New ideas for scalable attractor detection and control of asynchronous Boolean networks

Van-Giang Trinh[1] and Samuel Pastva[2]

[1]LIS, Aix-Marseille University, Marseille, France
[2]Institute of Science and Technology, Austria

October 25, 2022

Attractor detection and control of asynchronous Boolean networks

Two most important issues in Boolean networks research with many applications in various fields [Akutsu, 2018, Schwab et al., 2020].

Usually, the attractor detection issue is the preceding step of the control issue.

Unfortunately, both of them are very challenging, especially for large-scale networks.

Many methods have been proposed.

Attractor detection methods

BDD-based methods: genYsis [Garg et al., 2008], geneFatt [Zheng et al., 2013], CABEAN [Mizera et al., 2019, Su and Pang, 2021a].

Reduction-based methods: AEON [Benes et al., 2021].

Trap space-based methods: PyBoolNet [1] [Klarner et al., 2017], pystablemotifs [Rozum et al., 2021b, Rozum et al., 2021a].

Feedback vertex set-based methods: FVS-ABN [Trinh et al., 2020], iFVS-ABN [Giang and Hiraishi, 2021], mtsNFVS [Trinh et al., 2022b].

All of them have advantages and disadvantages. To our best knowledge, none of them can robustly handle complex networks with thousands of nodes.

[1] It is not an exact method.

Control methods

BDD-based and decomposition-based methods:
CABEAN [Su and Pang, 2021b, Su and Pang, 2021a].

Reduction-based methods: AEON [Brim et al., 2021].

Trap space-based methods:
PyBoolNet [Fontanals et al., 2020, Cifuentes-Fontanals et al., 2021],
pystablemotifs [Zañudo and Albert, 2015, Rozum et al., 2021a].

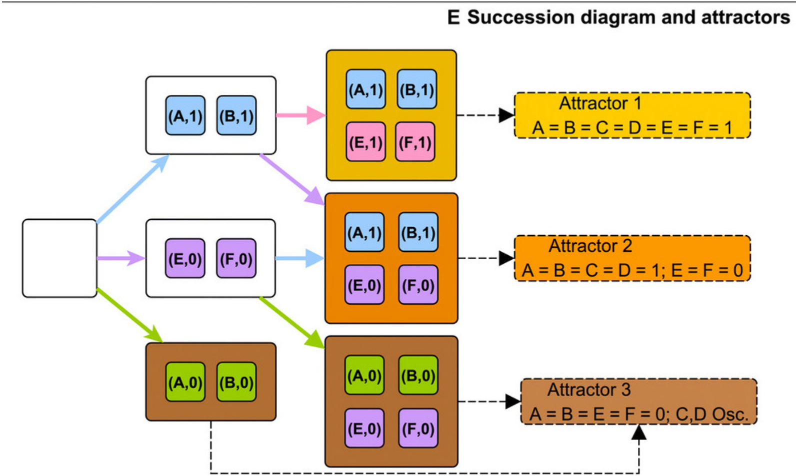
Objective

pystablemotifs and mtsNFVS have much more potential to reach the genome scale.

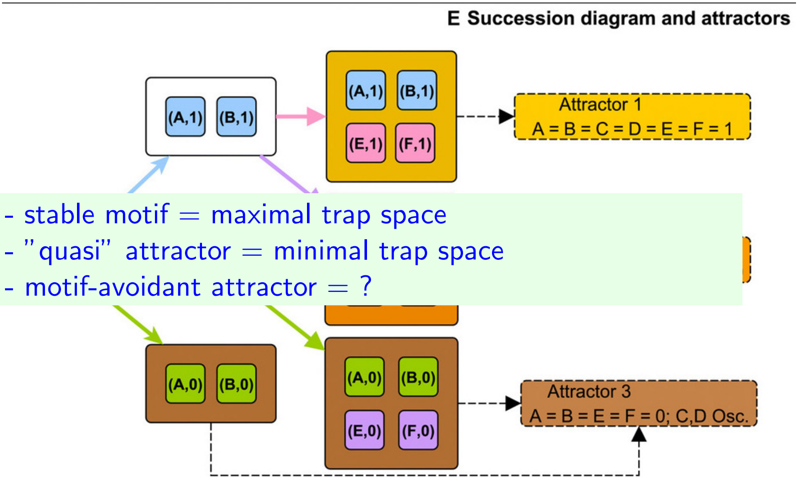
Both of them have their own advantages and disadvantages, but fortunately they can be complementary to each other.

Develop a more efficient method that can benefit both the attractor detection and control issues, and in particular can handle networks at the genome scale.

pystablemotifs

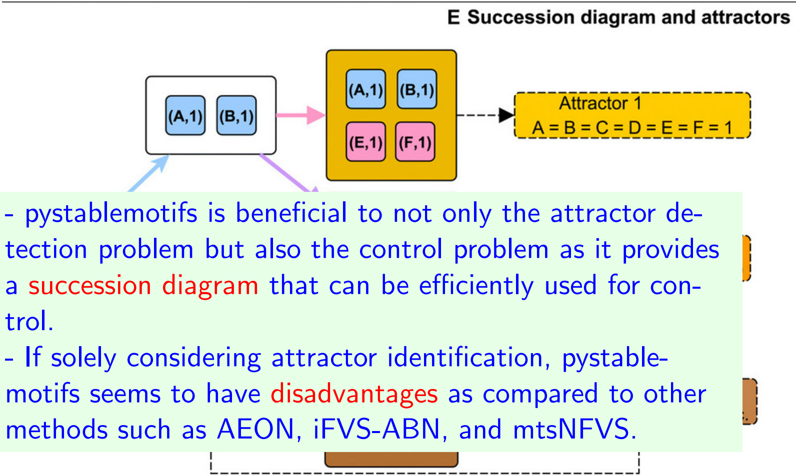


pystablemotifs



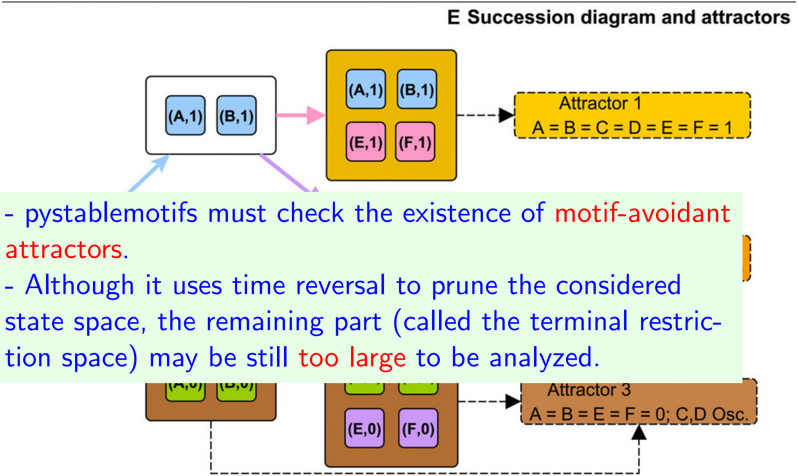
- stable motif = maximal trap space
- "quasi" attractor = minimal trap space
- motif-avoidant attractor = ?

pystablemotifs



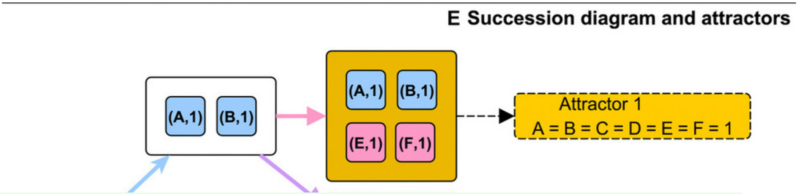
- pystablemotifs is beneficial to not only the attractor detection problem but also the control problem as it provides a succession diagram that can be efficiently used for control.
- If solely considering attractor identification, pystablemotifs seems to have disadvantages as compared to other methods such as AEON, iFVS-ABN, and mtsNFVS.

pystablemotifs



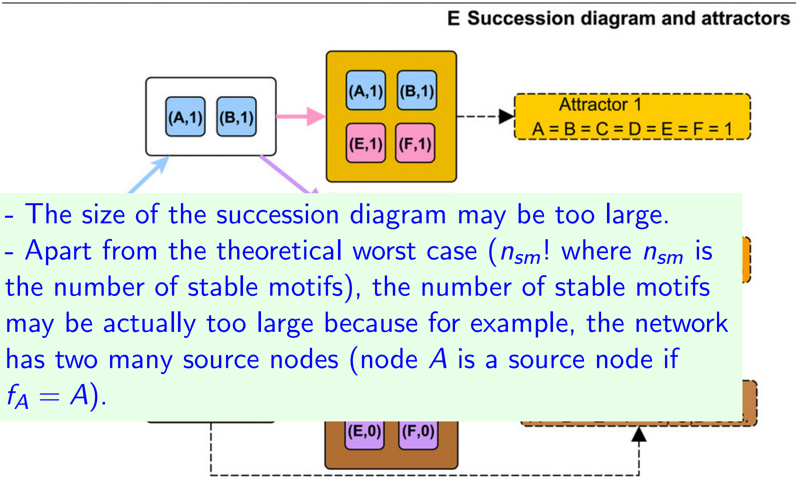
- pystablemotifs must check the existence of motif-avoidant attractors.
- Although it uses time reversal to prune the considered state space, the remaining part (called the terminal restriction space) may be still too large to be analyzed.

pystablemotifs



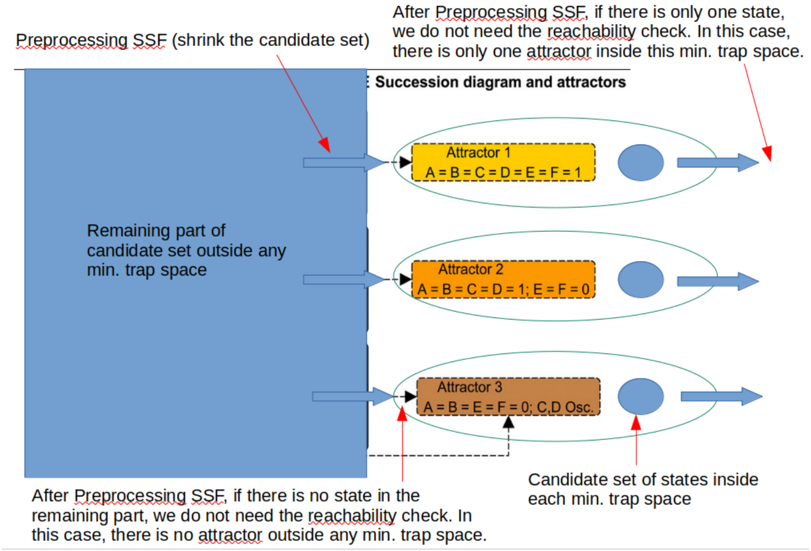
- pystablemotifs relies on PyBoolNet to compute stable motifs (i.e., maximal trap spaces). However, PyBoolNet does not work well with networks with Boolean functions having many input nodes.
- The reason for this is PyBoolNet mainly relies on prime-implicants computation, whereas the number of prime-implicants may be too huge if there are many input nodes.

pystablemotifs

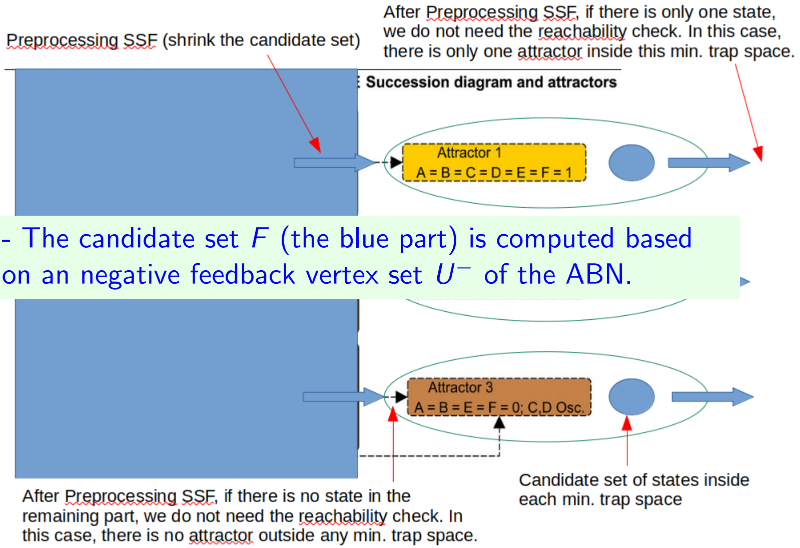


- The size of the succession diagram may be too large.
- Apart from the theoretical worst case (n_{sm}! where n_{sm} is the number of stable motifs), the number of stable motifs may be actually too large because for example, the network has two many source nodes (node A is a source node if $f_A = A$).

mtsNFVS


Preprocessing SSF (shrink the candidate set)
After Preprocessing SSF, if there is only one state, we do not need the reachability check. In this case, there is only one attractor inside this min. trap space.
Succession diagram and attractors
Remaining part of candidate set outside any min. trap space
Attractor 1
A = B = C = D = E = F = 1
Attractor 2
A = B = C = D = 1; E = F = 0
Attractor 3
A = B = E = F = 0; C,D Osc.
After Preprocessing SSF, if there is no state in the remaining part, we do not need the reachability check. In this case, there is no attractor outside any min. trap space.
Candidate set of states inside each min. trap space

mtsNFVS

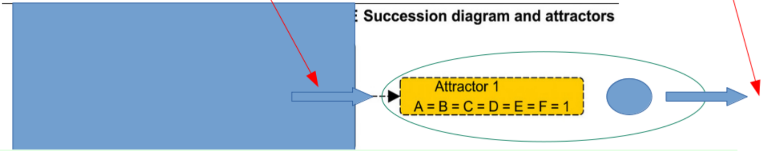


- The candidate set F (the blue part) is computed based on an negative feedback vertex set U^{-} of the ABN.

mtsNFVS

Preprocessing SSF (shrink the candidate set)

After Preprocessing SSF, if there is only one state, we do not need the reachability check. In this case, there is only one attractor inside this min. trap space.

