iterative elimination of non-autoregulated components, as described in [9, 10, 22]. When a variable is selected for elimination, the regulatory functions of its targets are changed to remove the selected variable and replace it with its regulatory function. The asynchronous dynamics and regulatory structures of the original network and the network obtained with this reduction process are related in a useful way; for instance, the networks have the same steady states, and the number of attractors of the reduced network cannot be smaller than the original one. Properties of network reduction have already been exploited for attractor identification, in particular in [15], where reduction is used in combination with other methods under some specific cases. Here we show that a systematic use of variable elimination, adopted as the first step of the network analysis, can be quite beneficial.

The standing of variable elimination as a useful tool for identification of asynchronous attractors comes from the following property. For each attractor $\mathcal{A}$ of a Boolean network f , if $\tilde{f}$ is obtained from

f by iteratively eliminating some non-autoregulated components, there exists at least one state in an attractor of $\tilde{f}$ from which a state in $\mathcal{A}$ can be reconstructed, by tracing the reduction process backwards. This property allows us to identify, from the attractors of $\tilde{f}$, a limited set of “candidate” states that cover all the attractors of f (Section 2.2). Then, a screening of these candidate states is required for filtering out those outside the attractors of f . By calculating the minimal trap spaces [19, 6] we can first check if a candidate state is part of an attractor which is contained in a minimal trap space, and is the sole attractor contained in that minimal trap space. If there is more than one candidate attractor in a minimal trap space, or if there are candidate states outside of minimal trap spaces, other checks are required to determine whether they are actually part of an attractor. More computationally demanding techniques, for example from model checking, are useful for this step [12, 21] to check if a previously identified attractor, or a trap space not containing the candidate state, can be reached from the candidate state.

We tested these ideas on both biological and random networks by implementing variable elimination using colomoto’s `minibn` [8]. The reduced network were then studied using AEON [1] and mtsNFVS [20], to identify candidate attractor states. We used `trappist` to find the minimal trap spaces [19] and enable the first screening of the candidate states. For the check of the remaining candidate states we used mtsNFVS’s reachability analysis software [20]. We found, in general, high potential for reduction of computational times (Section 3.2). In addition, we observed that the reachability check, which is the most computationally expensive task in the pipeline, needs to be invoked only occasionally. In fact, although variable elimination can lead to an increased number of attractors, this appears to happen in a quite limited number of cases, meaning that the number of candidate states is often very close, if not equal, to the number of attractors. As a result, the amount of work necessary to handle such situations is also contained. In the last section (Section 4) we comment on some avenues that could be explored for possible further improvements of the techniques discussed.

2 Background

2.1 Boolean networks

We call V the set of variables or species of interest, and set $n = |V|$. A Boolean network is defined by a map $f = (f_1, \dots, f_n): 2^n \rightarrow 2^n$. The set 2^n is called the set of states or configurations of the Boolean network. For $i = 1, \dots, n$ and $x \in 2^n$, we denote by $\bar{x}^i$ the state that coincides with x on all components except i . A *subspace* is a subset of the state space consisting of all the states that share the same values for a set of components, called the *fixed* variables of the subspace.

The Boolean function f_i , $i = 1, \dots, n$, is sometimes called the *update function* of variable i . The *influence graph* $G(f)$ of f is the signed multi-digraph with set of nodes $\{1, \dots, n\}$ and edges capturing the dependence of update functions on each variable: there exist an edge (i, j) in $G(f)$ with sign s if and only if, for some $x \in 2^n$, $f_j(\bar{x}^i) \neq f_j(x)$ and $s = (f_j(\bar{x}^i) - f_j(x))(\bar{x}_i - x_i)$.

A *state transition graph* or *dynamics* associated to f is a graph with set of vertices 2^n and set of edges (also called *transitions*) that depends on the chosen semantics. This work deals with the *asynchronous dynamics*, a form of non-deterministic dynamics which includes only transitions between states differing by one component. Specifically, an edge exists from a state x to a state $\bar{x}^i$ if and only if $f_i(x) \neq x_i$. We will write $\Gamma(f)$ to denote the asynchronous state transition graph of f .

A *trap set* for the dynamics is a subset of the state space that does not admit any outgoing transition. An *attractor* is a trap set that is minimal under inclusion. Attractors can consist of singleton states, which are called *steady states* or *fixed points*, or can involve more than one state, in which case they are referred to as *cyclic* or *complex attractors*.

Determining whether a given state belongs to an attractor is a PSPACE-complete problem with asynchronous or synchronous dynamics [13]. In practice, computations usually rely on computing (partly and symbolically) the state transition graph, which can be significantly more complex in the asynchronous than in the synchronous case.

