Computing Bottom SCCs Symbolically Using Transition Guided Reduction

Nikola Beneš, Luboš Brim, Samuel Pastva^(✉), and David Šafránek

Faculty of Informatics, Masaryk University, Brno,
Czech Republic

{xbenes3,brim,xpastva,safranek}@fi.muni.cz

Abstract. Detection of bottom strongly connected components (BSCC) in state-transition graphs is an important problem with many applications, such as detecting recurrent states in Markov chains or attractors in dynamical systems. However, these graphs' size is often entirely out of reach for algorithms using explicit state-space exploration, necessitating alternative approaches such as the symbolic one.

Symbolic methods for BSCC detection often show impressive performance, but can sometimes take a long time to converge in large graphs. In this paper, we provide a symbolic state-space reduction method for labelled transition systems, called *interleaved transition guided reduction* (ITGR), which aims to alleviate current problems of BSCC detection by efficiently identifying large portions of the non-BSCC states.

We evaluate the suggested heuristic on an extensive collection of 125 real-world biologically motivated systems. We show that ITGR can easily handle all these models while being either the only method to finish, or providing at least an order-of-magnitude speedup over existing state-of-the-art methods. We then use a set of synthetic benchmarks to demonstrate that the technique also consistently scales to graphs with more than 2^{1000} vertices, which was not possible using previous methods.

Keywords: Bottom SCC · Symbolic algorithm · Boolean network

1 Introduction

Finding strongly connected components (SCCs) is a basic problem in graph theory. It is impractical or even impossible for large graphs to find SCCs using explicit depth-first search, motivating the study of symbolic SCCs computation. The structure of SCCs in a graph is captured by its quotient graph, obtained by collapsing each SCC into a single node. This graph is acyclic, thus defines a partial order on the SCCs. *Bottom SCCs* (BSCCs) are SCCs corresponding to leaf nodes in the quotient graph (alternatively referred to as *Terminal SCCs*).

Detection of BSCCs is an important problem with many applications. For example, in Markov chains and Markov decision processes, the recurrent states

belong to terminal SCCs [1, 11, 38]. In LTL model checking, the detection of bottom SCCs is used during the decomposition of the property automaton to speed up the model checking procedure [52]. Another example of an application where detection of BSCCs is crucial is detecting non-terminating sections of parallel programs written in C or C++ [55]. In models of dynamical systems, which are of our primary interest, BSCCs correspond to so-called *attractors* that determine the long-term behaviour of the system [43]. Identification of attractors has many important applications, including communication protocols [4, 47], systems biology [31, 40], mathematical physics [26], ecology [54], epidemiology [42], etc. In biology, the possibility of reaching a particular phenotype of a living cell is indicated by the presence of a specific attractor [40]. The knowledge of attractors then unlocks the path towards cell control [33], reprogramming [49] and even regenerative medicine [17]. Consequently, detection of BSCCs is a fundamental task important not only in computer-aided verification but also many other disciplines.

Our motivation to develop a new symbolic approach to find BSCCs comes from the need to handle extremely large symbolic graphs representing *labelled transition systems* that encode the behaviour of complex real-world concurrent processes. In particular, assuming we deal with finite-state systems, such large transition systems are typically generated from models encoded in a compact formalism such as process calculi, Petri nets, Boolean networks [32, 57], their combinations [6] or other higher-level modelling languages. For such transition systems, the limits of general symbolic SCC algorithms also define the limits of realistic applications.

In most cases, the size of a transition system generated from a model is exponential in the number of concurrently interacting entities. For example, in the case of biological systems, the number of entities is typically ranging from several hundred to hundreds of thousands. Despite strong simplifications employed at the side of models, the size of respective transition systems rarely falls below 10^6 states and is usually much bigger [23, 27, 44]. Thus, the need to tackle large transition systems gives us a solid motivation to revisit the algorithmics for BSCC detection.

In general, it is possible to find all BSCCs as a part of a general SCC decomposition algorithm. There is a rich history of research on computing SCCs symbolically. An algorithm based on forward and backward reachability performing $\mathcal{O}(n^2)$ symbolic steps was presented by Xie and Beerel in [59]. Bloem et al. present an improved $\mathcal{O}(n \cdot \log n)$ algorithm in [7]. Finally, an $\mathcal{O}(n)$ algorithm was presented by Gentilini et al. in [25]. This bound has been proved to be tight in [16]. In [16], the authors argue that the algorithm from [25] is optimal even when considering more fine-grained complexity criteria, like the diameter of the graph and the diameter of the individual components. Ciardo et al. [62] use the idea of saturation [20] to speed up state exploration when computing each SCC in the Xie-Beerel algorithm and compute the transitive closure of the transition relation using a novel algorithm based on saturation.

Our approach is motivated by the fact that techniques working very well for full SCC decomposition do not help to sufficiently accelerate BSCC detection. At the same time, some heuristics, such as saturation, can provide a meaningful impact even when using simpler algorithms [62]. The key novelty of our method lies in a heuristic called *transition guided reduction* that filters the state space by reflecting the possibility of transitions to appear in BSCCs. This step allows to remove some states not belonging to any BSCC, and that way reduce the transition system under analysis to be tractable by the modified Xie-Beerel algorithm with saturation [62].

To target specific characteristics of transition systems representing dynamical systems, e.g., those generated by Boolean networks (BNs) [32, 57], several specialised symbolic SCC decomposition methods have been developed. Since systems for our evaluation come primarily from BNs, we also discuss these specialised methods here. A BN consists of Boolean variables, each having a Boolean update function. Update functions change the state of the variables. The semantics of a BN is a transition system where the states are the possible valuations of the variables, and the transitions are induced by the execution of the update functions. Some of the existing algorithms utilise the synchronous update semantics (updates of all variables executed synchronously) that significantly simplifies the problem [24]. However, it is known that synchronous update can produce unrealistic behaviour [37, 53]. Models with asynchronous update (concurrently executed updates of variables) are closer to reproducing the real behaviour [15]. For the evaluation of our method, we consider asynchronous BNs. Various specialised techniques of BSCC detection have been developed for asynchronous BNs, including BDDs [24, 46, 56, 60], optimisation [34, 35], algebraic-based methods [29], SAT [28], answer set programming [45], concurrency theory [14], sampling [61], or network structure decomposition [18, 21]. Moreover, detection of BSCCs is also present as a necessary step in cell reprogramming [41, 49] and cell control [2, 33] based on BNs. To the best of our knowledge, existing methods specialised for asynchronous BNs do not satisfactorily handle huge models (hundreds of variables and beyond). The best state-of-the-art tools [21, 56] are not yet able to robustly work with BNs of such size. We believe that the generally applicable heuristic we propose in this paper can significantly shift the present technology towards massive real-world applications (thousands of variables and beyond).

The main contribution of this paper is a *novel symbolic method for BSCCs detection in state-transition graphs of huge labelled transition systems* for which the problem cannot be handled by existing algorithms. We introduce a novel reduction technique, called *interleaved transition guided reduction (ITGR)*, which aims to enable the use of existing methods by removing large portions of the irrelevant non-BSCC states. The method relies on the observation that BSCCs in real-world systems rarely employ all transition labels available. Therefore, if a state s can fire a transition with a label that is not employed by some BSCC reachable from s , after applying ITGR, s is eliminated. As a result, all paths in the remaining state space only perform transitions with labels employed

by their reachable BSCCs. What makes the method truly competitive is the interleaving of multiple processes, each of which performs the reduction for a different transition label. The completion of faster processes speeds up the remaining parts of the computation, which would be otherwise intractable.