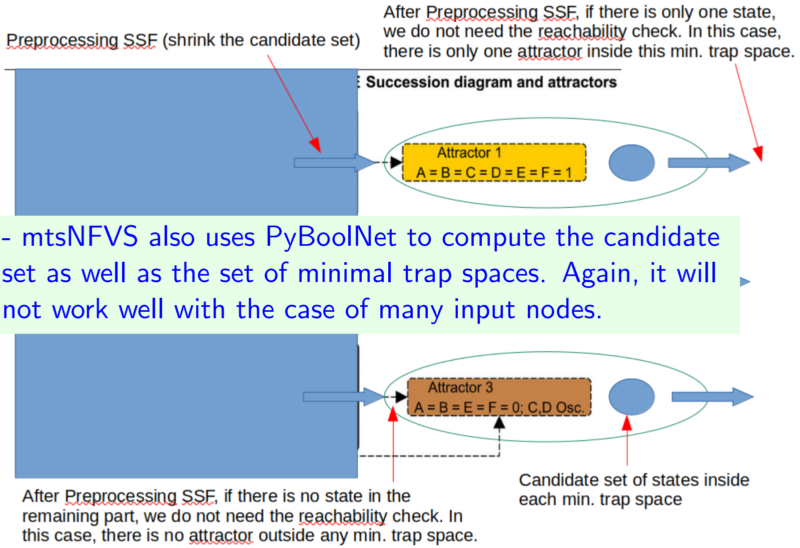


- If the size of the negative feedback vertex set is too large, there may be too many states in the candidate set F (i.e., the set of fixed points of the reduced STG), leading to extremely longer time for the computation of F, Preprocessing SSF, and maybe the reachability analysis.
- Note that real-world models usually have small minimum negative feedback vertex sets.

After Preprocessing SSF, if there is no state in the remaining part, we do not need the reachability check. In this case, there is no attractor outside any min. trap space.

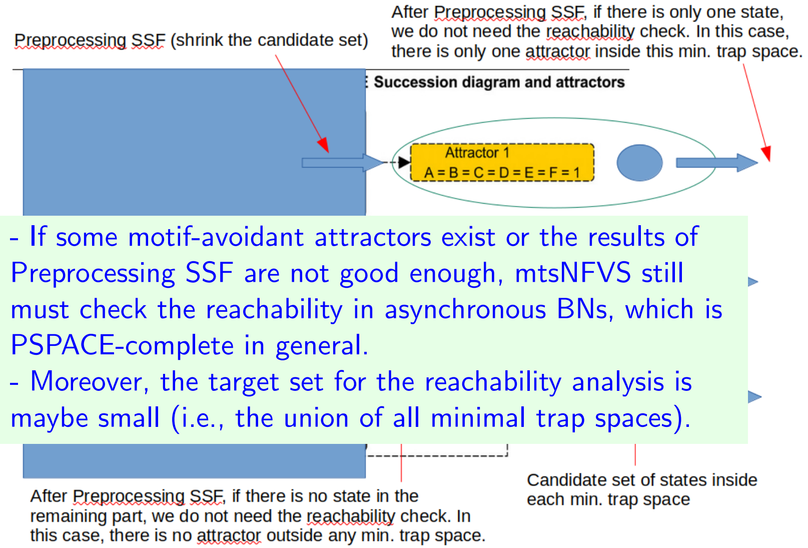
Candidate set of states inside each min. trap space

mtsNFVS



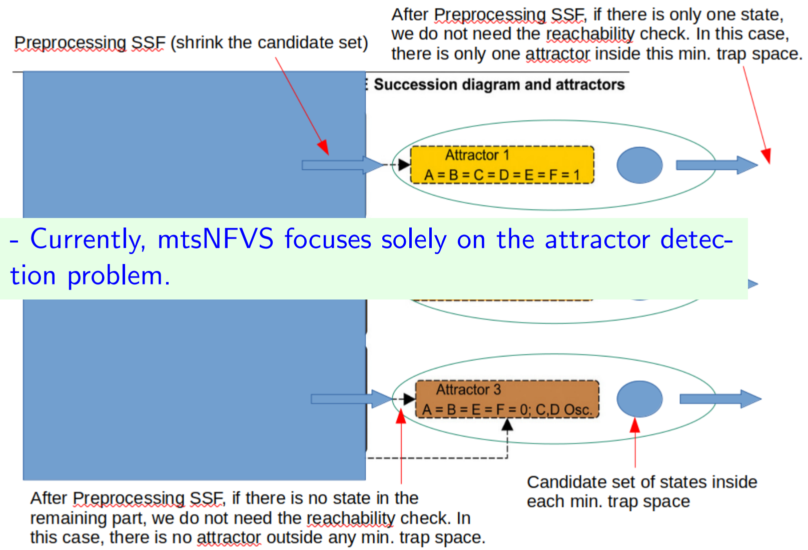
- mtsNFVS also uses PyBoolNet to compute the candidate set as well as the set of minimal trap spaces. Again, it will not work well with the case of many input nodes.

mtsNFVS



- If some motif-avoidant attractors exist or the results of Preprocessing SSF are not good enough, mtsNFVS still must check the reachability in asynchronous BNs, which is PSPACE-complete in general.
- Moreover, the target set for the reachability analysis is maybe small (i.e., the union of all minimal trap spaces).

mtsNFVS



- Currently, mtsNFVS focuses solely on the attractor detection problem.

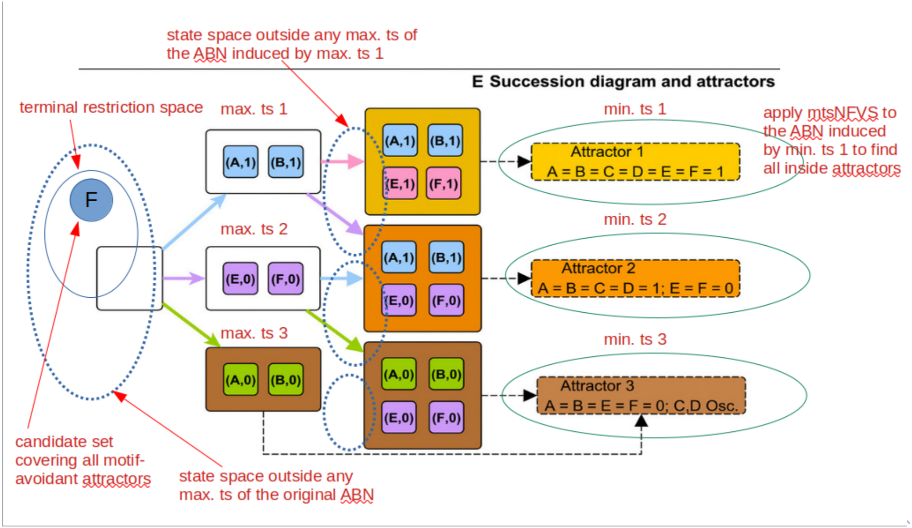
New idea

In principle, our new idea is to combine pystablemotifs and mtsNFVS to obtain a more efficient method that can benefit both the attractor detection and control problems.

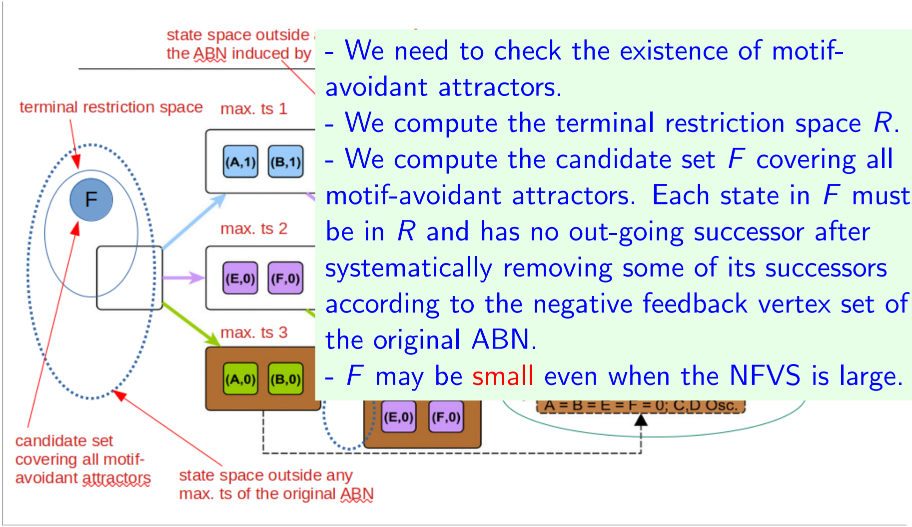
Hereafter, we will present the general approach along with many new findings that can speedup the whole process.

General approach

It follows the process of pystablemotifs (i.e., building the succession diagram).

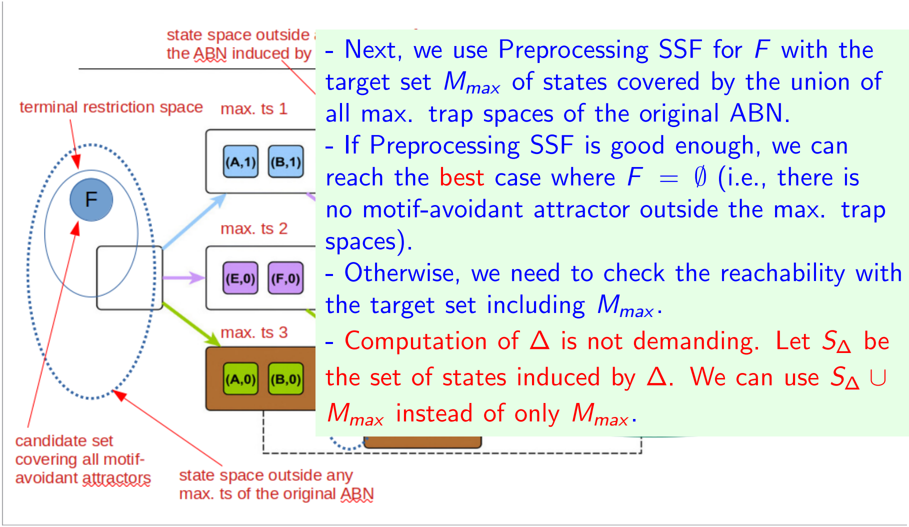


General approach



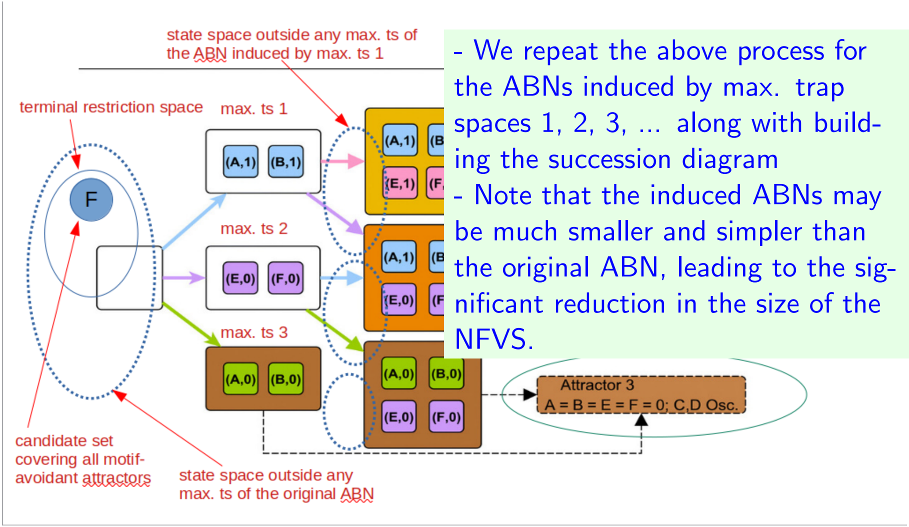
- We need to check the existence of motif-avoidant attractors.
- We compute the terminal restriction space R.
- We compute the candidate set F covering all motif-avoidant attractors. Each state in F must be in R and has no out-going successor after systematically removing some of its successors according to the negative feedback vertex set of the original ABN.
- F may be small even when the NFVS is large.