Trap spaces are subspaces that do not admit any outgoing transition. They can be viewed as generalizations of steady states, since they are fixed points for the restriction of the dynamics to some

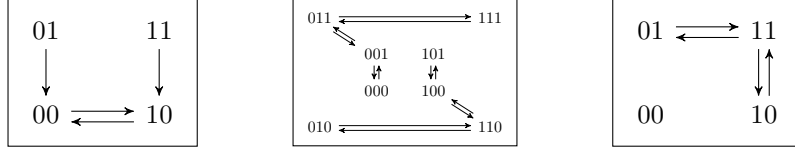


Figure 1: Asynchronous state transition graphs of the Boolean networks f , g and h of Example 2.1. $\Gamma(f)$ has one minimal univocal attractor, $\Gamma(g)$ has two minimal nonunivocal attractors, and $\Gamma(h)$ has a minimal univocal and a nonminimal attractor.

components. Compared to attractor identification, computation of trap spaces and related properties is a much more tractable problem, due to a largely reduced theoretical complexity [6]. Minimal trap spaces are particularly interesting, since each of them must contain at least one attractor. Often minimal trap spaces of Boolean networks that serve as biological models contain only one attractor [3], meaning, for instance, that they can replace the attractor they contain as reachability targets in reachability analysis. Although some properties of the regulatory structure that can guarantee a “good” attractor landscape (attractors only in minimal trap spaces, uniqueness of attractors in minimal trap spaces) have been identified [14, 11], these have limited application. In general, establishing whether a Boolean network admits multiple attractors in a minimal trap space, or attractors outside of minimal trap spaces (the “motif-avoidant” attractors of [15]), remains a difficult task.

Based on these considerations, we can lay out the following classification of attractors of Boolean asynchronous dynamics into four categories, which will be useful in our discussion later:

- (A) *steady states*: these are “easy” to find. Since for our algorithm we calculate all minimal trap spaces, we find the steady states as the minimal trap spaces where all variables are fixed variables.
- (B) *minimal univocal*: we use the term that was introduced in [3] to refer to minimal trap spaces that contain only one attractor. For convenience, we apply the term also to the attractors contained in these minimal trap spaces.
- (C) *minimal nonunivocal*: these are attractors that are contained in a minimal trap space but are not the only attractor contained in that trap space. To investigate their existence we have the convenience of being allowed to restrict the search space to the minimal trap space.
- (D) *nonminimal* or *motif-avoidant* [15]: these are the most difficult to detect or exclude the existence of.

Example 2.1. The asynchronous state transition graphs of the maps

$$\begin{aligned}
 f(x_1, x_2) &= (x_1 x_2 \vee \bar{x}_1 \bar{x}_2, 0), \\
 g(x_1, x_2, x_3) &= (x_2 \bar{x}_1 \vee x_1 \bar{x}_2, x_1(x_2 x_3 \vee \bar{x}_2 \bar{x}_3) \vee \bar{x}_1(x_2 \bar{x}_3 \vee x_3 \bar{x}_2), x_2 x_3 \vee \bar{x}_2 \bar{x}_3), \\
 h(x_1, x_2) &= (x_1 \bar{x}_2 \vee \bar{x}_1 x_2, x_1 \bar{x}_2 \vee \bar{x}_1 x_2)
 \end{aligned}$$

are represented in Fig. 1.

$\Gamma(f)$ has a cyclic attractor and minimal trap space $\{00, 10\}$.

$\Gamma(g)$ has two attractors that are nonunivocal, since they are found in the same minimal trap space (the full space).

$\Gamma(h)$ has two attractors, one steady state 00 and one nonminimal attractor $\{01, 10, 11\}$. The steady state is the unique minimal trap space.

In the next section we review how the removal of components works in Boolean networks and discuss how it can help with identification of attractors.

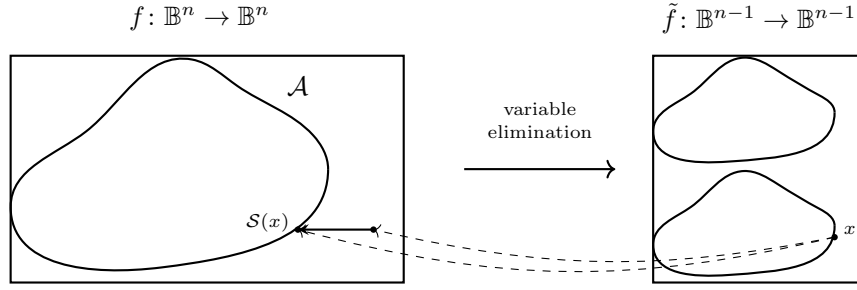


Figure 2: Idea behind the approach: states in attractors of reduced networks $\tilde{f}$ can be used to find candidate states in attractors of f .

2.2 Reduction

A popular reduction method for asynchronous dynamics of Boolean networks iteratively eliminates variables that are not autoregulated [9, 10, 22]. The approach has been recently extended to negatively autoregulated components [17]. Although all our observations here can be extended, *mutatis mutandis*, to negatively autoregulated components, we only discuss the standard case for sake of simplicity.

Suppose that $G(f)$ has a node that does not have a loop. Without loss of generality, we can assume that the node is n . We write $\pi: 2^n \rightarrow 2^{n-1}$ for the projection on the first $n-1$ variables.

By definition of $G(f)$, we have that $f_n(x, 0) = f_n(x, 1)$ for all $x \in 2^{n-1}$. In particular, the state transition graph of f admits exactly one of the transitions from $(x, 0)$ to $(x, 1)$ or from $(x, 1)$ to $(x, 0)$. We can therefore define a map $\mathcal{S}^n$ that associates to a “reduced” state a state in the larger space, as follows:

$$\begin{aligned} \mathcal{S}^n: 2^{n-1} &\rightarrow 2^n \\ x &\mapsto (x, f_n(x, 0)) = (x, f_n(x, 1)). \end{aligned}$$

The reduction $\tilde{f}$ of f obtained by elimination of component n is then defined as

$$\tilde{f} = (f_1 \circ \mathcal{S}^n, \dots, f_{n-1} \circ \mathcal{S}^n): 2^{n-1} \rightarrow 2^{n-1}.$$

Given $y \in 2^n$, the state $\mathcal{S}^n(\pi(y))$ can be thought of as the “representative state” of the pair $(\pi(y), 0)$, $(\pi(y), 1)$, or the state that “survives the reduction”, since all transitions leaving the state $\mathcal{S}^n(\pi(y))$ have a corresponding transition in $\Gamma(\tilde{f})$, whereas the transitions leaving the state $\overline{\mathcal{S}^n(\pi(y))}^n$ are not guaranteed to be preserved [10].