To show the real-world benefits of our method, we use a wide collection of models and compare the prototype implementation of ITGR to the state-of-the-art tool CABEAN [56] as well as to our own implementation of the Xie-Beerel BSCC detection algorithm [62]. In particular, we consider a set of 125 real-world asynchronous BNs selected from publicly available Boolean network repositories, and show that ITGR can easily handle all these models, while either being the only method to finish the computation or providing at least an order-of-magnitude speedup over existing methods. Additionally, we analyse a set of 200 even larger but structurally similar synthetic BNs, which generate transition systems with approximately 2^{1000} states. We show that ITGR is the only method that can consistently handle systems of this magnitude.

2 Preliminaries

To represent a wide variety of large discrete systems, we consider the abstraction generally known as *labelled transition systems*:

Definition 1. Let $\mathcal{L}$ be a non-empty set of labels. A labelled transition system over $\mathcal{L}$ is a pair $T = (S, \{\xrightarrow{a} \mid a \in \mathcal{L}\})$, where S is a finite non-empty set of states, and for each $a \in \mathcal{L}$, $\xrightarrow{a} \subseteq S \times S$ is a transition relation.

When $(s, s') \in \xrightarrow{a}$, we write $s \xrightarrow{a} s'$, and when $(s, s') \in \xrightarrow{a}$ for some $a \in \mathcal{L}$, we simply write $s \rightarrow s'$. When there is a path $s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$, we write $s_1 \rightarrow^* s_n$. Each labelled transition system T can be seen as a directed *state-transition graph* $G_T = (S, E)$, whose vertices are the states of T and whose edges are given by the transition relations, i.e. $(s, s') \in E \iff s \rightarrow s'$. This formalism can naturally describe a wide variety of modelling frameworks with built-in nondeterminism, such as Petri nets, Boolean networks, or multi-valued regulatory networks.

We assume to have a symbolic representation of a labelled transition system that allows us to perform symbolic set operations on the subsets of S (union $\cup$, intersection $\cap$, difference $\setminus$, subset test $\subseteq$, pick an element PICK, etc.) as well as apply the following operations using the associated transition relations:

$$\begin{aligned} \text{POST}(a, X) &= \{s' \in S \mid \exists s \in X. s \xrightarrow{a} s'\} \\ \text{PRE}(a, X) &= \{s \in S \mid \exists s' \in X. s \xrightarrow{a} s'\} \\ \text{CANPOST}(a, X) &= \{s \in X \mid \exists s' \in S. s \xrightarrow{a} s'\} \end{aligned}$$

We further assume a symbolic complexity model where the complexity of each such operation is in $\mathcal{O}(1)$. Additionally, we use the notation $\text{ALLPOST}(X) = \bigcup_{a \in \mathcal{L}} \text{POST}(a, X)$ for *all successors* and $\text{ALLPRE}(X) = \bigcup_{a \in \mathcal{L}} \text{PRE}(a, X)$ for *all*

predecessors. However, the symbolic complexity of these operations is in $\mathcal{O}(|\mathcal{L}|)$. Finally, we assume that the labels in $\mathcal{L}$ are sorted based on the order in which they influence the variables in the symbolic representation (as in, for example, an ordered binary decision diagram [10]). As a consequence, we index the labels and write $\mathcal{L} = \{a_1, \dots, a_{|\mathcal{L}|}\}$.

Now let us recall a few basic definitions from graph theory in order to define the BSCC detection problem for labelled transition systems:

Definition 2. Let $G = (V, E)$ be a directed graph. A subset $C \subseteq V$ is a strongly connected component (SCC) of G iff it is a maximal subset such that for all pairs $v, v' \in C$, there is a path from v to v' .

A strongly connected component C is called bottom (or terminal, BSCC in the following) when there is no edge going from any $v \in C$ to any $v' \in V \setminus C$.

For a given $s \in V$, we write $SCC(s)$ to denote the strongly connected component that contains s . Furthermore, we say that a set X is SCC-closed when $X = \bigcup_{x \in X} SCC(x)$. This means that every SCC of G is either included in X , or is completely disjoint with X . As an example, a set of all reachable vertices from any given initial set is SCC-closed. When $|SCC(x)| = 1$ and $(x, x) \notin E$, the SCC is called *trivial*.

For a set $X \subseteq V$, we sometimes use the term *the basin of* X to denote the set of all the vertices that have a path to a state in X , formally $basin(X) = \{u \mid \exists v \in X : u \rightarrow^* v\}$. Note that although the name is motivated by the notion of attractor basins in dynamical systems, we use it in a more generalised form here, i.e. we do not require X to be a BSCC.

Problem 1. Let T be a labelled transition system and G_T its corresponding state-transition graph. The problem of bottom strongly connected component detection (BSCC detection) is to identify all subsets of S that correspond to the bottom strongly connected components of G_T .

A detailed analysis of optimal symbolic asymptotic complexity of a full SCC decomposition can be found in [16]. However, the authors in [16] use a slightly different complexity model, where operations like ALLPOST and ALLPRE also assume $\mathcal{O}(1)$ complexity. However, their observations about the relationship between problem complexity and graph (or component) diameter are very relevant.

3 Basic Symbolic BSCC Detection

First, we discuss a BSCC detection algorithm from [62], which will form our baseline going forward. In [62], the authors discuss several symbolic approaches to SCC and BSCC computation, as well as fair cycle detection using various symbolic algorithms, and compare them on large systems from computer science.

In particular, the paper points out that more complex approaches, like the lock-step method [7], work well for full SCC decomposition but do not bring much benefit to the detection of BSCCs. However, the authors do highlight the benefits

Algorithm 1: Basic BSCC detection algorithm with saturation.

```

1 Function BSCC(universe  $\subseteq S$ )
2   while universe  $\neq \emptyset$  do
3     pivot  $\leftarrow$  PICK(universe);
4     basin  $\leftarrow$  BWD*({pivot}, universe);
5     forward  $\leftarrow$  {pivot};
6     repeat
7       (fixpoint, forward)  $\leftarrow$  FWD(forward, universe);
8     until fixpoint or forward  $\not\subseteq$  basin;
9     if forward  $\subseteq$  basin then
10      OUTPUT(forward);
11    universe  $\leftarrow$  universe  $\setminus$  basin;

12 Function BWD*(reachable, universe)
13   repeat
14     (fixpoint, reachable)  $\leftarrow$  BWD(reachable, universe);
15   until fixpoint;
16   return reachable;

17 Function BWD(reachable, universe)
18   for  $i \in |\mathcal{L}| \dots 1$  do
19     pre  $\leftarrow$  universe  $\cap$  PRE( $a_i$ , reachable);
20     if pre  $\not\subseteq$  reachable then
21       return (false, reachable  $\cup$  pre);
22   return (true, reachable);

```

of basin saturation [19] as a heuristic to speed up the state space search. What we present here is therefore the Xie-Beerel algorithm [59], adapted to BSCC detection with saturation, based on the notes from [62] (we have rewritten the pseudocode to better match the presentation style and background of this paper, though).