General approach



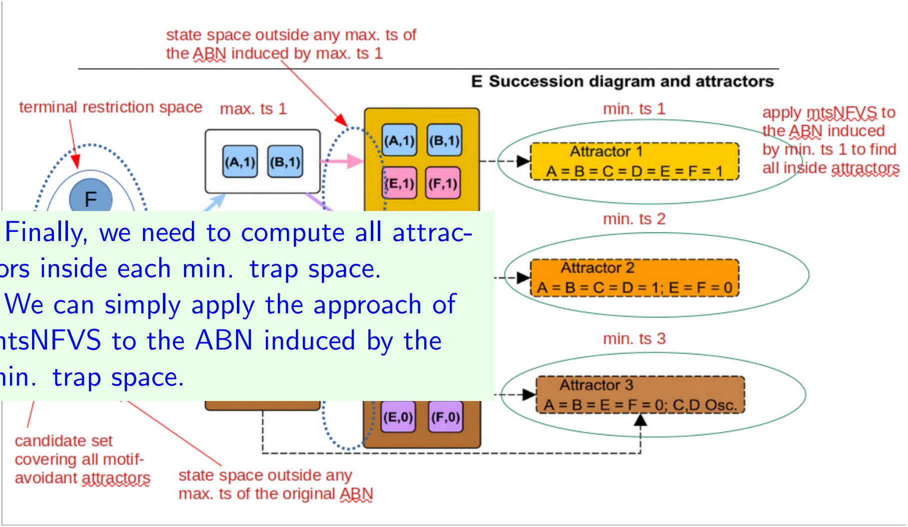
- Next, we use Preprocessing SSF for F with the target set M_{max} of states covered by the union of all max. trap spaces of the original ABN.
- If Preprocessing SSF is good enough, we can reach the best case where $F = \emptyset$ (i.e., there is no motif-avoidant attractor outside the max. trap spaces).
- Otherwise, we need to check the reachability with the target set including M_{max}.
- Computation of Δ is not demanding. Let S_Δ be the set of states induced by Δ. We can use $S_\Delta \cup M_{max}$ instead of only M_{max}.

General approach



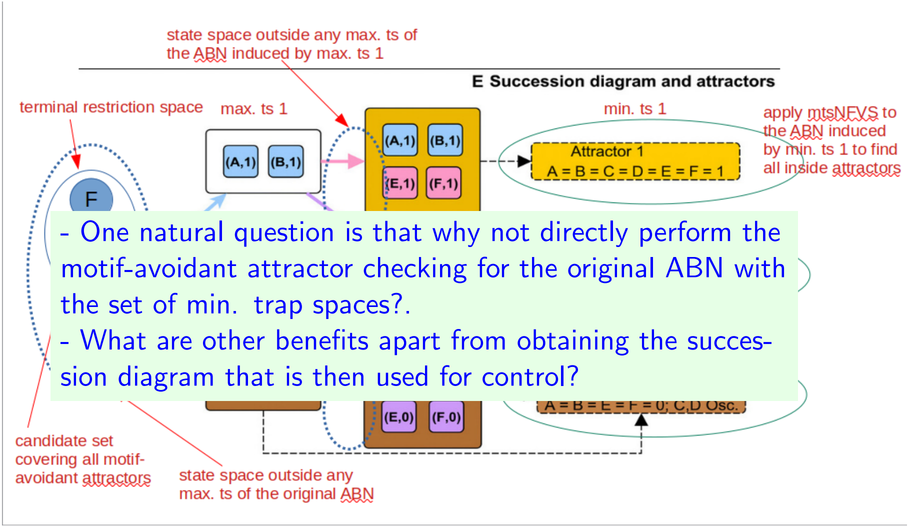
- We repeat the above process for the ABNs induced by max. trap spaces 1, 2, 3, ... along with building the succession diagram
- Note that the induced ABNs may be much smaller and simpler than the original ABN, leading to the significant reduction in the size of the NFVS.

General approach



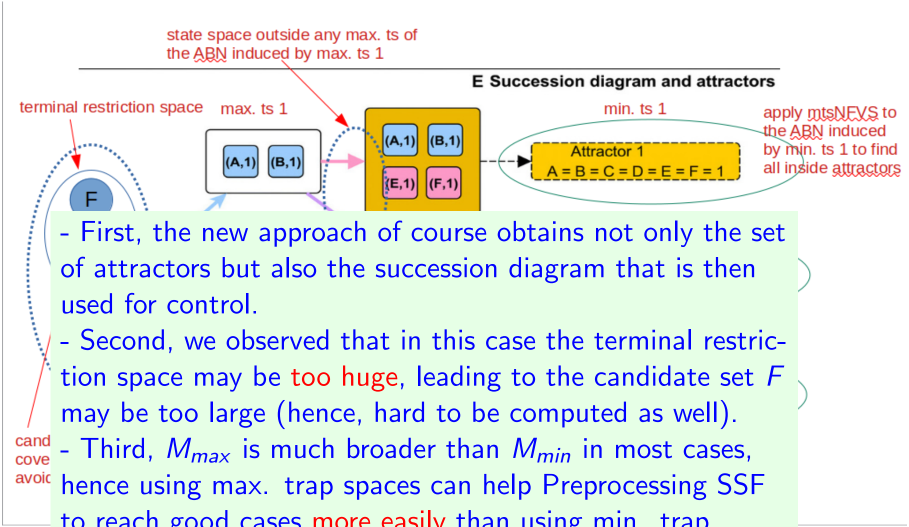
- Finally, we need to compute all attractors inside each min. trap space.
- We can simply apply the approach of mtsNFVS to the ABN induced by the min. trap space.

General approach



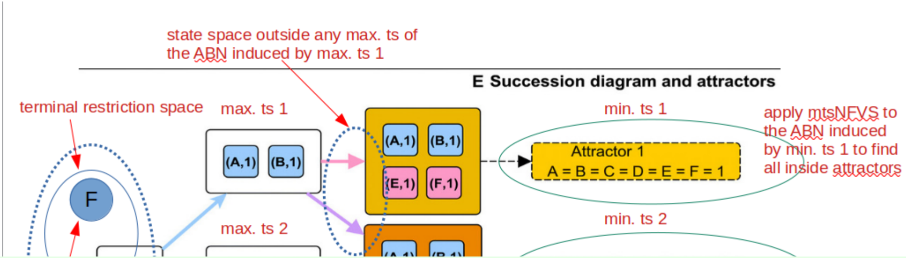
- One natural question is that why not directly perform the motif-avoidant attractor checking for the original ABN with the set of min. trap spaces?.
- What are other benefits apart from obtaining the succession diagram that is then used for control?

General approach



- First, the new approach of course obtains not only the set of attractors but also the succession diagram that is then used for control.
- Second, we observed that in this case the terminal restriction space may be too huge, leading to the candidate set F may be too large (hence, hard to be computed as well).
- Third, M_{max} is much broader than M_{min} in most cases, hence using max. trap spaces can help Preprocessing SSF to reach good cases more easily than using min. trap spaces.

General approach

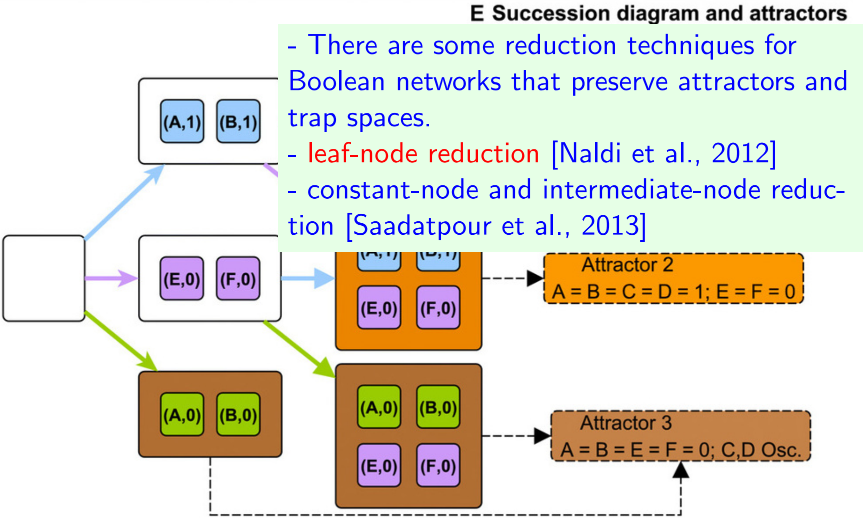


Jordan: We need to consider the percolation of a stable motif (LDOI) along with building the succession diagram. The percolation is based on PyBoolNet.

Sam: When we have already obtained the stable motif (i.e., max. trap space), we can perform the percolation easily by algebraically propagating the fixed values.

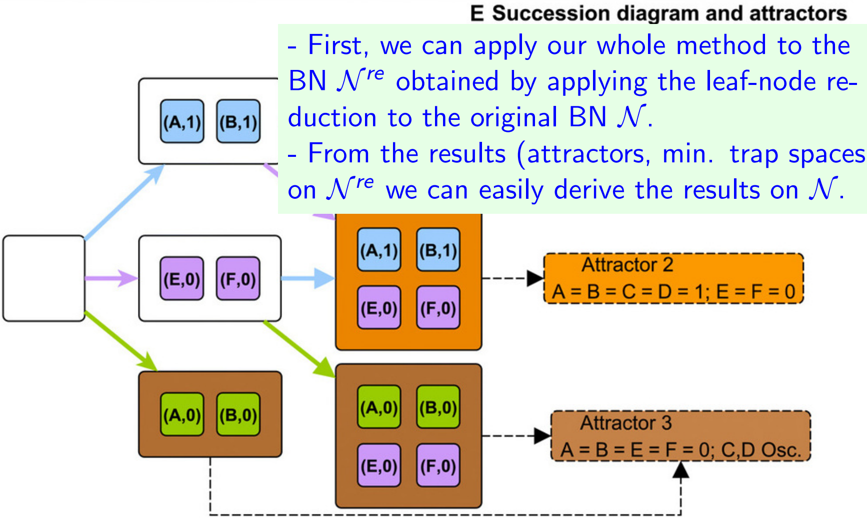
Giang: Sure. It is possible. Note that we can store Boolean functions as Boolean expressions or BDDs (not as prime-implicants).

Network reduction



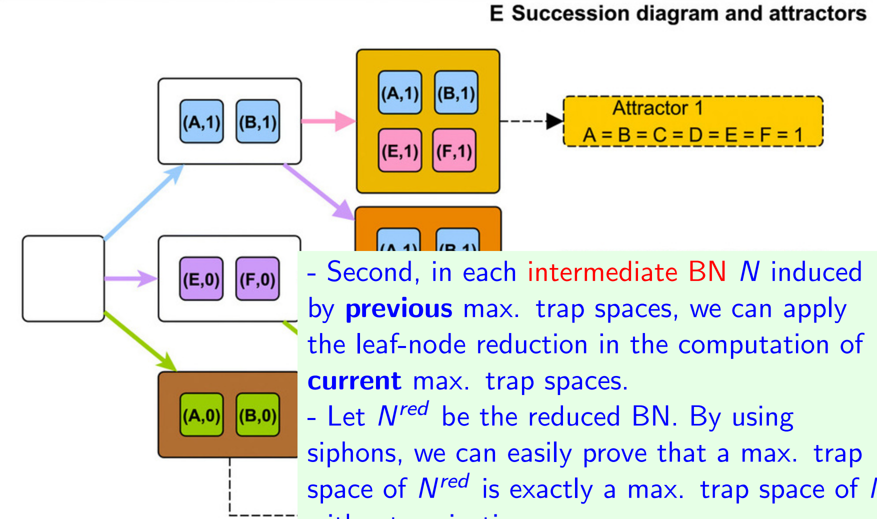
- There are some reduction techniques for Boolean networks that preserve attractors and trap spaces.
- leaf-node reduction [Naldi et al., 2012]
- constant-node and intermediate-node reduction [Saadatpour et al., 2013]

Network reduction



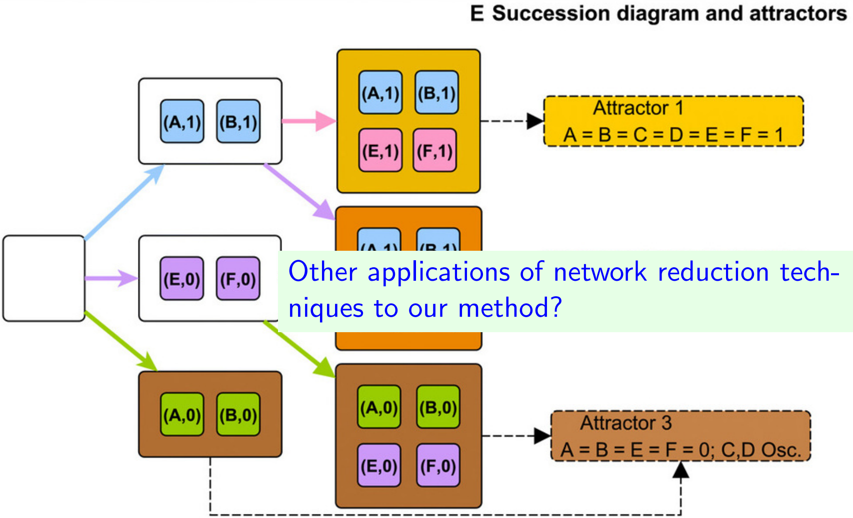
- First, we can apply our whole method to the BN $\mathcal{N}^{re}$ obtained by applying the leaf-node reduction to the original BN $\mathcal{N}$.
- From the results (attractors, min. trap spaces) on $\mathcal{N}^{re}$ we can easily derive the results on $\mathcal{N}$.