It was shown in [9, 10] that f and $\tilde{f}$ admit the same number of steady states, and the number of attractors of $\tilde{f}$ is greater or equal to the number of attractors of f . The following result, which forms the basis for our method, is a simple consequence of properties proved in [10].

Theorem 2.2. *If $\mathcal{A}$ is an attractor of f , then there exists at least one attractor for $\tilde{f}$ in $\pi(\mathcal{A})$, and for each $x \in \pi(\mathcal{A})$ contained in an attractor of $\tilde{f}$, $\mathcal{S}^n(x) \in \mathcal{A}$.*

Proof. The first part is a consequence of the observation that, if B is a trap set for $\Gamma(f)$, then $\pi(B)$ is a trap set for $\Gamma(\tilde{f})$.

Given a state x in an attractor for $\tilde{f}$ contained in $\pi(\mathcal{A})$, by definition of π either $\mathcal{S}^n(x)$ or $\overline{\mathcal{S}^n(x)}^n$ is in $\mathcal{A}$, and since $\mathcal{S}^n(x)$ is reachable from $\overline{\mathcal{S}^n(x)}^n$, $\mathcal{S}^n(x)$ must be in $\mathcal{A}$. $\square$

The theorem gives us the following property: if there exists an attractor for f , then we are able to identify one state of this attractor by finding the attractors of the reduced version $\tilde{f}$, sampling a state from each of these attractors, and “lifting” the states to the original space using $\mathcal{S}^n$ (Fig. 2).

Before we discuss more in detail how we can use the reduced network to identify attractors of the original network, let us look at some examples, which can give an idea of what can happen to attractors with variable elimination.

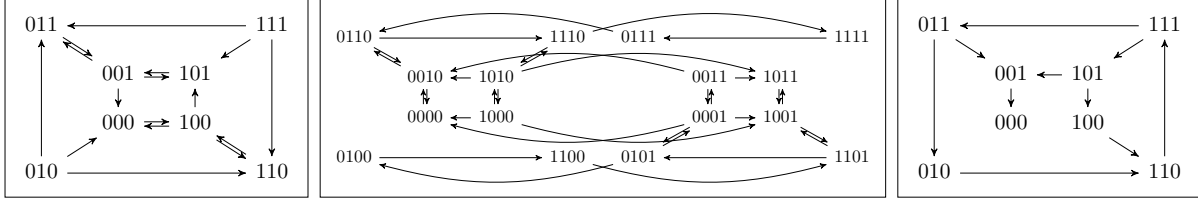


Figure 3: Asynchronous state transition graphs of the Boolean networks $\hat{f}$, $\hat{g}$, $\hat{h}$ of Examples 2.4 to 2.6.

Example 2.3. Consider the map f from Example 2.1. Since the second variable is not autoregulated, it can be removed using the elimination method described in this section. The constant value 0 replaces x_2 in the update function of x_1 , giving the reduced network $\tilde{f}(x_1) = \bar{x}_1$. We have $\mathcal{S}^2(0) = \mathcal{S}^2(00)$ and $\mathcal{S}^2(1) = \mathcal{S}^2(10)$. This simple example is sufficient to illustrate how, given a state x in the attractor of a reduced network (say $x = 0$ here), to retrieve a state in an attractor of the original network we cannot pick any state in the preimage $\pi^{-1}(x)$ (the state 01 does not work), but we need to apply the function $\mathcal{S}^2$.

Example 2.4. If $\mathcal{A}$ is an attractor for a Boolean network and x is a state in $\mathcal{A}$, does $\pi(x)$ necessarily belong to an attractor of the reduced network? The answer is negative. Take for instance

$$\hat{f}(x_1, x_2, x_3) = (\bar{x}_1\bar{x}_2 \vee \bar{x}_1\bar{x}_3 \vee x_2\bar{x}_3, x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3, x_1\bar{x}_2 \vee \bar{x}_1x_2).$$

By removing the third component, we obtain the function f of Example 2.1. The state 011 is in the unique attractor of $\hat{f}$, while $\pi(011) = 01$ does not belong to any attractor of f .

In general, therefore, we are not able to retrieve all states of an attractor of a Boolean network by lifting states in attractors of its reduction. We will only find some states, from which we can visit the attractor if required.

Example 2.5. The Boolean network

$$\hat{g}(x_1, x_2, x_3, x_4) = (x_2\bar{x}_4 \vee \bar{x}_2x_4, x_4(x_2x_3 \vee \bar{x}_2\bar{x}_3) \vee \bar{x}_4(x_2\bar{x}_3 \vee x_3\bar{x}_2), x_2x_3 \vee \bar{x}_2\bar{x}_3, x_1)$$

has one cyclic attractor, that fills the whole state space (see Fig. 3). By removing variable x_4 we obtain the network g in Example 2.1, which has two attractors.

Example 2.6. The Boolean network

$$\hat{h}(x_1, x_2, x_3) = (x_1\bar{x}_3 \vee x_2\bar{x}_3, x_1\bar{x}_3 \vee x_2\bar{x}_3, x_1x_2)$$

has a unique attractor, the fixed point 000. By eliminating variable x_3 we obtain the Boolean network h in Example 2.1, which has two attractors.