The method is summarised in Algorithm 1, which shows the main procedure (BSCC) as well as the reachability procedures BWD and BWD*, which we also use in the later sections. We omit the pseudocode for FWD and FWD*, as they are identical to the BWD case, only swapping PRE for POST.

Reachability and Saturation. The forward and backward reachability procedures are divided into two methods each, FWD, BWD, FWD* and BWD*. Since they are functionally symmetrical, we only explicitly discuss backward reachability, with everything directly translating to forward reachability as well.

BWD performs a single backward reachability step and returns the new set of states together with an indication of whether a fixed point has been reached (i.e. whether no new states have been discovered). Note that in classical saturation, once BWD selects some a_i , it is typically applied repeatedly. However, for our primary application domain (Boolean networks), multiple subsequent applications of a single transition would not yield any benefit; we thus use this

observation to simplify the pseudocode. In other cases, the recommended approach is to follow [19].

BWD^* then simply wraps BWD into a cycle that actually computes the full fixed point of the `reachable` set. This separation into two sub-procedures allows us to perform reachability step-by-step or even interleave multiple reachability procedures (which will come into play later). Remember that for saturation to work well, the ordering of labels needs to follow the ordering of variables in the symbolic representation.

Xie-Beerel Algorithm. The main algorithm relies on the well-known observation that for a fixed `pivot` vertex, the SCC of this vertex can be computed as the intersection of vertices forward and backward reachable from `pivot`. When searching for BSCCs, we can easily extend this with two extra observations: First, `pivot` is in a BSCC when only the SCC itself is forward-reachable from `pivot`. Second, a vertex backward-reachable from `pivot` is either in the same SCC as `pivot` (in which case it is in a BSCC iff `pivot` is in a BSCC), or it is not in a BSCC.

Based on these two extra observations, the original algorithm is modified in two ways: First, not just the SCC around `pivot`, but all backward-reachable vertices are eliminated at the end of each iteration. Second, the backward reachability from `pivot` is computed in full, as these are the vertices we can eliminate. However, the forward reachability is terminated early if it leaves the backward-reachable set, since this implies that `pivot` does not belong to a BSCC.

The asymptotic complexity of this algorithm (in terms of symbolic operations) is $\mathcal{O}(|\mathcal{L}| \cdot |S|)$, which follows from the fact that every vertex will appear in `basin` exactly once but may need $\mathcal{O}(|\mathcal{L}|)$ operations to be discovered. Note that optimal symbolic algorithms for BSCC detection are expected to have linear asymptotic complexity. That is, however, assuming a model where `ALLPOST` is an $\mathcal{O}(1)$ operation, not $\mathcal{O}(|\mathcal{L}|)$. This may be reasonable for some (in particular synchronous) systems, but as demonstrated in [19], saturation is typically more effective in practice, even though it is not asymptotically optimal in this model.

In [62], the authors show very impressive performance numbers for this simple algorithm. However, there are two drawbacks, which we believe can be improved significantly. And as we demonstrate in the evaluation, while powerful, this algorithm certainly has limits on some real-world models.

First, the performance of this method is directly tied to the selection of the `pivot` vertex. If the BSCCs of the graph are relatively small, the probability of picking a right pivot is also tiny (remember, even an SCC with 2^{100} vertices is only a minuscule fraction of a graph with 2^{1000} vertices). As a consequence, the algorithm may require a lot of pivots to explore the entire graph. Second, the overall complexity is limited by the diameter of the whole graph instead of the diameter of the BSCCs. Even if the `pivot` is picked perfectly, the algorithm still has to explore each BSCC's whole basin sequentially. To some extent, this is inevitable; however, as we hope to demonstrate in the next section, it is not always necessary.

Algorithm 2: Core reduction principle

```

1 Function REDUCE(pivots, universe)
2   forward  $\leftarrow$  FWD*(pivots, universe);
3   extendedComponent  $\leftarrow$  BWD*(pivots, forward);
4   bottom  $\leftarrow$  forward  $\setminus$  extendedComponent;
5   if universe  $\neq$  forward then
6     basin  $\leftarrow$  BWD*(forward, universe);
7     universe  $\leftarrow$  universe  $\setminus$  (basin  $\setminus$  forward);
8   if bottom  $\neq \emptyset$  then
9     basin  $\leftarrow$  BWD*(bottom, universe);
10    universe  $\leftarrow$  universe  $\setminus$  (basin  $\setminus$  bottom);
11  return universe;

```

To sum up, Algorithm 1 is a powerful tool for the detection of BSCCs. However, it performs best in graphs where the BSCCs either form a large portion of the state space or have basins of small diameter, allowing the algorithm to converge quickly.

4 Transition Guided Reduction

Fig. 1. Example of transition guided reduction. Square nodes show the **pivots** set used for this reduction (in this case, the states that can fire transition a). Double-drawn states are the BSCCs of the graph. The green area then shows the **extendedComponent** induced by the two a transitions, and the blue area is the **bottom** set. The striped states are the basins of the two sets, which are eliminated in this reduction. (Color figure online)

In this section, we introduce a technique that we call *transition guided reduction* (TGR) to eliminate a large portion of non-BSCC states. Algorithm 1 can then perform much better on this reduced state space.

We present the technique in two steps: First, in Algorithm 2, we present the core principle of the reduction procedure and prove its correctness. This approach is generally applicable to any directed graph. Then in Algorithm 3 we show how to apply Algorithm 2 in the context of a labelled transition system. Here, we can exploit the knowledge of the transition labels to guide the reduction.

The reduction principle is described in Algorithm 2 and illustrated in Fig. 1. Given a set of **pivot** states and the current **universe** of all considered states, the method starts by computing **forward**—the set of all states reachable from the **pivot** states. Using this **forward** set, we then compute the **extendedComponent** of the given **pivot** states. Formally, an extended component of set X is a subset $X' \subseteq S$ that contains all states from X , as well as all paths between the states in X . It is a superset of the union $\bigcup_{x \in X} SCC(x)$ but also contains all paths (and SCCs on these paths) that lead between the elements of this union.

We can observe the following properties:

- The **forward** set is SCC-closed, as it is the result of a reachability procedure. Thus any state that can reach but is not contained in **forward** is not a part of any BSCC.
- The set **bottom** (i.e. $\text{forward} \setminus \text{extendedComponent}$) is also SCC-closed (as it is the difference of two SCC-closed sets). Notice that if this set is not empty, it must contain at least one BSCC, and also that any state that can reach **bottom** but is not contained in it is necessarily not a part of any BSCC.

The algorithm then computes the two sets of states that definitely do not contain a BSCC according to these observations and discards these sets from the state space. This is done on lines 5–7 and 8–10, respectively.

Now we can formulate the following theorem:

Theorem 1. *If state $s \in \text{universe}$ is discarded by Algorithm 2, then it is not part of any BSCC.*

Proof. The proof follows from the two previous observations. If the state s is removed on line 7, it means s can reach a state $s' \in \text{forward}$. Since the set **forward** is SCC-closed, we get $SCC(s) \neq SCC(s')$. State s therefore does not belong to a BSCC.