Network reduction



- Second, in each intermediate BN N induced by **previous** max. trap spaces, we can apply the leaf-node reduction in the computation of **current** max. trap spaces.
- Let N^{red} be the reduced BN. By using siphons, we can easily prove that a max. trap space of N^{red} is exactly a max. trap space of N without projection.

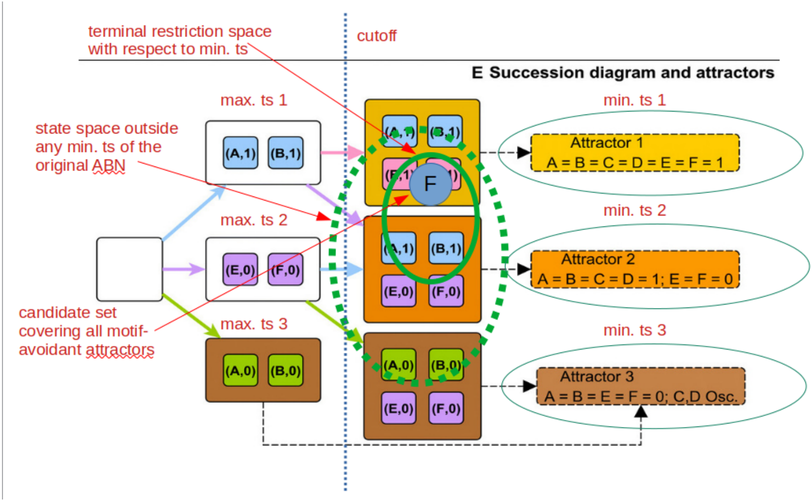
Network reduction



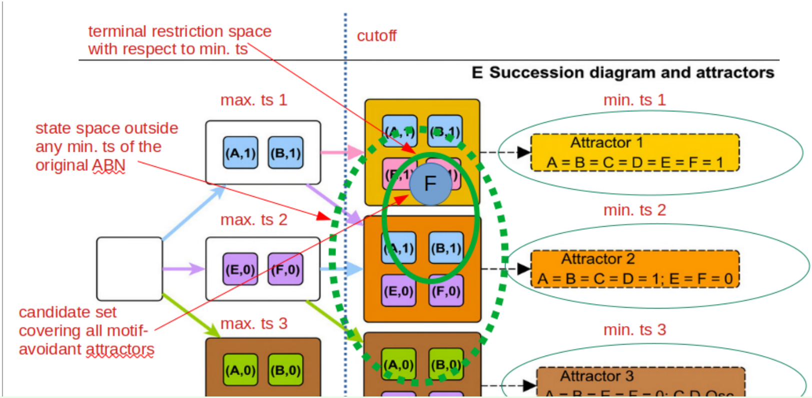
Other applications of network reduction techniques to our method?

Deal with the problem of large succession diagrams

The above approach is promising. However, how to deal with the problem of large succession diagrams?.

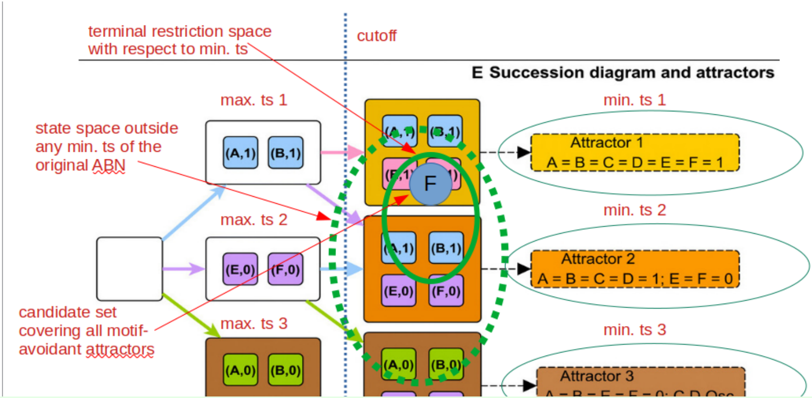


Deal with the problem of large succession diagrams



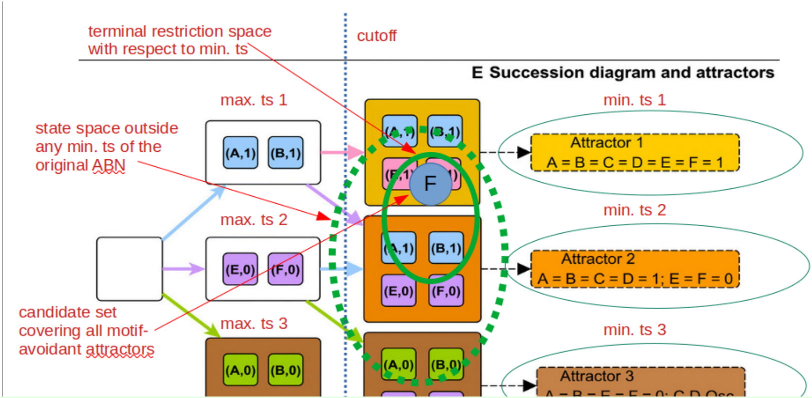
- When the size of the succession diagram becomes too large (maybe exceeding a threshold), we stop its construction.
- However, we still need to check the existence of motif-avoidant attractors.

Deal with the problem of large succession diagrams



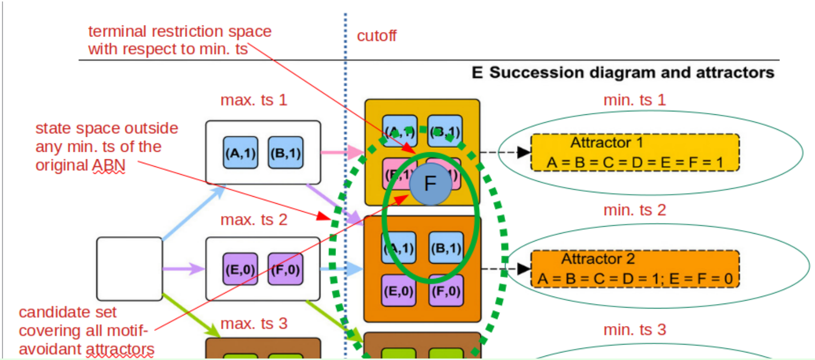
- First, we compute all min. trap spaces of the original ABN.
- Second, we compute the terminal restriction space R with respect to these min. trap spaces.
- Next, we get the intersection between R and the state spaces induced by the already computed max. trap spaces and their LDOIs (e.g., max. ts 1, 2, 3).

Deal with the problem of large succession diagrams



- Then, we compute the candidate set F with respect to the NFVS of the original ABN.
- Each state in F must be in the intersected set obtained beforehand.
- F may be small even when the NFVS is large because the intersected set may be not too large.

Deal with the problem of large succession diagrams

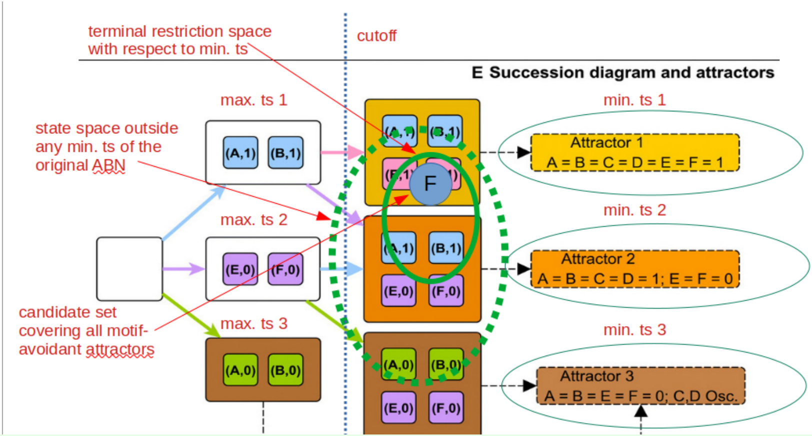


Next, we use Preprocessing SSF for F with the target set M_{min} of states covered by the union of all min. trap spaces of the original ABN.

- If Preprocessing SSF is good enough, we can reach the best case where $F = \emptyset$ (i.e., there is no motif-avoidant attractor).

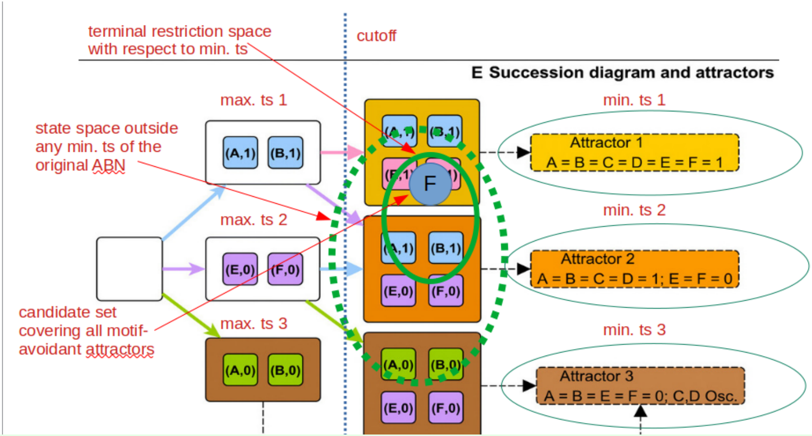
- Otherwise, we need to check the reachability with the target set including M_{min}.

Deal with the problem of large succession diagrams



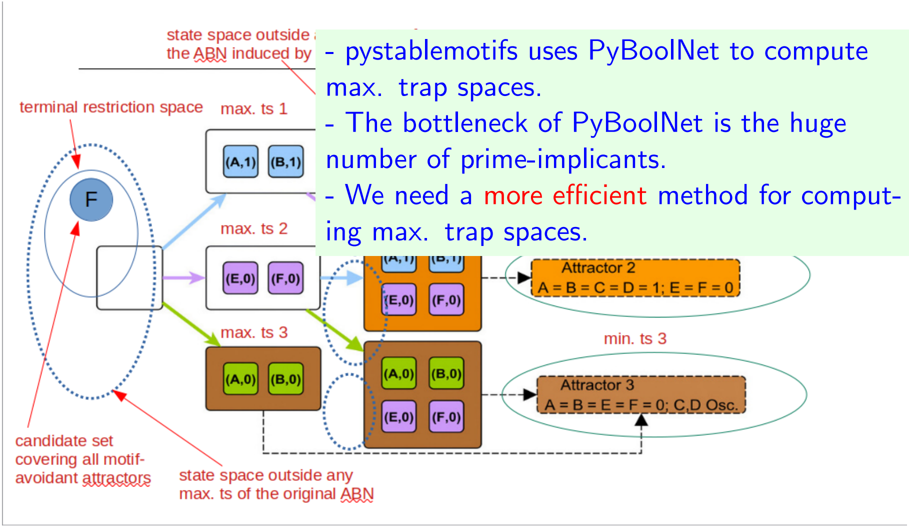
- Finally, we need to compute all attractors inside each min. trap space.
- We can simply apply the approach of mtsNFVS to the ABN induced by the min. trap space.

Deal with the problem of large succession diagrams



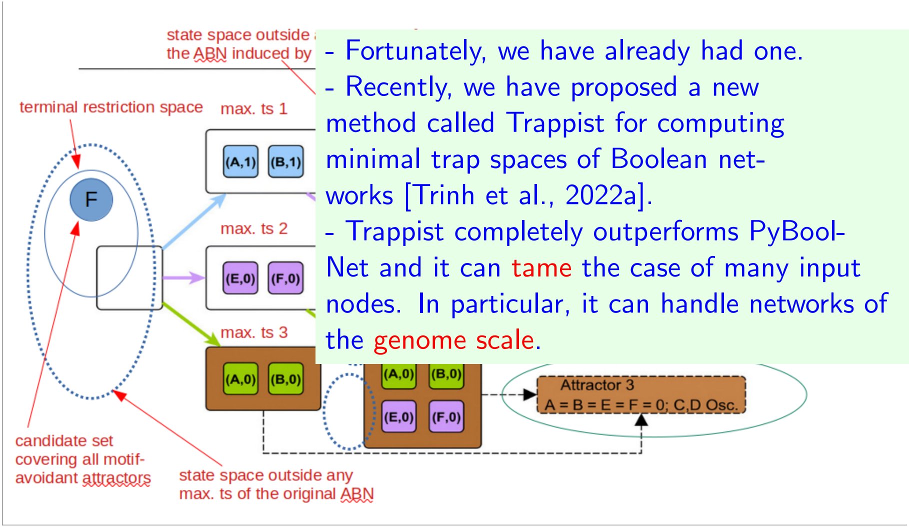
- After finishing the above process, we will obtain the whole attractor landscape and the partial succession diagram.