Suppose that $\tilde{f}: 2^m \rightarrow 2^m$ is obtained from f by iteratively eliminating variables $n, n-1, \dots, m+1$, and that $\mathcal{A}_1, \dots, \mathcal{A}_M$ are attractors of $\tilde{f}$. Take one state $x^1, \dots, x^M \in 2^m$ in each attractor. We can reconstruct the corresponding states in 2^n by applying the map $\mathcal{S} = \mathcal{S}^n \circ \mathcal{S}^{n-1} \circ \dots \circ \mathcal{S}^{m+1}$. How can we establish whether each of these states is in an attractor of f , and how many attractors f has?

We can calculate the minimal trap spaces of f , and make the following observations. Given a set of candidate states $C = \{\mathcal{S}(x^1), \dots, \mathcal{S}(x^M)\}$:

- (a) If x is a steady state of $\tilde{f}$, then $\mathcal{S}(x)$ is a steady state of f (and all steady states of f can be calculated in this fashion).
- (b) if $\mathcal{S}(x)$ belongs to a minimal trap space t of f , and is the only state in C that belongs to t , then $\mathcal{S}(x)$ belongs to an attractor of f that is minimal univocal. We call these *univocal states*.

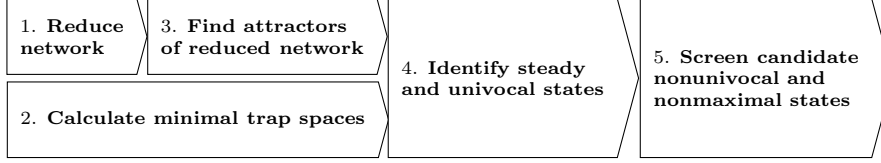


Figure 4: Main steps of the algorithm.

- (c) if $\mathcal{S}(x)$ is contained in a minimal trap space t of f , and is not the only state in C that is contained in this minimal trap space, then we need to study the dynamics in t to clarify whether each candidate state contained in t belongs to an attractor, and whether the candidate states in t belong to different attractors. We can call these states *candidate nonunivocal*. The number of states in C contained in t gives an upper bound on the number of attractors contained in t .
- (d) if $\mathcal{S}(x)$ is not contained in any minimal trap space, then $\mathcal{S}(x)$ *might* belong to a nonminimal attractor of f . To establish whether this is the case, and to find the number of nonminimal attractors, we need to do additional work. We refer to these states as *candidate nonminimal*.

Note that these observations would have to be slightly changed were one to consider the elimination of negatively autoregulated components [17]. We can now give the description of the approach to the identification of attractors based on elimination of components.

3 Method

Based on the analysis of the relationship between attractors of reduced versions of a Boolean network and attractors of the original network, we propose the following pipeline (see Fig. 4).

First reduce the network f by eliminating variables (step 1), and, possibly in parallel, find the minimal trap spaces of the Boolean network f (step 2). Then, identify one state for each attractor of the reduced the network $\tilde{f}$ that is not a steady state (step 3), obtaining a set of candidate attractor states for f .

Step 4 and 5 deal with the screening of these candidate attractor states. Step 4 is the easy part: we check whether each candidate state is contained in a minimal trap space. If a minimal trap space contains only one candidate state, then this is a univocal state and we identified a univocal attractor for f (point (b) of the last section).

Step 5 takes care of the remaining candidate states. These can be either multiple states contained in the same minimal trap space (point (c) of the last section), or states that do not belong to any minimal trap space (point (d) of the last section). In both scenarios, we have to study in some way the state transition graph to understand whether the candidate states belong to an attractor, and how many attractors exist (some candidate states might be part of the same attractor, which has been “split” during the reduction process, like in Example 2.5, or part of an attractor that was created by the reduction process Example 2.6). For this step one can use model checking approaches (see the next section), possibly combined with other techniques (see the discussion in the last section).

3.1 Implementation

3.1.1 Software

For our implementation, we use the Python library `colomoto's minibn` [8] to compute the network reduction (step 1 in Fig. 4). In `minibn`, the local functions of the Boolean network are represented in propositional logic formula, with usual Boolean algebra. Given a component i to reduce, our implementation simply substitutes x_i with the expression of f_i in all its targets. Then, basic Boolean expression simplifications are performed, which may result in variable elimination.

For step 2, we use `trappist` to calculate the minimal trap spaces [19]. Because `trappist` relies on a Petri net representation of the Boolean network, this step involves a transformation of each local Boolean

function f_i in two expressions in DNF form (one for $f_i(x) = 1$ and one for $f_i(x) = 0$), from which are derived the Petri net transitions.

For step 3, the identification of attractors of the reduced network, we consider two recently developed methods, AEON [1] and mtsNFVS [20]. In the analysis of [20], they have been shown to be the two fastest methods available, while implementing two very different approaches. The outputs of the two methods are also quite different: AEON, for instance, can give information on the attractors (cardinality, list of states) that is not directly available with mtsNFVS, and can deal with families of networks at the same time, do postprocessing control tasks, etc. Here we are only looking at the performance of the methods in regard to the identification of the number of attractors and their nature (steady or cyclic), and we are mostly concerned with understanding whether reduction might be useful in this respect.