Similarly, if the state s is removed on line 10, then it means s can reach a state $s' \in \text{bottom}$, and again due to the fact that **bottom** is SCC-closed, the state s does not belong to a BSCC. $\square$

However, this does not provide any guidance as to which **pivots** should we select for the reduction or why. This is addressed in Algorithm 3. Here, we go through all the available transitions $a \in \mathcal{L}$ and select as the **pivots** the set of all the states that can fire a (notice that **pivots** in Fig. 1 also correspond to such states). As a result, all BSCCs that use a are contained in **extendedComponent** and all BSCCs that do not use a , but a is performed in their basin are contained in the **bottom** set. This effectively separates the BSCCs based on the transitions that they use.

Algorithm 3: Transition Guided Reduction

```

1 Function TGR( $\text{universe} \subseteq S$ )
2   for  $a \in \mathcal{L}$  do
3      $\text{universe} \leftarrow \text{REDUCE}(\text{CANPOST}(a, \text{universe}), \text{universe});$ 
4     if  $\text{CANPOST}(a, \text{universe}) = \emptyset$  then
5        $\mathcal{L} \leftarrow \mathcal{L} \setminus a;$ 
6   return  $\text{universe};$ 

```

Finally, notice that if all transitions of a certain type are eliminated, we remove a from $\mathcal{L}$ completely. In large systems, this can significantly reduce the overhead of the FWD/BWD procedures that have to iterate through $\mathcal{L}$.

To better describe the cases in which this reduction works well, let us first formally define the following:

Definition 3. *Given a labelled transition system T and a state s , the fire set $F(s)$ is the subset of transition labels $F(s) \subseteq \mathcal{L}$ that can be fired in state s , i.e. $a \in F(s) \Leftrightarrow \exists s' \in S : s \xrightarrow{a} s'$. A transitive fire set $F^*(s)$ is the union of all the fire sets $F(s')$ of all the states s' reachable from s (i.e. $s \rightarrow^* s'$).*

Notice that for any two states such that $s \rightarrow^* s'$, it holds that $F^*(s') \subseteq F^*(s)$. This also means that in any SCC, the transitive fire set of all states is the same.

Theorem 2. *Let s be an arbitrary state and s' be a state of a BSCC such that $s \rightarrow^* s'$. If $F^*(s) \neq F^*(s')$, Algorithm 3 discards the state s .*

Proof. Since $s \rightarrow^* s$ and $F^*(s) \neq F^*(s')$, it follows that $F^*(s') \subset F^*(s)$. Let $a \in F^*(s) \setminus F^*(s')$ be arbitrary and let us consider the iteration of the main loop when a is selected. Assume that s has not been discarded in any of the previous iterations (otherwise, the proof is already finished). Let E be the `extendedComponent` computed in the current iteration. Then s is either in E , or s can reach E , because $a \in F^*(s)$.

If $s \notin E$, but s can reach E , then s is eliminated on line 7 of Algorithm 3 as part of the `forward` basin. When $s \in E$, it holds that $s' \in \text{bottom}$, since $a \notin F^*(s')$. However, since $s \rightarrow^* s'$, we know that s is removed on line 10 because it belongs to the basin of the `bottom` set. $\square$

However, note that the other implication does not hold. That is, these are not the *only* states that Algorithm 3 eliminates (this can be also seen in Fig. 1).

Based on this theorem, we can derive two extra observations which help to explain the effectiveness of the reduction:

Corollary 1. *If a transition system T has a trivial BSCC, then the whole basin of this SCC is discarded by Algorithm 3.*

Corollary 2. *If a state s is not discarded by Algorithm 3, then all paths starting in s in the reduced state space only use the same transitions as contained in BSCCs reachable from s .*

The first corollary follows from the fact that $F^*(s) = \emptyset$ iff s is a trivial BSCC. The second corollary is essentially a rephrasing of Theorem 2, but it highlights an important property of the reduction: if some transition label is not used by a BSCC, all states in its basin that use it will be eliminated. In our experience, real-world systems rarely use *all* available labels in *all* BSCCs (unless most of the state space is just a single large BSCC). Thus by using this pre-processing step, we can greatly simplify the work of Algorithm 1 by pruning “easily identifiable” non-BSCC states.

There is one more point to be made here: Algorithm 1 has to walk the entire depth of the BSCC basins, which can be substantial. Meanwhile, our approach can often “skip ahead” because it is not starting from a single `pivot` but rather a larger subset of states. However, this may not always be sufficient. In practice, some transitions may be much harder to reduce than others. We address this problem in the next section.

5 Interleaved Transition Guided Reduction

While TGR can significantly reduce the number of states Algorithm 1 needs to consider, TGR itself iterates transitions in an arbitrary order which can significantly influence the speed and number of steps the reduction needs to perform. Removing a transition potentially reduces the number of states which subsequent reductions need to consider. It is thus beneficial to perform the “easiest” reductions first, as this can greatly simplify the following “harder” cases.

However, determining which reductions are “easy” and which are “hard” is not a simple problem. We could try to use additional structural information about the system to determine this, but that would limit us to a specific subclass of models. Instead, we let the algorithm determine this dynamically on the fly.

Our approach is summarised in Algorithm 4. Instead of reducing one transition relation at a time, we interleave all reductions in one procedure. This is done by creating a number of *processes*, one per each $a \in \mathcal{L}$, that we run in an interleaving fashion. The processes work in two phases: `FORWARD` and `EXTENDED COMPONENT`. The goal of a process in the `FORWARD` phase is to compute the value of the `forward` set starting from the states that enable an a -transition, and then switch to the `EXTENDED COMPONENT` phase, in which the goal of the process is to compute the corresponding `extendedComponent` set. The computation proceeds using the one-reachability-step functions `FWD` and `BWD`, which we defined in Algorithm 1. Every process has its process variables that are local to each process, but their values are kept between steps: The set `reach` represents the part of `forward` that has already been discovered; the set `component` represents the part of `extendedComponent` that has already been discovered; the variable `weight` is explained below; and the variable `phase` holds the current phase of the process (`FORWARD` or `EXTENDED COMPONENT`).

The process selected for execution in each iteration (line 31) is the one with the smallest *weight*. The weight of a process is determined by the size of the symbolic representation of the set it is currently expanding (`reach` in the first

Algorithm 4: Interleaved Transition Guided Reduction

```

1  Process ITGRWORKER( $a \in \mathcal{L}$ )
   process variables: reach, component, weight, phase
   shared variables :  $\mathcal{L}$ , universe, processes
2  initialisation:
3      reach  $\leftarrow$  CANPOST( $a, \text{universe}$ );
4      weight  $\leftarrow$  NODECOUNT(reach);
5      phase  $\leftarrow$  FORWARD;
6  step if phase is FORWARD:
7      (fixpoint, reach)  $\leftarrow$  FWD(reach  $\cap$  universe, universe);
8      weight  $\leftarrow$  NODECOUNT(reach);
9      if fixpoint then
10         if universe  $\neq$  reach then
11             basin  $\leftarrow$  BWD*(reach);
12             universe  $\leftarrow$  universe  $\setminus$  (basin  $\setminus$  reach);
13         component  $\leftarrow$  CANPOST( $a, \text{universe}$ );
14         weight  $\leftarrow$  NODECOUNT(component);
15         phase  $\leftarrow$  EXTENDED COMPONENT;
16  step if phase is EXTENDED COMPONENT:
17      (fixpoint, component)  $\leftarrow$  BWD(component, reach  $\cap$  universe);
18      weight  $\leftarrow$  NODECOUNT(component);
19      if fixpoint then
20         bottom  $\leftarrow$  (reach  $\cap$  universe)  $\setminus$  component;
21         if bottom  $\neq \emptyset$  then
22             basin  $\leftarrow$  BWD*(bottom, universe);
23             universe  $\leftarrow$  universe  $\setminus$  (basin  $\setminus$  bottom);
24         if CANPOST( $a, \text{universe}$ ) =  $\emptyset$  then
25              $\mathcal{L} \leftarrow \mathcal{L} \setminus \{a\}$ ;
26         stop current process (remove it from processes);
27 Function ITGR(universe  $\subseteq S$ )
28     processes  $\leftarrow$  {ITGRWORKER( $a$ ) |  $a \in \mathcal{L}$ };
29     initialise all processes;
30     while processes  $\neq \emptyset$  do
31         p  $\leftarrow$  MINBYKEY(processes, weight);
32         run one step of p;
33     return universe;