Computation of max. trap spaces



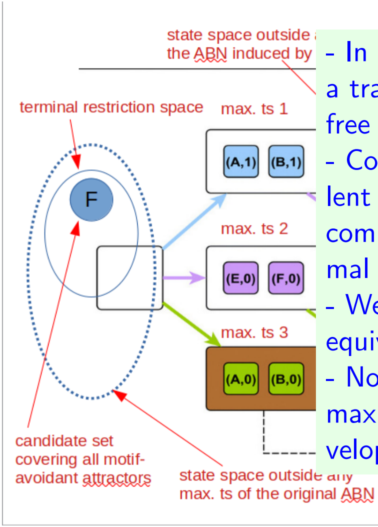
- pystablemotifs uses PyBoolNet to compute max. trap spaces.
- The bottleneck of PyBoolNet is the huge number of prime-implicants.
- We need a more efficient method for computing max. trap spaces.

Computation of max. trap spaces



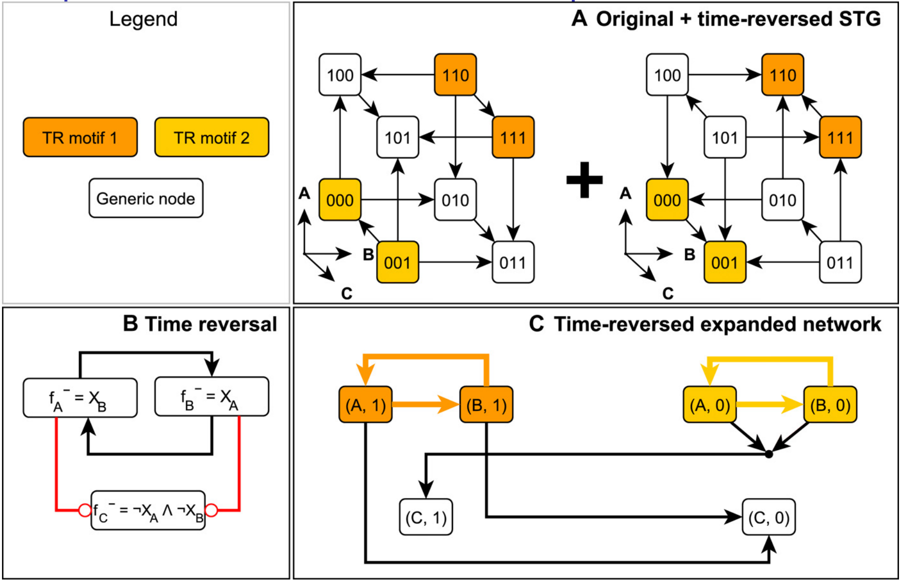
- Fortunately, we have already had one.
- Recently, we have proposed a new method called Trappist for computing minimal trap spaces of Boolean networks [Trinh et al., 2022a].
- Trappist completely outperforms PyBoolNet and it can tame the case of many input nodes. In particular, it can handle networks of the genome scale.

Computation of max. trap spaces

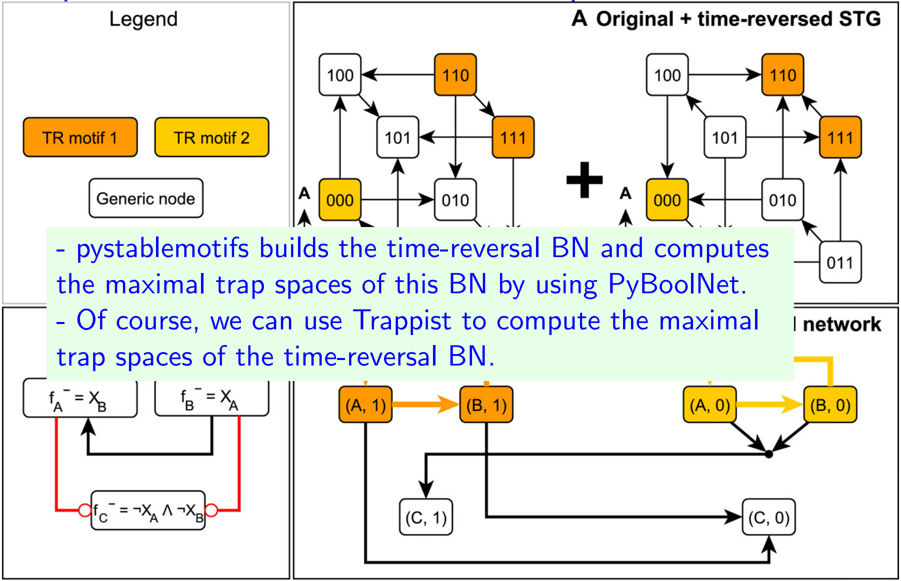


- In [Trinh et al., 2022a], we have proved that a trap space of a BN is equivalent to a conflict-free siphon of its Petri net encoding.
- Consequently, a min. trap space is equivalent to a maximal conflict-free siphon. Trappist computes min. trap spaces by computing maximal conflict-free siphons.
- We can easily prove that a max. trap space is equivalent to a minimal conflict-free siphon.
- Now, Trappist does not support computing max. trap spaces. However, we can easily develop this feature for Trappist.

Computation of terminal restriction spaces

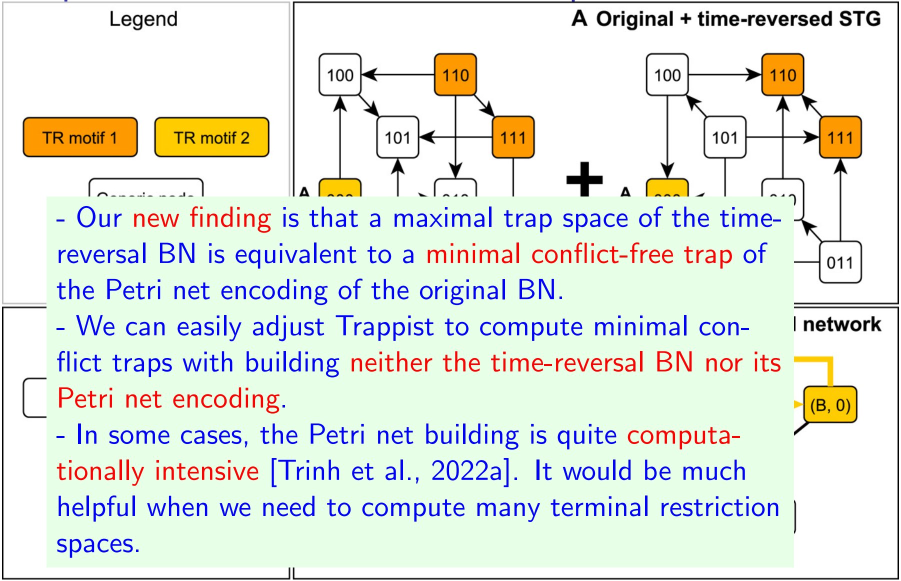


Computation of terminal restriction spaces



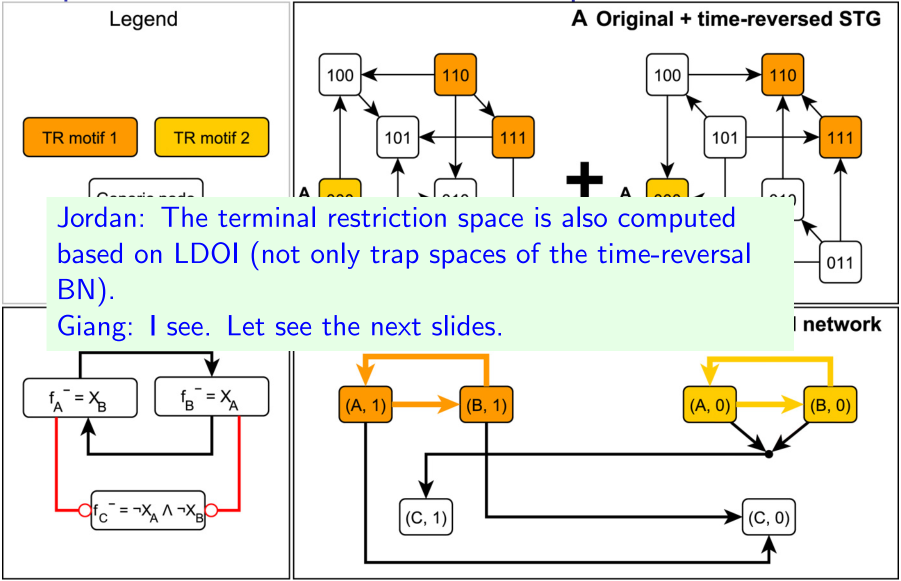
- pystablemotifs builds the time-reversal BN and computes the maximal trap spaces of this BN by using PyBoolNet.
- Of course, we can use Trappist to compute the maximal trap spaces of the time-reversal BN.

Computation of terminal restriction spaces



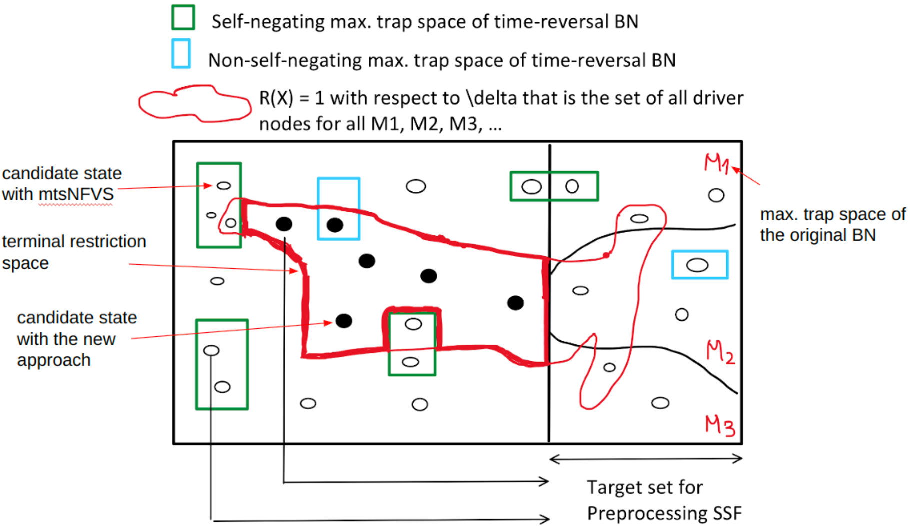
- Our new finding is that a maximal trap space of the time-reversal BN is equivalent to a minimal conflict-free trap of the Petri net encoding of the original BN.
- We can easily adjust Trappist to compute minimal conflict traps with building neither the time-reversal BN nor its Petri net encoding.
- In some cases, the Petri net building is quite computationally intensive [Trinh et al., 2022a]. It would be much helpful when we need to compute many terminal restriction spaces.

Computation of terminal restriction spaces



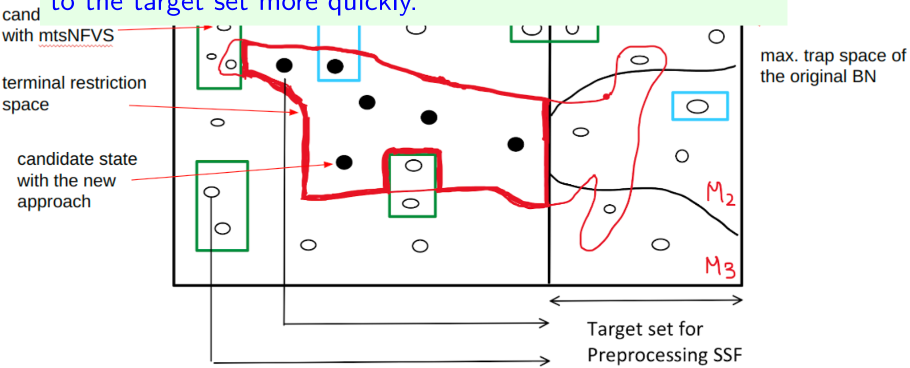
Jordan: The terminal restriction space is also computed based on LDOI (not only trap spaces of the time-reversal BN).
Giang: I see. Let see the next slides.

Computation of terminal restriction spaces (cont.)

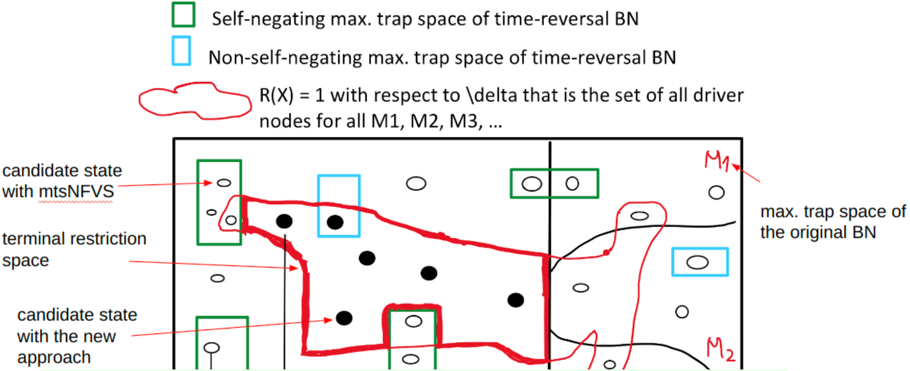


Computation of terminal restriction spaces (cont.)