The pieces are put together and step 4 performed in `python`. For step 5, we run the `java` tool `mtsNFVS.jar` available as part of mtsNFVS. The tool uses `Pint` [12] for a first check via static analysis followed by bounded and exact model checking [21]. We noticed however that mtsNFVS (without reduction) produces wrong results in several instances (some examples are provided in our project repository). Hence, the results obtained with mtsNFVS should be taken with caution. For the biological networks we considered, we checked that all methods found the same number of steady states and cyclic attractors. As we will discuss in the Results section, the reachability analysis step has limited impact on the overall process, since nonminimal candidates appear only rarely, and no nonunivocal candidates are found in any of the biological or random tests.

3.1.2 Elimination order and other considerations

The choice of order for the variables to be eliminated has an influence on the reduced network as well as on the running times. When a variable with r regulators and t targets is eliminated, the $r + t$ regulations could be replaced, in the worst case scenario, by $r \cdot t$ regulations. We therefore pick for elimination, at each step, one of the variables for which this product is minimum. A more systematic study of how the elimination order influences the running time could lead to identification of better elimination orders. For instance, one might try to favour elimination orders that have the least impact on the number of trap spaces.

Another crucial point when using variable elimination is deciding where to stop the iterative process. Unfortunately, there does not seem to be a simple answer to this question. In some cases, at some point in the elimination process reduction can introduce complicated update functions that can slow the processing steps that follow. In addition, although reduction seems to be beneficial as a preliminary step for the methods we tested, we will see that different levels of reduction might favour different methods variably. In our implementation we considered two possible parameters to stop the reduction process. One specifies the minimum size for the reduced network in terms of number of nodes (`stop_at`). The other looks at the minimum product of number of regulators and number of targets for each node (that is, the parameter used to choose the variable to eliminate), and sets a maximum for this value (`max_product`).

We also investigated the impact of the simplification of Boolean expressions on running times. We found that, although simplification is expensive, it leads to faster elimination. In the next section we will discuss the impact of the elimination step on running times for both biological and random networks.

3.2 Results

The experiments on the biological network models were run on a desktop computer with an Intel(R) Core(TM) i7-8700 processor, 32GB of RAM, operating system Debian GNU/Linux 11. The experiments on the random networks were run on an HPC cluster nodes with an AMD(R) Zen2 EPYC 7702 @ 2 GHz, 1TB of RAM, operating system, operating system CentOS Linux. The code is available at <https://github.com/etonello/attractors-with-reduction>.

3.2.1 Biological models

We run our implementation using AEON [1] and mtsNFVS [20] to find the attractors of the reduced networks, and compared the running times to those of AEON and mtsNFVS applied on the networks

Model	file name	file source	n. nodes	n. edges	n. steady states	n. cyclic attractors
MAPK	grieco_mapk	PyBoolNet	53	108	12	6
IL-6	IL_6_Signalling	mtsNFVS	55	95	28672	4096
EMT	selvaggio_emt	PyBoolNet	56	159	1452	0
T-LGL	TLGLSurvival	mtsNFVS	58	193	172	146
CACC	Colitis_associated_...	mtsNFVS	66	144	2	8
AD	A_model	mtsNFVS	74	198	0	2
AGS	id-148-AGS-CELL-FATE-...	biodivine	83	193	1	0
CC	cell_cycle_2019	mtsNFVS	87	467	8	0
SP	id-192-SEGMENT-POLA...	biodivine	102	432	65	0
SIPC	SIGNALING-IN-PROST...	mtsNFVS	116	428	2460	300
DSP	id-210-DRUG-SYNERGY-...	biodivine	144	367	0	1
C3.0	CASCADE3	mtsNFVS	176	449	0	1
EP	id-211-EPITHELIAL-DER...	biodivine	183	602	0	1

Table 1: Information on the sourcing of the bnet models considered, their size and the number of steady states and cyclic attractors.

Model	(1) max_product=20			(2) max_product=50			(3) max_product=100		
	n. nodes	n. edges	time	n. nodes	n. edges	time	n. nodes	n. edges	time
MAPK	10	41	0.1	10	41	0.0	10	41	0.1
IL-6	17	28	0.0	17	28	0.0	17	28	0.0
EMT	19	94	0.1	17	130	0.1	17	130	0.1
T-LGL	21	91	0.0	18	111	0.0	18	111	0.0
CACC	14	81	0.0	11	56	0.0	11	56	0.0
AD	11	93	0.0	10	97	0.0	10	97	0.0
AGS	2	4	0.0	2	4	0.0	2	4	0.0
CC	38	415	0.2	35	490	0.3	29	669	23.9
SP	35	234	0.0	33	273	0.1	32	357	0.1
SIPC	41	390	0.3	32	465	0.6	28	522	8.7
DSP	14	108	0.0	10	71	0.0	10	71	0.0
C3.0	23	191	0.0	14	175	0.1	13	193	0.1
EP	33	322	0.1	25	365	0.1	21	316	0.1

Table 2: Reduction scenarios considered for the biological models listed in Table 1. **max_product**= k indicates that the elimination ends when the product of number of regulators and number of targets is above k for all nodes. The elimination is also set to stop when the number of nodes reaches 10.

without reduction. We consider the seven models examined in [20] as available in the tool repository, as well as additional models extracted from the PyBoolNet repository [4] and the biodivine-boolean-models repository (<https://github.com/sybila/biodivine-boolean-models>). Information on the source of each model, as well as the size of the networks is given in Table 1. We refer the reader to the respective sources for references detailing each model, which explain if and how the models have been modified from their original sources.