```

phase, component in the second). For BDDs (or MDDs), this is the number of nodes in the decision diagram (NODECOUNT). The algorithm thus prioritises processes that have the potential to advance quickly because they will use fast symbolic operations.

Notice that the **universe** variable is shared by all processes and needs to be now taken into account in multiple places. This means that whenever one process discards some states from **universe**, all processes benefit from this change. This

update can be performed safely because whenever Algorithm 4 discards some states, the discarded set is SCC-closed.

Because both Algorithm 3 and Algorithm 4 compute the same states for **forward** and **extendedComponent** (modulo the states eliminated by other reductions), Theorem 1 and Theorem 2 remain valid for Algorithm 4 as well. The only difference is that this approach should be much more resilient to a bad ordering of transition relations.

Finally, let us note that this approach should be quite simple to parallelise to some extent. If w parallel workers are available, the algorithm can advance w processes at a time instead of picking a single process. However, we do not pursue this approach in this paper as the other methods we use as a reference are also not parallelised.

6 Evaluation

To see how ITGR affects the performance of attractor detection for real-life systems, we implemented the method for asynchronous Boolean networks (BN), a common logical modelling framework used predominantly in systems biology.

Using this implementation, we aim to support the following claims:

1. ITGR performs significantly better than available state-of-the-art tools (for Boolean networks) on real-world models.
2. On realistic Boolean networks, ITGR easily scales to 1000 or more variables, which is not possible with other methods.
3. Interleaving plays a crucial role in making ITGR competitive.

To evaluate the first claim, we compare our implementation with the tool CABEAN [56] on a set of 125 real-world Boolean networks with up to 350 variables. We then generate a pseudo-random set of 200 networks with similar structural characteristics to our real-world benchmarks, but with up to 1100 variables. We show that ITGR can successfully deal with models of this magnitude as well. Finally, we compare the performance of ITGR with the basic attractor detection algorithm as well as with “sequential” TGR on both benchmark sets, showing that ITGR is overall faster and is the only method able to handle the large benchmarks efficiently.

The whole set of benchmarks, as well as the implementation of all the algorithms in Rust, is available as a paper artefact at Zenodo¹. Additionally, the method is successfully employed by our tool AEON that facilitates long-term analysis of Boolean networks [3].

Before we present the actual benchmark results, let us also first briefly comment on the modelling paradigm chosen (Boolean networks) and the actual setup used to perform the measurements.

¹ <https://doi.org/10.5281/zenodo.4709882>.

6.1 Boolean Networks

A Boolean network, as the name suggests, consists of n Boolean variables, each variable having an associated update function b_i . The state space of the network consists of 2^n Boolean variable valuation vectors, $\{0, 1\}^n$. Each update function takes the current state of the network and produces a new value that is assigned to the associated variable, $b_i : \{0, 1\}^n \rightarrow \{0, 1\}$. We assume the update functions can be applied non-deterministically, resulting in an asynchronous updating scheme. This is not the only updating scheme used in practice (e.g. synchronous or generalised asynchronous can be used as well) but is generally considered to cover the possible behaviour of the biological system well.

Typically, an update function of a particular variable x only depends on a smaller subset of the system variables. In such a case, we say that these variables *regulate* x (specifically, y regulates x if the update function of x depends on the value of y). The number of such *regulations* in a Boolean network can be viewed to represent the connectedness or structural complexity of the network in general. In short, the more regulations the network has, the more complex update functions it contains, possibly resulting in more complex behaviour. Variables and regulations together form a directed *regulatory graph*.

A Boolean network with an asynchronous updating scheme fits naturally into our definition of a labelled transition system. The state space of the network variables corresponds with S , i.e. $S = \{0, 1\}^n$. Each transition a_i of $\mathcal{L}$ corresponds to the application of the i -th Boolean update function b_i to the i -th Boolean variable, i.e. $(s, s') \in a_i \Leftrightarrow s' = s[i \leftarrow b_i(s)] \wedge s' \neq s$.

When dealing with Boolean networks, BSCCs are typically referred to as *attractors*. The rationale behind this term is that the BSCCs are the states where the fair runs of any system eventually converge to—the behaviour is thus *attracted* towards these states. In the following, we use these two terms interchangeably.

As a symbolic representation, the most natural choice for Boolean networks are Reduced Ordered Binary Decision Diagrams (ROBDD, or BDD) [10], as these can be easily used to represent sets of Boolean vectors. We do not make any specific optimisations with regards to variable ordering, but to enable saturation-like reachability, we assume that the ordering of transitions $a_1, \dots, a_n$ follows that of the variables in the ROBDD that they update.

Since a Boolean network consists of n Boolean variables, a set of states of such a network can be seen as a Boolean formula (represented as a BDD) over the network variables. Here, a state belongs to such a set iff it represents a satisfying assignment of this formula – a fairly standard approach to state-space encoding using BDDs. To apply a particular update function, we must first construct a BDD describing all states where the update function should change the value of its associated variable (note that this BDD can be reused in subsequent steps). By computing an intersection of a state set with this BDD (yielding the result of the CANPOST operation) and then performing a “bit flip” of the updated variable in the result BDD, we obtain a set of successor states with respect to this one update function (i.e. POST). Similarly, we can obtain CANPRE and PRE.

6.2 Benchmark Set-Up

Real-world Models. To provide the best possible real-world evaluation of our method, we have collected all the models from publicly available Boolean network repositories that we were aware of, and that support the universally accepted SBML-qual format [12] for model transfer. Specifically, our benchmark set includes models available through GINsim [13], Cellcollective [30], Biomod-els [39] and the COVID-19 disease map project [48]. Together, the benchmark set consists of 125 models, peaking at 351 variables and 1100 regulations, respectively.

Note that some of these models contain Boolean constants (also called inputs or parameters) that can be specified by the user. For such models, we performed a simple parameter sampling to determine if some of the valuations result in non-trivial attractors, as these are the main focus of this paper. If such valuation was found, we have used it in our benchmark set. However, for the vast majority of models (approximately 90%), there were either no tunable parameters or the sampling did not find any significant changes in the structure of attractors.

Environment. We ran all the benchmarks on a machine with a modest 4-core i7 4790 and 32 GB of RAM. However, none of the benchmarks used more than one core at a time, and typical memory consumption was significantly below 1 GB. Hence our evaluation should be reproducible even using a much slower machine.