- I emphasize an advantage of the new approach.
- The new candidate set seems to have shorter distances to the target set.
- This is better for Preprocessing SSF as it may converge to the target set more quickly.

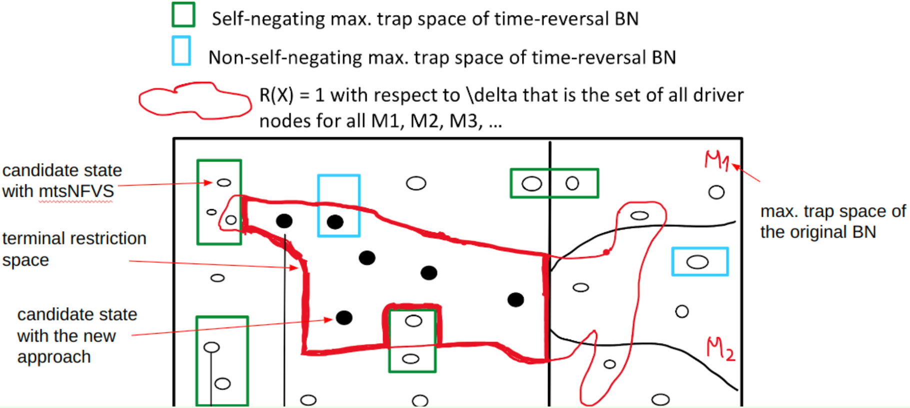


Computation of terminal restriction spaces (cont.)



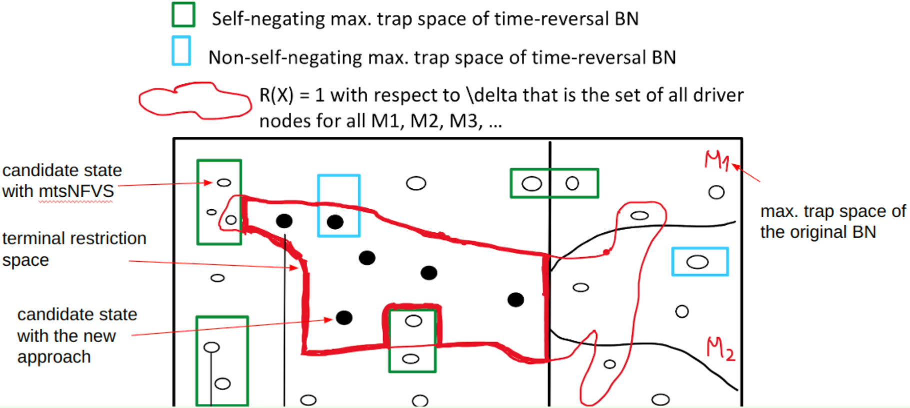
- We can efficiently compute max. trap spaces of the time-reversal BN by using Trappist.
- The self-negation of a max. trap space of the time-reversal BN can be checked by computing its LDOI in the original BN.

Computation of terminal restriction spaces (cont.)



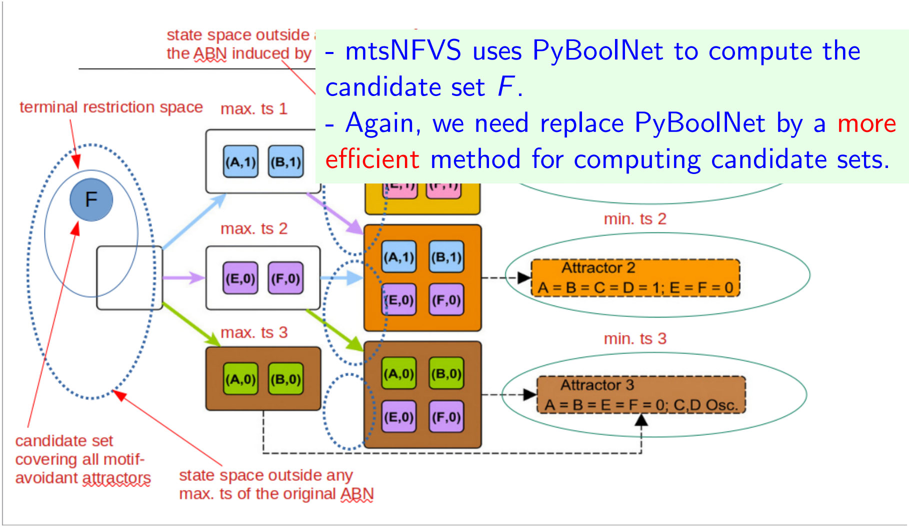
- Δ is the set of all virtual nodes that individually drive any stable motif.
- A question is how to compute Δ efficiently? Note that we need to compute Δ many times in the whole process.
- If the computation is expensive, we can ignore $R(X)$. In this case, I hope that self-negating max. trap spaces are good enough.

Computation of terminal restriction spaces (cont.)



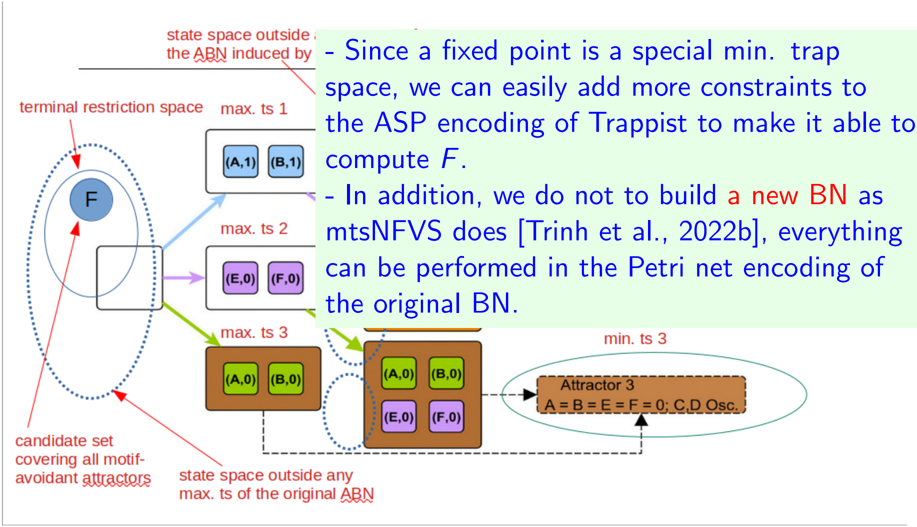
- The set of retained values B largely affects the size of the candidate set [Trinh et al., 2020].
- One interesting question is that how we can propose heuristics for setting B based on the information about the terminal restriction space to get a smaller candidate set?

Computation of candidate sets



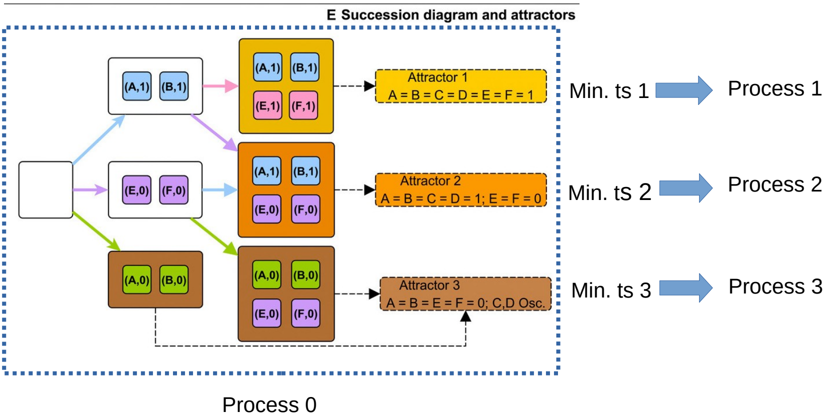
- mtsNFVS uses PyBoolNet to compute the candidate set F.
- Again, we need replace PyBoolNet by a more efficient method for computing candidate sets.

Computation of candidate sets

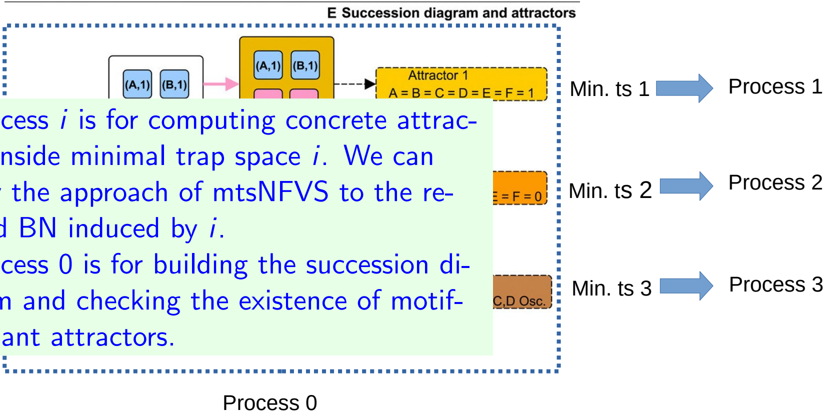


- Since a fixed point is a special min. trap space, we can easily add more constraints to the ASP encoding of Trappist to make it able to compute F.
- In addition, we do not to build a new BN as mtsNFVS does [Trinh et al., 2022b], everything can be performed in the Petri net encoding of the original BN.

Parallelization

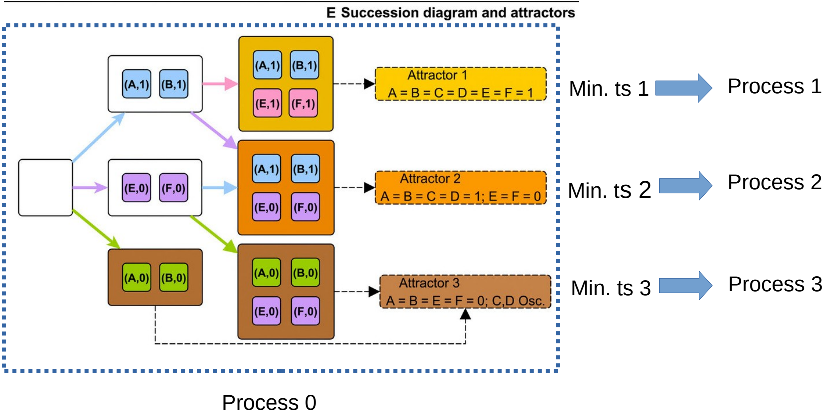


Parallelization



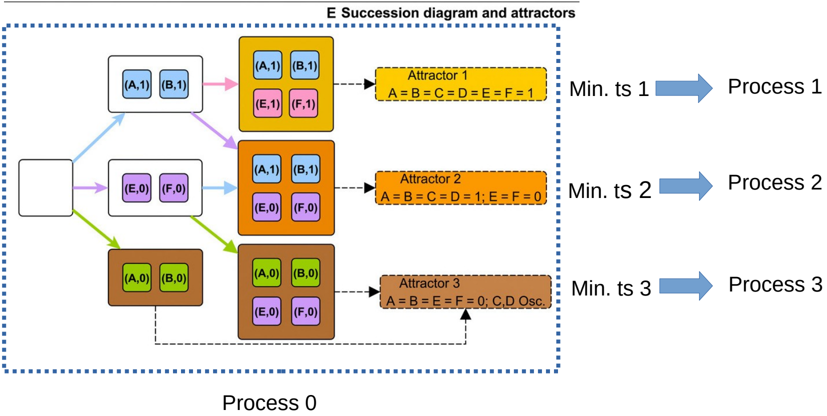
- Process *i* is for computing concrete attractors inside minimal trap space *i*. We can apply the approach of mtsNFVS to the reduced BN induced by *i*.
- Process 0 is for building the succession diagram and checking the existence of motif-avoidant attractors.

Parallelization



- Processes 0, 1, 2, ... can be completely paralleled.

Parallelization



- Process 0 can even be divided into many paralleled sub-processes.

Target control

After finishing the attractor detection phase, we will obtain the whole attractor landscape as well as the partial/full succession diagram. If the succession diagram is full, its size should be moderate.

The next step is how to compute minimum control policies that drive the network dynamics into a target attractor/min. trap space from any initial state?

There are some existing methods for the target control of asynchronous Boolean networks: CABEAN [Su and Pang, 2021a],
PyBoolNet [Fontanals et al., 2020, Cifuentes-Fontanals et al., 2021],
pystablemotifs [Zañudo and Albert, 2015, Rozum et al., 2021a],
AEON [Brim et al., 2021].

Target control: our observations I

CABEAN always ensures that the resulting control policies are minimum. It relies on the calculation of strong basin of attraction, which may be very computationally expensive.

PyBoolNet exploits the information on trap spaces to find control policies. However, it does not ensure that the resulting control policies are minimum. It uses model checking to improve the quality of the control policies, which may be very computationally expensive as well. Moreover, its scalability may be limited by the necessity to scan through the available perturbations.

pystablemotifs uses the succession diagram to find control policies. However, it does not ensure that the resulting control policies are minimum.