As we mentioned in the previous section, different levels of reduction can have different impacts on running times of both the elimination procedure and the remainder of the attractor identification process. Here we report running times obtained in three scenarios, where we set the parameter **max_product** described in the previous section (maximum value accepted for the minimum product of in- and out-regulations over all nodes) to three levels, 20 in scenario (1), 50 in scenario (2) and 100 in scenario (3). In all three cases, we set the minimum size of the reduced network to 10. In some cases these parameters lead to network with fewer than 10 nodes, because constants variables might be generated

Model	AEON running times				mtsNFVS running times				
	No red.	(1)	(2)	(3)	No red.	(1)	(2)	(3)	
MAPK	5.7	0.3	0.3	0.3	28.9 $\pm$ 5.7 (3 DNF)	0.7	0.7	0.7	
IL-6	774.6	6.0	6.0	6.0	14.8 $\pm$ 1.8	7.4	7.4	7.4	
EMT	25.6	0.8	0.7	0.7	DNF	1.3	1.4	1.4	
T-LGL	17.5	0.9	0.9	1.1	2.2	1.8	1.9	2.7	
CACC	9.3	0.3	0.3	0.3	0.5	0.8	0.7	0.7	
AD	361.9	0.3	0.4	0.4	0.7	1.0	0.8	0.8	
AGS	1.7	0.3	0.3	0.3	0.6	0.7	0.7	0.7	
CC	DNF	27.0	11.1	26.9	8.2 $\pm$ 3.5	2.2	6.0	39.4	
SP	DNF	0.9	0.8	0.8	DNF	1.0	1.1	1.4	
SIPC	DNF	28.2	6.9	14.4	1664.7 $\pm$ 506.3	47.8 $\pm$ 5.6	53.7 $\pm$ 7.4	138.8	
DSP	DNF	0.4	0.4	0.4	2.3	0.8	0.7	0.7	
C3.0	DNF	0.5	0.4	0.4	2.1	1.0	1.0	0.9	
EP	DNF	1.5	0.6	0.5	62.7 $\pm$ 64.2	2.5	2.4	1.9	

Table 3: Running times for AEON [1] and mtsNFVS [20] without reduction and for the three reduction scenarios described in Table 2. The running times in the three reduction scenarios include the time for network reduction. Running times for mtsNFVS are averaged over five iterations, and show the standard deviation if this is higher than 1s. “DNF” indicates that the processing did not complete within one hour.

by the elimination process, and these are always eliminated. In Table 2 we show the number of nodes and signed regulations of the reduced network obtained in each of the three scenarios, as well as the time required by the reduction process. The time for reduction is below one second except for two networks, CC and SIPC, where it goes up to 24 and 9 seconds.

Table 3 shows the running times for the two methods and the different scenarios (no reduction, reduction scenario (1), (2) and (3)). The algorithm of mtsNFVS relies on some random choices, which cause its running times to vary, sometimes significantly. We report the minimum and maximum running times obtained on five iterations. We set a timeout of one hour, and “DNF” indicates that the processing did not complete within this time.

It is clear from the results that approaching the attractor identification problem by first reducing the network can speed up the processing significantly. We can also observe that a more aggressive reduction does not necessarily translate to faster attractor identification times. This can be observed for both attractor identification methods for the networks CC and SIPC. Looking at the numbers for these two networks, we can also see that the reduction levels giving the shortest processing times are not the same for AEON and mtsNFVS. In particular, mtsNFVS seems to work better with more conservative reduction levels, and in one case (network CC, scenario (3)) the time required for the reduction process was higher than the running time for mtsNFVS without reduction.

Importantly, only for one of the networks (TLGL) nonminimal candidates were identified, and no network presented non-univocal candidates, meaning that reduction had overall a limited impact on the general attractor configuration. Looking at what happens on randomly-generated networks might help to clarify whether these behaviours could be specific of biological models or more general.

3.2.2 Random benchmarks

As in [20], we generate random networks by invoking `generateRandomNKNetwork` from the R package BoolNet ([7]), fixing $K = 2$ for the number of regulators for each variable. We consider networks with $n = 100k$ nodes, with $k = 1, \dots, 5$, and create 10 networks for each size.

We tested several stopping conditions for the reduction process, and did not identify a general rule that would give optimal times in all cases. To give an idea of the range of possible results, we report running times for three reduction scenarios, where we set the parameter `max_product` to $\frac{n}{2}$, n and $2n$,

Model size	(1) max_product=n/2			(2) max_product=n			(3) max_product=2n		
	average n. nodes	average n. edges	average time	average n. nodes	average n. edges	average time	average n. nodes	average n. edges	average time
100	14.0	219.9	0.1	12.3	213.0	0.5	11.9	199.5	0.6
200	24.0	621.9	0.5	21.2	618.0	1.9	20.7	596.3	5.8
300	37.6	1418.5	2.1	34.4	1527.5	7.8	30.0	1240.0	19.7
400	49.4	2266.0	2.6	42.5	2471.0	23.1	41.1	2498.3	55.3
500	61.5	3489.8	18.4	55.3	3985.7	18.8	55.5	4683.5	286.5

Table 4: Statistics for three reduction scenarios on 10 random networks (generated with BoolNet [7], setting $K = 2$). `max_product=k` indicates that the elimination ends when the product of number of regulators and number of targets is above k for all nodes. The elimination is also set to stop when one-tenth of the number of nodes of the original network is reached.

while `stop_at` is set to $\frac{n}{10}$. The details of the three scenarios are shown in Table 4. In Table 5, we show the average running times for networks that were processed within the timeout of one hour, and the number of networks that were not processed in time.