We have measured the runtime for each model using the standard Unix `time` utility, with a one-hour timeout per benchmark model. We have run a large portion of the benchmarks repeatedly but have not observed any significant variance in runtime; we thus only report average runtime values.

CABEAN. In the real-world performance test, we compare our method to the tool CABEAN [56]. To the best of our knowledge, CABEAN is both the most recent and the most advanced tool that targets the detection of non-trivial attractors in asynchronous Boolean networks. Other tools that we know of (such as [14, 21, 36]) are not built for systems of the size we are dealing with (for example, due to explicit state-space representation). CABEAN focuses on Boolean network reprogramming, but as a necessary component of this process, it also provides state-of-the-art methods for attractor detection. Specifically, CABEAN uses symbolic manipulation using BDDs, just as our method, but implements advanced decomposition techniques [50] to reduce the state space of the network.

6.3 Real-World Networks

The core of our results is summarised in Fig. 2. On the left, we see a comparison of total successfully completed benchmarks by both CABEAN and ITGR, and on the right, we have relative speedup for each individual benchmark. On the right, we only show benchmarks that took CABEAN more than one second to complete (remaining models would be normally easy to compute even without any special techniques).

Fig. 2. The left plot shows the total number of benchmarks that each tool has completed before a certain time limit. The dashed line represents CABEAN, whereas the solid line shows our ITGR implementation. On the right, the graph displays the relative runtime between CABEAN and ITGR. The dotted lines represent 10x and 100x speedup compared to CABEAN. The solid circles are the benchmarks where CABEAN successfully computed the attractors. The crosses represent the benchmarks where CABEAN was able to finish the decomposition but failed to extract the actual attractors. Notice that we use logarithmic scaling for the time in both graphs.

In this test, ITGR completed all but one benchmark in less than 1 min. The one remaining case took almost 15 min to complete. However, the reduction process for this model was also quite fast at roughly 100 s. The rest of the computation was spent on identifying the 352 non-trivial attractors in the remaining state space (together, the attractors account for almost 2^{85} states – by far the largest we have seen in any model). Out of the 125 benchmarks, we uncovered non-trivial attractors in exactly 40 models (however, this also includes 6 models with only small 2- or 4-state attractors).

On the other hand, CABEAN failed to compute attractors for 19 of these 125 models (15.2%). Upon closer inspection, all but one of these 19 benchmarks contained non-trivial attractors, which means CABEAN failed for 45% of models with non-trivial attractors.

However, we note that on some models, CABEAN did not simply timeout but actually terminated early due to a segfault. While this behaviour does not seem to be directly linked to the total size of the attractors, it certainly appears to be more common in such networks. We have seen this happen in networks with relatively small attractors, while other networks (even one with a 2^{30} -state attractor) were completed successfully. We hypothesise that this occurs

when the decomposition is (at least partially) successful but does not reduce the complexity of the network enough to continue with attractor search².

These failed attempts are visualised in the right plot using crosses, as they represent an interesting lower-bound approximation on the performance of this decomposition technique. Overall, the plot shows that for the vast majority of models, ITGR provides an order of magnitude (10x) speedup compared to CABEAN, with some (especially larger) models attacking or exceeding the 100x speedup threshold.

Overall, we have shown that ITGR is capable of solving all publicly available problem instances (that we know of), and it outperforms current state-of-the-art decomposition methods with the median of 16x speedup (77x average). Naturally, we also compared the actual attractors found by CABEAN and ITGR, and we are happy to report that we found no inconsistencies between the two methods.

6.4 Pseudo-random Networks

Next, we set out to test the limits of ITGR on larger models than the ones available in the public repositories today. Specifically, we wanted to test networks with 1000 or more Boolean variables. While such a number is arguably not possible to achieve in a single hand-made model, fully or semi-automated machine learning techniques [5, 8, 22, 51] are making models of this magnitude much more approachable.

Pseudo-Random BNs. To create a benchmark set of larger models, we have decided to generate pseudo-random networks structurally similar to our real-world benchmark set. Biological systems, specifically protein and gene regulatory networks, are known to follow certain properties of small-world networks [9, 58]. However, aside from other differences, they are directed and typically quite sparse (our real-world benchmark set has the average node degree of 4.3). This makes most common random network models unsuitable for this specific task. For example, the famous Watts–Strogatz model would, in this case, assume that the average degree is significantly larger than $\ln(1000) \sim 7$.

We have thus first measured the relative in- and out-degree distributions in the regulatory graphs of our real-world networks and then generated random networks by sampling from this distribution to approximate the real-world dataset. Additionally, regulatory graphs of Boolean networks are essentially always weakly connected. In each model, we have thus filtered out all variables except the largest weakly connected component. Note that this makes the dataset slightly skewed towards more connected networks (i.e. more challenging), as these have a higher chance of being weakly connected when randomly generated. However, it is still well within the connectivity limits expected based on the real-world dataset.

² Developers of CABEAN have confirmed that this assumption is essentially accurate, i.e. for some structurally complex networks with large non-trivial attractors, the tool can segfault instead of timing-out.

To generate the boolean functions, we have measured that 80.7% of the regulations in the real-world dataset are positively monotonous, with the remaining being negatively monotonous (monotonicity is typically expected in biological networks). Each regulation was thus assigned monotonicity based on this distribution, and a function was generated by randomly choosing between $\wedge$ and $\vee$ when connecting the positive/negative literals. Note that this does not cover the full spectrum of possible Boolean functions, but it is well within reason for biological networks, where some techniques tend to even implicitly assume the function is just a simple conjunction/disjunction of literals.

Fig. 3. Runtime comparison of ITGR, “sequential” TGR and the basic symbolic BSCC detection. The left plot shows the real-world benchmark set with up to 350 variables per model. On the right, we see the *medium* synthetic benchmark ranging from 50 to 1100 variables. Note that the *large* synthetic benchmark (~ 1000 variables only) is not shown, as ITGR was the only method capable of actually completing these models. All the time axes have a logarithmic scale.

Performance. In the end, we have obtained two benchmark sets: A *medium* set with 100 networks ranging from 50 to 1100 variables, and one *large* benchmark set, also with 100 models, but all with ~ 1000 variables and ranging from 2471 to 5099 regulations. Out of these 200 models, we discovered non-trivial attractors in 61 of them.

The runtime for the *medium* benchmarks is summarised in Fig. 3 (right). Here, we see that ITGR successfully completed all instances within 10 min. For the *large* benchmark set, ITGR consistently finished 98% of the models within 5 to 10 min. The remaining two outliers took 28 and 55 min to complete. Similar to what we saw in the real-world benchmark, these models contained very large non-trivial attractors (the largest having again more than 2^{80} states) and were thus not limited by the speed of the reduction but by the diameter of the actual attractors.

Additionally, for each reduction procedure, we kept track of the actual number of reachability iterations that needed to be performed (specifically, how many times we applied line 21 of BWD as shown in Algorithm 1, or the same line in FWD). For all models, this number was well below 10 000 iterations, which is quite low considering the procedure needed to evaluate up to 1100 transition relations. In particular, this supports our hypothesis that ITGR works well due to typically short distances between states that can fire individual transitions.

6.5 Interleaving Performance Impact

Finally, we would like to evaluate the influence of smart interleaving on the performance of ITGR. For this purpose, we consider three algorithms:

1. the basic symbolic algorithm with saturation as described in Sect. 3;
2. TGR, as described in Sect. 4, applied to variables in the order in which they appear in the network declaration (that is, without any interleaving);
3. the full ITGR as described in Sect. 5.