Target control: our observations II

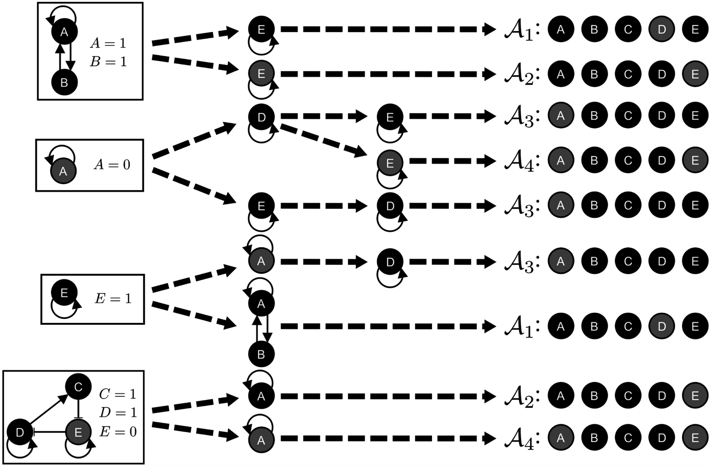
AEON uses methods based on the computation of the strong basin, but with symbolic representation to exhaustively analyse all perturbations simultaneously. A postprocessing step is then needed to extract the minimal perturbations from the final result. Alternatively, if the network has parameters, such postprocessing can also be used to extract perturbations with favourable robustness within the parameter space. The method is mainly limited by the complexity of the symbolic state space search for all available perturbations.

In the reported experiments [Su and Pang, 2021b, Su and Pang, 2021a], pystablemotifs is more time-efficient in most cases, though it did not return the minimum results in some cases.

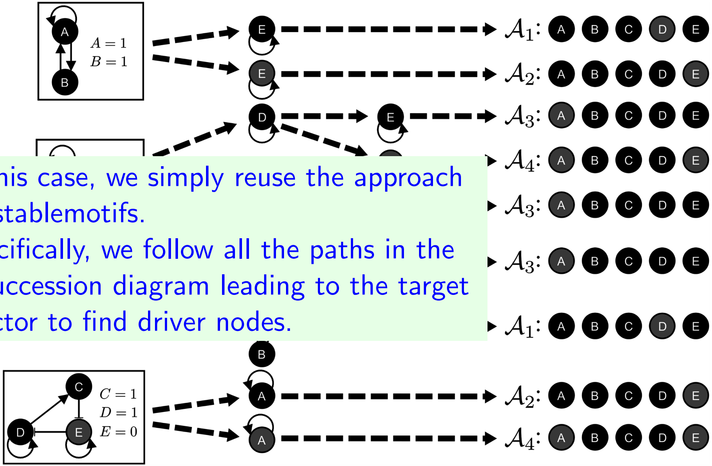
Target control: our observations III

In our opinion, pystablemotifs has much more potential than the others. We should dive into it. Of course, we need to improve both its time-efficiency and accuracy.

Target control with the full succession diagram

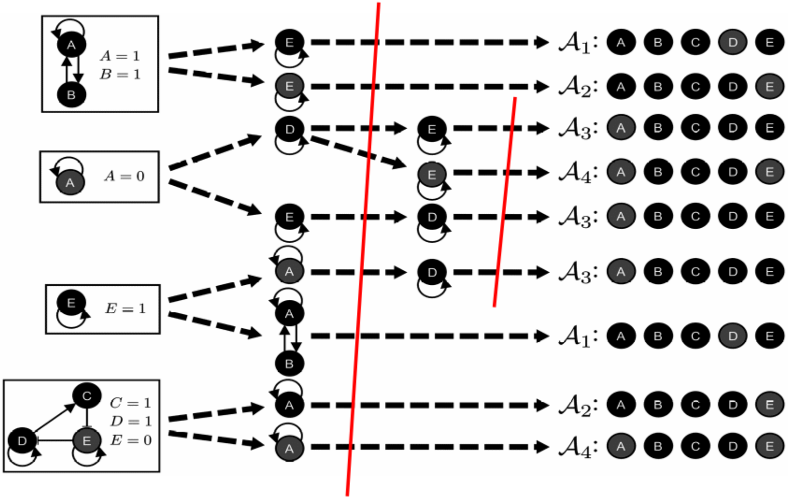


Target control with the full succession diagram

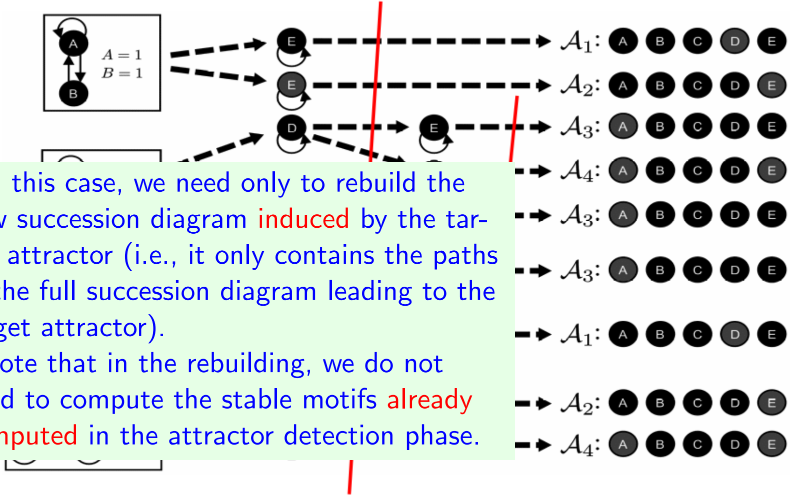


- In this case, we simply reuse the approach of pystablemotifs.
- Specifically, we follow all the paths in the full succession diagram leading to the target attractor to find driver nodes.

Target control with the partial succession diagram

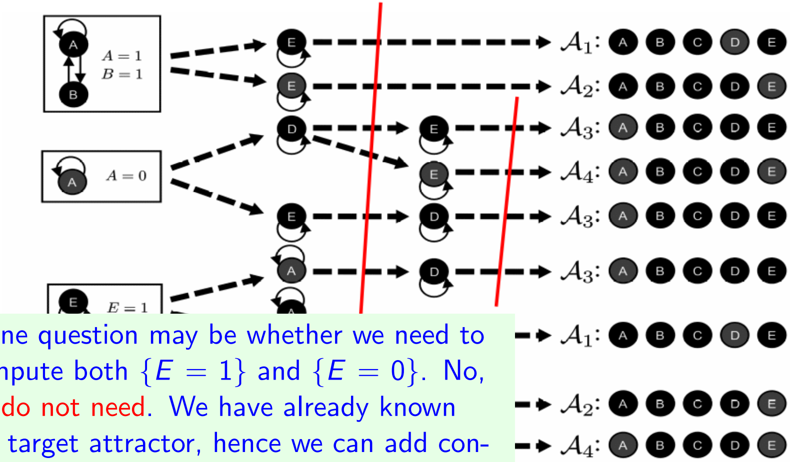


Target control with the partial succession diagram



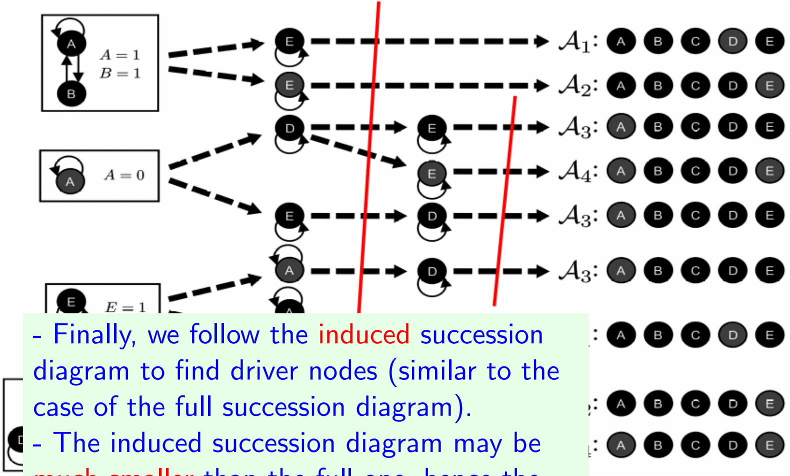
- In this case, we need only to rebuild the new succession diagram induced by the target attractor (i.e., it only contains the paths in the full succession diagram leading to the target attractor).
- Note that in the rebuilding, we do not need to compute the stable motifs already computed in the attractor detection phase.

Target control with the partial succession diagram



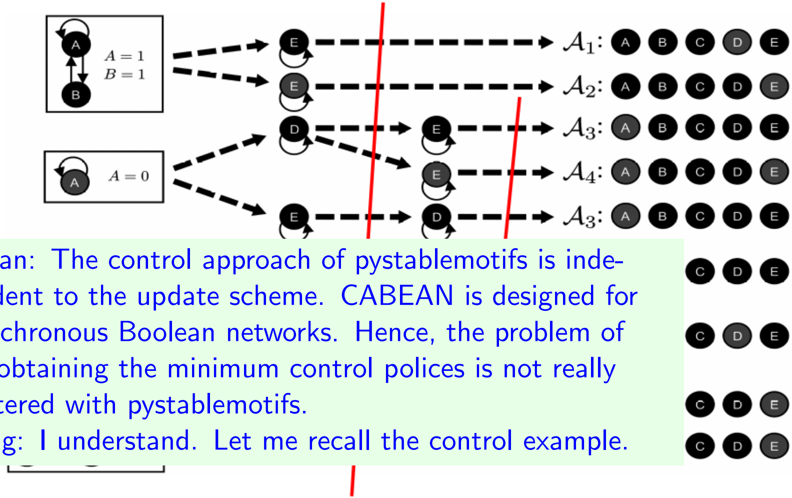
- One question may be whether we need to compute both $\{E = 1\}$ and $\{E = 0\}$. No, we do not need. We have already known the target attractor, hence we can add constraints to the ASP encoding to guide Trappist for searching only stable motifs that are compatible with the target attractor $\mathcal{A}_3$.

Target control with the partial succession diagram



- Finally, we follow the induced succession diagram to find driver nodes (similar to the case of the full succession diagram).
- The induced succession diagram may be much smaller than the full one, hence the computational burden may be reduced significantly.

Target control with the partial succession diagram

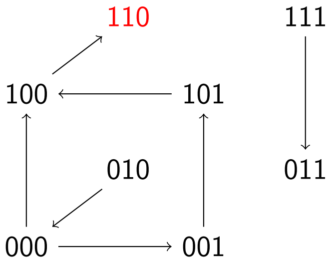


Jordan: The control approach of pystablemotifs is independent to the update scheme. CABEAN is designed for asynchronous Boolean networks. Hence, the problem of not obtaining the minimum control polices is not really mattered with pystablemotifs.
Giang: I understand. Let me recall the control example.

Improve the accuracy

Consider the example asynchronous Boolean network (Figure 2 in [Cifuentes-Fontanals et al., 2021]).

$$\begin{cases} f_1 = \neg x_2 \vee (x_1 \wedge \neg x_3) \\ f_2 = (x_1 \wedge \neg x_3) \vee (x_2 \wedge x_3) \\ f_3 = (\neg x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3) \end{cases}$$



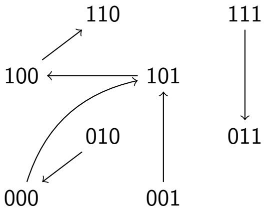
The result of pystablemotifs is $\{x_1 = 1, x_3 = 0\}$. The result of CABEAN (also of PyBoolNet with model checking) is $\{x_3 = 0\}$.

We have obtained a preliminary idea for improving the accuracy of pystablemotifs. However, we need to make it more specific as well as investigate its efficiency. In most cases, smaller control policies, longer computational time. We will present it in the next slides.

Improve the accuracy (cont.)

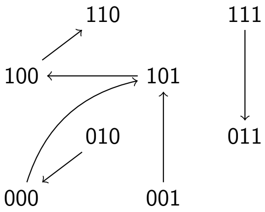
Consider the synchronous counterpart of the previous asynchronous Boolean network.

$$\begin{cases} f_1 = \neg x_2 \vee (x_1 \wedge \neg x_3) \\ f_2 = (x_1 \wedge \neg x_3) \vee (x_2 \wedge x_3) \\ f_3 = (\neg x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3) \end{cases}$$



Improve the accuracy (cont.)

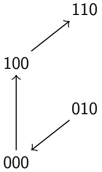
$$\begin{cases} f_1 = \neg x_2 \vee (x_1 \wedge \neg x_3) \\ f_2 = (x_1 \wedge \neg x_3) \vee (x_2 \wedge x_3) \\ f_3 = (\neg x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3) \end{cases}$$



Assume that the target attractor is $\{110\}$. The result of pystablemotifs is $\{x_1 = 1, x_3 = 0\}$. The minimum result still should be $\{x_3 = 0\}$.

Improve the accuracy (cont.)

$$\begin{cases} f_1 = \neg x_2 \vee x_1 \\ f_2 = x_1 \end{cases}$$

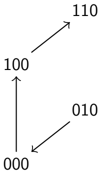


$\{x_1 = 1, x_3 = 0\}$ is a stable motif obtained by pystablemotifs. The function `minimal_drivers` (in the file `drivers.py`) finds the minimum subsets of stable motif's states that, when fixed in the logical model, are enough to force the state of every node in the motif into the stable motif M. This function considers all subsets of size 1, 2, ..., $|M| - 1$ and relies on LDOI (`implied,contradicted = logical_domain_of_influence(driver_dict,primes)`).