We observe again that, by adopting the reduction approach, the number of networks that can be successfully analysed within the timeout given increases significantly. AEON without reduction could process 4 out of the 10 networks of size 100, and no network of larger size. With reduction and AEON, all networks of size 200 and 8 out of the 10 networks of size 300 could be processed, as well as one network of size 400. The running times are just a few seconds for networks of size 100, and vary significantly for larger networks.

The tool mtsNFVS without reduction could process all networks of size 100 and some networks of size 200 and 300. With reduction, networks of up to size 500 could be handled. However, as we pointed out in Section 3.1, the results generated by mtsNFVS might require further validation, as several failures were identified in test networks.

We can nevertheless, by observing the results obtained with AEON, note that the number of candidate states that need processing via reachability analysis remains very limited. Only five nonminimal candidate states were identified for the networks that were processed with reduction and AEON, and no nonunivocal candidate states. This suggests that nonminimal and nonunivocal attractors might be rare phenomena, even for random networks.

Reduction allowed the size of networks to be drastically reduced, but the results shown in Table 4 illustrate how different number of reduction steps might affect networks in different ways. Some networks could only be processed in time in scenario (3), with the highest number of eliminated variables; for others, processing times were lower when a smaller number of variables was removed.

4 Discussion

We investigated how reduction can make the process of attractor identification faster. We observed that reduction generally makes the process easier, but the impact can depend on the adopted attractor detection approach. Elimination of variables, while reducing the size of the network, increases the complexity of the influence graph and update functions. This, on its turn, has a different impact on the two tools we tested, AEON [1] and mtsNFVS [20], which implement very different approaches to identification of attractors. Deeper investigations targeted on the specific tool might give some insights on why certain levels of reduction work better than others. At the same time, we did limited analysis on the impact of the order of elimination; we cannot exclude that specific orders of elimination might be devised to better suit a specific method (for instance, for mtsNFVS, one might want to investigate orders that do not increase the size of minimal feedback vertex sets).

Although candidate states that require reachability analysis are only rarely encountered, this step might benefit from additional developments. One improvement would come from the screening of nonunivocal candidate states, as the technique of mtsNFVS currently can fail to detect the existence of multiple

AEON running times				
Model size	No red.	(1)	(2)	(3)
100	1104.6 $\pm$ 676.0 (6)	2.3 $\pm$ 1.3	3.0 $\pm$ 1.5	3.2 $\pm$ 1.5
200	(10)	83.7 $\pm$ 205.7	48.9 $\pm$ 113.7	146.1 $\pm$ 402.4
300	(10)	1331.3 $\pm$ 1394.7 (5)	696.0 $\pm$ 684.1 (3)	888.9 $\pm$ 1185.0 (2)
400	(10)	(10)	2994.6 (9)	3033.2 (9)
500	(10)	(10)	(10)	(10)
mtsNFVS running times				
Model size	No red.	(1)	(2)	(3)
100	3.3	6.9 $\pm$ 2.0	7.6 $\pm$ 2.1	7.7 $\pm$ 2.0
200	106.8 $\pm$ 476.8 (11)	378.4 $\pm$ 592.9 (5)	314.5 $\pm$ 629.4 (7)	379.8 $\pm$ 654.0 (5)
300	71.7 $\pm$ 90.5 (25)	1440.8 (49)	2298.6 $\pm$ 653.3 (44)	3305.8 $\pm$ 432.1 (46)
400	(50)	2059.3 $\pm$ 644.0 (26)	2639.5 $\pm$ 718.3 (43)	2239.0 $\pm$ 482.2 (41)
500	(50)	1672.3 $\pm$ 562.8 (44)	(50)	1276.6 $\pm$ 418.3 (46)

Table 5: Average running times for AEON [1] and mtsNFVS [20], on random networks without reduction and in the three reduction scenarios of Table 4. In parentheses is the number of processes that did not terminate within one hour. Since running times for mtsNFVS have a high variability, we run each test five times. The mean and standard deviation shown are over all successful tests for the given size.

attractors contained in the same minimal trap space. In addition, other techniques could be incorporated for the exclusion of existence of nonmaximal attractors. For instance, at the moment only minimal trap spaces are used. Larger trap spaces (the maximal trap spaces that do not contain the candidate state) could provide bigger targets for reachability analysis, while still being easily identifiable with a small modification to the approach of **trappist** [19].

Finally, attractor detection tools are capable of other tasks, for instance, AEON can perform detection of bifurcations and source-target control. There is the potential that reduction could be used sensibly to speed up these activities too. Each task needs to be studied individually and carefully for the implications of variable elimination.

Acknowledgements

ET was supported by the Deutsche Forschungsgemeinschaft (DFG) under Germany’s Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID 390685689). LP was supported by the French Agence Nationale pour la Recherche (ANR) in the scope of the project “BNeDiction” (grant number ANR-20-CE45-0001). Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d’Aquitaine (see <https://www.plafrim.fr>).

References

- [1] Nikola Beneš, Luboš Brim, Ondřej Huvar, Samuel Pastva, David Šafránek, and Eva Šmijáková. AEON.py: Python library for attractor analysis in asynchronous Boolean networks. *Bioinformatics*, 38(21):4978–4980, 2022.
- [2] Nikola Beneš, Luboš Brim, Samuel Pastva, and David Šafránek. Computing bottom SCCs symbolically using transition guided reduction. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*, pages 505–528. Springer, 2021.