Keep in mind that all three approaches use the same implementation of symbolic representation and differ only in the actual attractor detection. Also, TGR/ITGR use the basic algorithm to actually identify attractors once the reduction is completed. Any speedup between TGR and the basic algorithm can be thus directly attributed to the state-space reduction, while any speedup between ITGR and TGR is due to the introduction of interleaving.

For the real-world benchmarks (up to 350 variables) and medium synthetic benchmarks (50 to 1100 variables), the comparison is presented in Fig. 3. Here, we see that the basic algorithm is indeed not generally sufficient for large networks, finishing only 62 of the 125 real-world models and only 5 of the medium synthetic benchmarks (the main reason was typically poor pivot selection; however, some instances also timed out due to long reachability procedures).

The difference between TGR and ITGR is not as drastic for real-world models. TGR finished in 122/125 instances but was consistently slower than ITGR, especially on larger models (on one instance, we have even seen a 55 min vs 2.9 s speedup, i.e. more than 1000x). However, as we look into even larger graphs with the medium synthetic benchmark set, ITGR easily outperforms TGR, which completed only 26/100 instances.

Finally, for the large benchmarks, all with ~ 1000 variables, we have ITGR completing all benchmarks within the 1-h timeout (with 98% finishing within 10 min); no other method has finished any of the 100 models within this limit. This leaves ITGR as the only implementation in this comparison capable of successfully analysing networks with 1000 or more Boolean variables.

7 Conclusions

In this paper, we present a novel symbolic method for BSCC detection in state-transition graphs of labelled transition systems, called *interleaved transition*

guided reduction (ITGR). The method relies on the observation that BSCCs in real-world systems rarely employ all transition labels available. Therefore, if a state s can fire a transition with a label that is not employed by some BSCC reachable from s , after applying ITGR, s is eliminated. As a result, all paths in the remaining state space only perform transitions with labels employed by their reachable BSCCs. If the system has only trivial BSCCs, this solves the problem completely. For non-trivial BSCCs, this may make the problem tractable using previously known techniques.

ITGR relies on smart interleaving to prioritise the elimination of “symbolically easier” transitions. Completing the reduction in this order allows ITGR to subsequently simplify the analysis of transitions which would initially be too complex to handle.

We tested the method on a large benchmark set of real-world Boolean networks (up to 350 variables) as well as randomly generated benchmarks (up to 1100 variables) with similar structural properties. Our experiments show that ITGR significantly outperforms the state-of-the-art tool CABEAN and can easily handle all models from both benchmark sets, pushing the boundary of what was previously possible in this field.

References

1. Abrahám, E., Jansen, N., Wimmer, R., Katoen, J.P., Becker, B.: DTMC model checking by SCC reduction. In: 2010 Seventh International Conference on the Quantitative Evaluation of Systems, pp. 37–46. IEEE (2010)
2. Baudin, A., Paul, S., Su, C., Pang, J.: Controlling large Boolean networks with single-step perturbations. *Bioinformatics* **35**(14), i558–i567 (2019)
3. Beneš, N., Brim, L., Kadlec, J., Pastva, S., Šafránek, D.: AEON: attractor bifurcation analysis of parametrised boolean networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 569–581. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_28
4. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: Digital bifurcation analysis of TCP dynamics. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 339–356. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_19
5. Berestovsky, N., Nakhleh, L.: An evaluation of methods for inferring Boolean networks from time-series data. *PLoS ONE* **8**(6), e66031 (2013)
6. Berestovsky, N., Zhou, W., Nagrath, D., Nakhleh, L.: Modeling integrated cellular machinery using hybrid Petri-Boolean networks. *PLOS Comput. Biol.* **9**(11), 1–13 (2013)
7. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in *nlogn* symbolic steps. In: Formal Methods in Computer-Aided Design, pp. 37–54. Springer (2000)
8. Bonnaïffoux, A., et al.: WASABI: a dynamic iterative framework for gene regulatory network inference. *BMC Bioinformatics* **20**(1), 1–19 (2019)
9. Bork, P., Jensen, L.J., Von Mering, C., Ramani, A.K., Lee, I., Marcotte, E.M.: Protein interaction networks from yeast to human. *Curr. Opin. Struct. Biol.* **14**(3), 292–299 (2004)
10. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.* **35**(8), 677–691 (1986)

11. Buchholz, P., Katoen, J.P., Kemper, P., Tepper, C.: Model-checking large structured Markov chains. *J. Logic Algebraic Program.* **56**(1), 69–97 (2003)
12. Chaouiya, C., et al.: SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Syst. Biol.* **7**(1), 1–15 (2013)
13. Chaouiya, C., Naldi, A., Thieffry, D.: Logical modelling of gene regulatory networks with GINsim. In: van Helden, J., Toussaint, A., Thieffry, D. (eds.) *Bacterial Molecular Networks*, pp. 463–479. Springer, New York (2012). https://doi.org/10.1007/978-1-61779-361-5_23
14. Chatain, T., Haar, S., Jezequel, L., Paulevé, L., Schwoon, S.: Characterization of reachable attractors using petri net unfoldings. In: Mendes, P., Dada, J.O., Smallbone, K. (eds.) *CMSB 2014. LNCS*, vol. 8859, pp. 129–142. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12982-2_10
15. Chatain, T., Haar, S., Kolčák, J., Paulevé, L., Thakkar, A.: Concurrency in Boolean networks. *Nat. Comput.* **19**(1), 91–109 (2020)
16. Chatterjee, K., Dvořák, W., Henzinger, M., Loitzenbauer, V.: Lower bounds for symbolic computation on graphs: strongly connected components, liveness, safety, and diameter. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2341–2356. SIAM (2018)
17. Cherry, A.B.C., Daley, G.Q.: Reprogramming cellular identity for regenerative medicine. *Cell* **148**(6), 1110–1122 (2012)
18. Choo, S.M., Cho, K.H.: An efficient algorithm for identifying primary phenotype attractors of a large-scale Boolean network. *BMC Syst. Biol.* **10**(1), 95 (2016)
19. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Saturation: an efficient iteration strategy for symbolic state—space generation. In: *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 328–342. Springer (2001)
20. Ciardo, G., Marmorstein, R.M., Siminiceanu, R.: The saturation algorithm for symbolic state-space exploration. *Int. J. Softw. Tools Technol. Transf.* **8**(1), 4–25 (2006)
21. Gan, X., Albert, R.: General method to find the attractors of discrete dynamic models of biological systems. *Phys. Rev. E* **97**, 042308 (2018)
22. Gao, S., Sun, C., Xiang, C., Qin, K., Lee, T.H.: Learning asynchronous Boolean networks from single-cell data using multiobjective cooperative genetic programming. *IEEE Transactions on Cybernetics* (2020)
23. García-Gómez, M.L., Ornelas-Ayala, D., Garay-Arroyo, A., García-Ponce, B., de la Paz Sánchez, M., Álvarez-Buylla, E.R.: A system-level mechanistic explanation for asymmetric stem cell fates: arabidopsis thaliana root niche as a study system. *Sci. Rep.* **10**(1), 1–16 (2020)
24. Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., De Micheli, G.: Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics* **24**(17), 1917–1925 (2008)
25. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 3, pp. 573–582. SIAM (2003)
26. Grebogi, C., Ott, E., Yorke, J.A.: Chaos, strange attractors, and fractal basin boundaries in nonlinear dynamics. *Science* **238**(4827), 632–638 (1987)
27. Grieco, L., Calzone, L., Bernard-Pierrot, I., Radvanyi, F., Kahn-Perles, B., Thieffry, D.: Integrative modelling of the influence of MAPK network on cancer cell fate decision. *PLOS Comput. Biol.* **9**(10), 1–15 (2013)