- [3] Hannes Klarner and Heike Siebert. Approximating attractors of Boolean networks by iterative CTL model checking. *Frontiers in bioengineering and biotechnology*, 3:130, 2015.
- [4] Hannes Klarner, Adam Streck, and Heike Siebert. PyBoolNet: a python package for the generation, analysis and visualization of boolean networks. *Bioinformatics*, 33(5):770–772, 2017.
- [5] Arnau Montagud, Jonas Béal, Luis Tobalina, Pauline Traynard, Vigneshwari Subramanian, Bence Szalai, Róbert Alföldi, László Puskás, Alfonso Valencia, Emmanuel Barillot, Julio Saez-Rodriguez, and Laurence Calzone. Patient-specific Boolean models of signalling networks guide personalised treatments. *eLife*, 11, 2022.
- [6] Kyungduk Moon, Kangbok Lee, and Loïc Paulevé. Computational Complexity of Minimal Trap Spaces in Boolean Networks. *arXiv preprint arXiv:2212.12756*, 2022.
- [7] Christoph Müssel, Martin Hopfensitz, and Hans A Kestler. BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10):1378–1380, 2010.
- [8] Aurélien Naldi, Céline Hernandez, Nicolas Levy, Gautier Stoll, Pedro T Monteiro, Claudine Chaouiya, Tomáš Helikar, Andrei Zinovyev, Laurence Calzone, Sarah Cohen-Boulakia, et al. The CoLoMoTo interactive notebook: accessible and reproducible computational analyses for qualitative biological networks. *Frontiers in physiology*, 9:680, 2018.
- [9] Aurélien Naldi, Elisabeth Remy, Denis Thieffry, and Claudine Chaouiya. A reduction of logical regulatory graphs preserving essential dynamical properties. In *International Conference on Computational Methods in Systems Biology*, pages 266–280. Springer, 2009.
- [10] Aurélien Naldi, Elisabeth Remy, Denis Thieffry, and Claudine Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, 412(21):2207–2218, 2011.
- [11] Aurélien Naldi, Adrien Richard, and Elisa Tonello. Linear cuts in Boolean networks. *arXiv preprint arXiv:2203.01620*, 2022.
- [12] Loïc Paulevé. Pint: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface. In *CMSB 2017 - 15th conference on Computational Methods for Systems Biology*, volume 10545 of *Lecture Notes in Computer Science*, pages 370 – 316. Springer, 2017.
- [13] Loïc Paulevé and Sylvain Sené. Boolean networks and their dynamics: the impact of updates. In *Systems Biology Modelling and Analysis: Formal Bioinformatics Methods and Tools*. Wiley, 2022.
- [14] Adrien Richard and Elisa Tonello. Attractor separation and signed cycles in asynchronous Boolean networks. *Theoretical Computer Science*, page 113706, 2023.
- [15] Jordan C Rozum, Jorge Gómez Tejeda Zañudo, Xiao Gan, Dávid Deritei, and Réka Albert. Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. *Science Advances*, 7(29), 2021.
- [16] Julian D. Schwab, Nensi Ikonomi, Silke D. Werle, Felix M. Weidner, Hartmut Geiger, and Hans A. Kestler. Reconstructing boolean network ensembles from single-cell data for unraveling dynamics in the aging of human hematopoietic stem cells. *Computational and Structural Biotechnology Journal*, 19:5321–5332, 2021.
- [17] Robert Schwieger and Elisa Tonello. Reduction for asynchronous Boolean networks: elimination of negatively autoregulated components. *arXiv preprint arXiv:2302.03108*, 2023.
- [18] Gautier Stoll, Barthélémy Caron, Eric Viara, Aurélien Dugourd, Andrei Zinovyev, Aurélien Naldi, Guido Kroemer, Emmanuel Barillot, and Laurence Calzone. MaBoSS 2.0: an environment for stochastic boolean modeling. *Bioinformatics*, 33(14):2226–2228, 2017.

- [19] Van-Giang Trinh, Belaid Benhamou, Kunihiro Hiraishi, and Sylvain Soliman. Minimal trap spaces of Logical models are maximal siphons of their Petri net encoding. In *Computational Methods in Systems Biology: 20th International Conference, CMSB 2022, Bucharest, Romania, September 14–16, 2022, Proceedings*, pages 158–176. Springer, 2022.
- [20] Van-Giang Trinh, Kunihiro Hiraishi, and Belaid Benhamou. Computing attractors of large-scale asynchronous boolean networks using minimal trap spaces. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–10, 2022.
- [21] Trinh Van Giang and Kunihiro Hiraishi. An Improved Method for Finding Attractors of Large-Scale Asynchronous Boolean Networks. In *2021 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–9. IEEE, 2021.
- [22] Alan Veliz-Cuba. Reduction of Boolean network models. *Journal of Theoretical Biology*, 289:167–172, 2011.
- [23] Jorge Gómez Tejeda Zañudo, Pingping Mao, Clara Alcon, Kailey Kowalski, Gabriela N. Johnson, Guotai Xu, Jose Baselga, Maurizio Scaltriti, Anthony Letai, Joan Montero, Réka Albert, and Nikhil Wagle. Cell line-specific network models of ER+ breast cancer identify potential PI3ka inhibitor resistance mechanisms and drug combinations. *Cancer Research*, 81(17):4603–4617, 2021.