28. Guo, W., Yang, G., Wu, W., He, L., Sun, M.I.: A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks. *PLOS ONE* **9**(4), 1–10 (2014)
29. Harvey, I., Bossomaier, T.: Time out of joint: attractors in asynchronous random Boolean networks. In: *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, pp. 67–75. MIT Press (1997)
30. Helikar, T., Kowal, B., McClenathan, S., Bruckner, M., Rowley, T., Madrahimov, A., et al.: The Cell Collective: toward an open and collaborative approach to systems biology. *BMC Syst. Biol.* **6**(1), 96 (2012)
31. Huang, S., Ernberg, I., Kauffman, S.: Cancer attractors: a systems view of tumors from a gene network dynamics and developmental perspective. *Seminars Cell Dev. Biol.* **20**(7), 869–876 (2009)
32. Kauffman, S.: Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.* **22**(3), 437–467 (1969)
33. Kim, J., Park, S.M., Cho, K.H.: Discovery of a kernel for controlling biomolecular regulatory networks. *Sci. Rep.* **3**, 2223 (2013)
34. Klarner, H., Bockmayr, A., Siebert, H.: Computing symbolic steady states of boolean networks. In: Waş, J., Sirakoulis, G.C., Bandini, S. (eds.) *ACRI 2014*. LNCS, vol. 8751, pp. 561–570. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11520-7_59
35. Klarner, H., Bockmayr, A., Siebert, H.: Computing maximal and minimal trap spaces of Boolean networks. *Nat. Comput.* **14**(4), 535–544 (2015)
36. Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinformatics* **33**(5), 770–772 (2017)
37. Klemm, K., Bornholdt, S.: Stable and unstable attractors in Boolean networks. *Phys. Rev. E* **72**(5), 055101 (2005)
38. Kučera, A., Stražovský, O.: On the Controller Synthesis for Finite-State Markov Decision Processes. In: Sarukkai, S., Sen, S. (eds.) *FSTTCS 2005*. LNCS, vol. 3821, pp. 541–552. Springer, Heidelberg (2005). https://doi.org/10.1007/11590156_44
39. Le Novère, N., et al.: Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.* **34**(suppl_1), D689–D691 (2006)
40. MacArthur, B.D., Ma’ayan, A., Lemischka, I.R.: Systems biology of stem cell fate and cellular reprogramming. *Nat. Rev. Mol. Cell Biol.* **10**(10), 672–681 (2009)
41. Mandon, H., Su, C., Pang, J., Paul, S., Haar, S., Paulevé, L.: Algorithms for the sequential reprogramming of Boolean networks. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **16**(5), 1610–1619 (2019)
42. Matouk, A.: Complex dynamics in susceptible-infected models for COVID-19 with multi-drug resistance. *Chaos, Solitons Fractals* **140**, 110257 (2020)
43. Mendes, N.D., Henriques, R., Remy, E., Carneiro, J., Monteiro, P.T., Chaouiya, C.: Estimating attractor reachability in asynchronous logical models. *Front. Physiol.* **9**, 1161 (2018)
44. Mizera, A., Pang, J., Qu, H., Yuan, Q.: Taming asynchrony for attractor detection in large Boolean networks. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **16**(1), 31–42 (2019)
45. Mushthofa, M., Schockaert, S., De Cock, M.: Computing attractors of multi-valued gene regulatory networks using fuzzy answer set programming. *FUZZ-IEEE* **2016**, 1955–1962 (2016)

46. Naldi, A., Thieffry, D., Chaouiya, C.: Decision diagrams for the representation and analysis of logical models of genetic networks. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS, vol. 4695, pp. 233–247. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75140-3_16
47. Nga, J., Iu, H.H., Ling, B.W.K., Lam, H.K.: Analysis and control of bifurcation and chaos in average queue length in TCP/RED model. *Int. J. Bifurcation Chaos* **18**(08), 2449–2459 (2008)
48. Ostaszewski, M., et al.: COVID-19 disease map, building a computational repository of SARS-CoV-2 virus-host interaction mechanisms. *Sci. Data* **7**(1), 1–4 (2020)
49. Pardo, J., Ivanov, S., Delaplace, F.: Sequential reprogramming of biological network fate. In: Bortolussi, L., Sanguinetti, G. (eds.) CMSB 2019. LNCS, vol. 11773, pp. 20–41. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31304-3_2
50. Paul, S., Su, C., Pang, J., Mizera, A.: A decomposition-based approach towards the control of Boolean networks. In: International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 11–20 (2018)
51. Razzaq, M., Paulevé, L., Siegel, A., Saez-Rodriguez, J., Bourdon, J., Guziolowski, C.: Computational discovery of dynamic cell line specific Boolean networks from multiplex time-course data. *PLoS Comput. Biol.* **14**(10), e1006538 (2018)
52. Renault, E., Duret-Lutz, A., Kordon, F., Poitrenaud, D.: Strength-based decomposition of the property Büchi automaton for faster model checking. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 580–593. Springer (2013)
53. Saadatpour, A., Albert, I., Albert, R.: Attractor analysis of asynchronous Boolean models of signal transduction networks. *J. Theor. Biol.* **266**(4), 641–656 (2010)
54. Steffen, W., et al.: Trajectories of the earth system in the anthropocene. *Proc. Natl. Acad. Sci.* **115**(33), 8252–8259 (2018)
55. Štill, V., Barnat, J.: Local nontermination detection for parallel C++ programs. In: Software Engineering and Formal Methods. pp. 373–390. Springer, Cham (2019)
56. Su, C., Pang, J.: CABEAN: a software for the control of asynchronous Boolean networks. *Bioinformatics* (2020)
57. Thomas, R.: Boolean formalization of genetic control circuits. *J. Theor. Biol.* **42**(3), 563–585 (1973)
58. Van Noort, V., Snel, B., Huynen, M.A.: The yeast coexpression network has a small-world, scale-free architecture and can be explained by a simple model. *EMBO Rep.* **5**(3), 280–284 (2004)
59. Xie, A., Beerel, P.A.: Implicit enumeration of strongly connected components and an application to formal verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **19**(10), 1225–1230 (2000)
60. Yuan, Q., Mizera, A., Pang, J., Qu, H.: A new decomposition-based method for detecting attractors in synchronous Boolean networks. *Sci. Comput. Program.* **180**, 18–35 (2019)
61. Zhang, S.Q., Hayashida, M., Akutsu, T., Ching, W.K., Ng, M.K.: Algorithms for finding small attractors in Boolean networks. *EURASIP J. Bioinformatics Syst. Biol.* **2007**, 4–4 (2007)
62. Zhao, Y., Ciardo, G.: Symbolic computation of strongly connected components and fair cycles using saturation. *Innovations Syst. Softw. Eng.* **7**(2), 141–150 (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.