In the example, $\{x_3 = 0\}$ cannot be obtained by using this method.

Improve the accuracy (cont.)

$$\begin{cases} f_1 = \neg x_2 \vee x_1 \\ f_2 = x_1 \end{cases}$$



I guess if we use DOI instead of LDOI, we can get $\{x_3 = 0\}$ instead of $\{x_1 = 1, x_3 = 0\}$. However, calculating DOI can be difficult in general.

Research question: How to develop an efficient algorithm for driver node identification in a stable motif regardless of update schemes?

The efficient word means that the algorithm is fast and can return smaller (maybe not optimal) sets of drive nodes.

Ideas for driver node identification

From the previous example, we observed that:

- If fixing $x_1 = 1$, the resulting BN has two max. trap spaces $\{x_3 = 0\}$ and $\{x_2 = 1, x_3 = 1\}$. The latter is not compatible with the stable motif $\{x_1 = 1, x_3 = 0\}$.
- If fixing $x_3 = 0$, the resulting BN has only one max. trap space $\{x_1 = 1\}$. All max. trap spaces are compatible with the stable motif $\{x_1 = 1, x_3 = 0\}$.

We formulate a problem as follows.

Driver node identification

Let $\mathcal{N}$ be a BN and $M = \{x_{i_1} = a_{i_1}, \ldots, x_{i_k} = a_{i_k}\}$ be a max. trap space of $\mathcal{N}$. Find all minimum subsets S of M such that of max. trap spaces of the new BN induced by S are compatible with M.

This problem is related to the fixed point control problem considered in [Biane and Delaplace, 2019, Moon et al., 2022].

Ideas for driver node identification

We can solve this problem by using answer set programming. The main idea is as follows.

- Consider the nodes in M as controllable nodes (x_1, x_3) and the remaining nodes as uncontrollable nodes (x_2).
- Introduce an auxiliary Boolean variable d_i for each controllable node x_i (d_1, d_3). If $d_i = 1$, node x_i is fixed to a_i. If $d_i = 0$, node x_i is updated by using its original Boolean function.
- Let $\mathcal{L}$ be the ASP characterizing the trap spaces of $\mathcal{N}$ considering all d_i. Let C be the constraint characterizing the max. trap space M ($x_1 = 1 \wedge x_3 = 0$).
- Compute the set A of minimal answer sets (projecting only d_i not x_i) of $\mathcal{L} + C$. $A = \{\{d_1\}, \{d_3\}\}$
- Compute the set A' of minimal answer sets (projecting only d_i not x_i) of $\mathcal{L} + \neg C + A$. $A' = \{\{d_1\}\}$
- Return $A \setminus A'$. Return $\{\{d_3\}\}$ where $\{d_3\}$ corresponds to $\{x_3 = 0\}$

Ideas for driver node identification

This ASP-based method can avoid considering explicitly all subsets of size 1, 2, ..., $|M| - 1$. This exploits the power of ASP solvers (e.g., clingo).

However, it is still needed to test its efficiency in practice. Moreover, we also need to investigate its correctness thoroughly.

Bottlenecks of both pystablemotifs and mtsNFVS

The new approach still encounters the two bottlenecks of both pystablemotifs and mtsNFVS.

Bottlenecks of both pystablemotifs and mtsNFVS

The case of many source nodes where there are actually too many attractors (at least 2^k where k is the number of source nodes). mtsNFVS is less affected than pystablemotifs, but this is still problem because mtsNFVS uses the set of explicit (not symbolic) min. trap spaces.

We have found a wise way to symbolically represent the maximal answer sets (corresponding to min. trap spaces) of the encoded ASP characterizing all trap spaces.

For max. trap spaces, we need to adjust the current method. We are thinking about this.

Bottlenecks of both pystablemotifs and mtsNFVS

There exist some very huge attractors inside min. trap spaces. In this case, the number free variables of a min. trap space is still too large (even all the nodes of the original network). pystablemotifs only ends with min. trap spaces. It is very hard for mtsNFVS to reach the best case (i.e., $|F| = 1$) to avoid the reachability analysis.

~~One example is the model shown in `https://github.com/jcrozum/pystablemotifs/issues/83`. This network has no stable motifs, and we have exactly one attractor for each combination of source node values.~~

Our general idea is as follows.

- We have the information about the min. trap space. We can use this information to guide Preprocessing SSF to reach the easy cases where we can avoid or at least reduce the reachability analysis times.

Deal with the issue of huge attractors

This is a very rare case. We can anticipate it in advance via the information about the number of free nodes of the min. trap space.

In this case, it is harder for Preprocessing SSF to converge into one state (i.e., the best case where we do not need to check the reachability).

However, we know that it is likely that there is only one huge attractor and all candidate states are in this attractor.

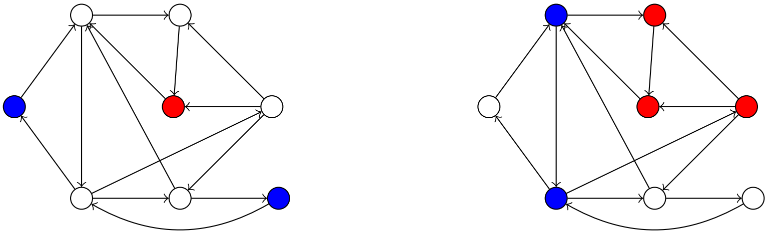
Deal with the issue of huge attractors (cont.)

My idea is as follows.

- First, choose a pivot state s^p from the candidate set F_m.
- Second, from s^p create and gradually enlarge a "trap" to attract other states in F_m. The trap is the set of states that can be reach s^p.
- Third, perform Preprocessing SSF and the enlargement in parallel. When a state converges into the trap, we can exclude it from F_m.
- Finally, we have F_m with the expectation that it contains only the pivot state.

The principle is that larger trap, more possibility to converge it.

Deal with the issue of huge attractors (cont.)



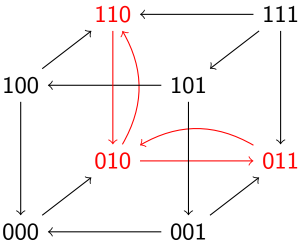
How to efficiently enlarge the trap (i.e., computing predecessors)? Using BDDs as in [Garg et al., 2008, Benes et al., 2021]?

Thanks to time-reversal, we can compute successors (instead of predecessors in the original ABN) in the time-reversal ABN.

This process can purely rely on the function evaluation (not BDDs or unfoldings). Hence, it is expected to be viable for very large models.

Deal with the issue of huge attractors (cont.)

$$\begin{cases} v_1, \neg v_1 \land v_2 \land \neg v_3 \\ v_2, \neg v_1 \lor \neg v_3 \\ v_3, \neg v_1 \land v_2 \land \neg v_3 \end{cases}$$



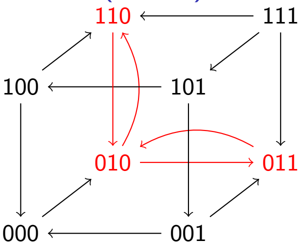
This BN has one min. trap space $m = \{v_1 = \star, v_2 = \star, v_3 = \star\}$. There is one attractor inside m, $\{010, 011, 110\}$.

The time-reversal BN has one max. trap space $\{v_1 = 1, v_2 = \star, v_3 = 1\}$, which is self-negating. Herein, $\{101, 111\}$ is not in the attractor.

Applying the approach of mtsNFVS with $U^{-}_{min} = \{v_1, v_3\}$ and $b_1 = 1, b_3 = 1$. Then, $F = \{011, 110, 101\}$. We can exclude 101 from F before any further analysis.

Deal with the issue of huge attractors (cont.)

$$\begin{cases} v_1, \neg v_1 \wedge v_2 \wedge \neg v_3 \\ v_2, \neg v_1 \vee \neg v_3 \\ v_3, \neg v_1 \wedge v_2 \wedge \neg v_3 \end{cases}$$



Theorem

Let m be a trap space of a BN N. Let N_m be the BN induced by m and N' be the time-reversal BN of N_m. If s is an attractor state inside m, then s cannot lie in any self-negating max. trap space of N'.

Proof

Assume s is in a self-negating max. trap space M of N'. Then, all states of the attractor containing s are also in M. The states satisfy M but following the transitions of N, they should reach states not satisfying M because M is a self-negating max. trap space of N'. This is a contradiction.

Reachability analysis

We may still need to check the reachability in some cases, for example, there is a motif-avoidant attractor, the Preprocessing SSF is not good enough.

The reachability analysis is the last resort ensuring the correctness of our approach.

The reachability in ABNs is PSPACE-complete. Its research is far from our focus now. Hence, we should use the existing methods for the reachability analysis in a wise manner relying on the information we have already known.

Reachability analysis (cont.)

For example, in the motif-avoidant attractor checking,

- If $|F| = 1$, it is likely that there is a motif-avoidant attractor, the result of the reachability analysis is likely unreachable. In this case, we should first use a static analyzer (e.g., PINT [Paulevé, 2017]) that is efficient for very large models in the case of unreachable.

For example, in the process of each min. trap space m,

- If $|F_m| = 2$, it is likely that there is only one attractor and the result of the reachability analysis is always reachable. In this case, we should first use a bounded model checking-based approach (e.g., SAT, ASP) that is efficient for very large models in the case of reachable. Goal-driven Petri net unfolding [Chatain and Paulevé, 2016] is also a possible option in this case, but I do not think it can handle very large models.

We need to consider this problem thoroughly.

Implementation

Trappist (in Python): .bnet file $\xrightarrow{\textit{pyeda}}$ BDDs of Boolean functions $\rightarrow$ Petri net (digraph) $\rightarrow$ ASP (string) $\xrightarrow{\textit{clingo}}$ min./max. trap spaces (json)

We think that we should start from scratch, i.e., we remove completely the dependency on PyBoolNet because it uses a heavy data structure (i.e, prime-implicants).

We propose to design a completely new system for attractor detection and control. We will gradually adapt the functions from pystablemotifs, mtsNFVS, and Trappist to the new system. Of course, we need a good system design first.

Note that our proposed approach contains many constituent tasks (e.g., computation of min./max. trap spaces, computation of LDOI, driver node identification). The new system should be easy to replace new implementations for each constituent task.

Trappist: new features

Now, Trappist supports computing

- max. trap spaces (original/time-reversal BN)
- min. trap spaces (original/time-reversal BN)
- fixed points (original/time-reversal BN)

Introduction . . .

System design

Design a new system for attractor detection and control (input/output formats, components, functions, etc.).

Discuss ...

Experimentation

We plan to test the new method on models without constant nodes.

N-K models: expected to handle networks with $K = 2$ and up to 500 nodes.

scale-free models: expected to handle networks with $K = 10$ and up to 5000 nodes.

real-world models: expected to handle all the largest and most complex networks that we can find in the literature. The repository [2] maintained by Samuel Pastva is a good source.

[2] https://github.com/sybila/biodivine-boolean-models

Conclusion

The new approach exploits advantages of both pystablemotifs and mtsNFVS.

It benefits not only the attractor detection issue but also the control issue of asynchronous Boolean networks.

We believe that it will be the most promising approach that can reach the genome scale.

Collaboration

We hope we will have a nice collaboration on this work. Hence, we should make clear something at the beginning.

Participants:

- Our side: Van-Giang Trinh and Samuel Pastva.
- Your side: Jordan Rozum and Kyu Hyong Park.

Authorship (if we can publish some papers :)):

- Two first authors, one from our side and one from your side with the equal contribution?

What do you think?

Next steps

Task 1: Complete our new approach and make it as detailed as possible.

- Find solutions for some hard problems (e.g., the case of very huge attractors).
- Prepare more specific slides (or a report) presenting technical details of the new approach.
- Add more concrete examples for illustration.

Task 2: Start the implementation of our new approach.

- Design a new system for attractor detection and control (input/output formats, components, functions, etc.).
- Programming

We should create a Github repository first. Everything will be updated there.

Our proposal

I and Samuel will focus on Task 1. Of course, Jordan and Kyu will also involve in this task to make the new approach as powerful as possible.

Jordan and Kyu will lead Task 2 because you already had the holistic view about pystablemotifs. You can divide and assign to all of them implementation sub-tasks. I and Samuel will contribute to this process.

Should keep regular exchanges between us.

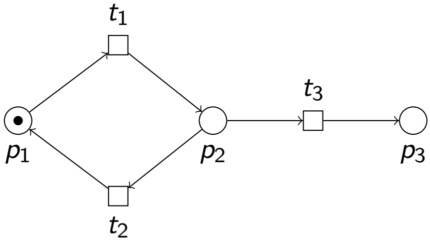
Your suggestions?

Thank you for your attention!

Petri nets

Bipartite graph

Places P
Transitions T
Weighted arcs W



A marking for a Petri net is a mapping $m : P \mapsto \mathbb{N}$ that assigns a number of tokens to each place. A place p is marked by a marking m if and only if $m(p) > 0$. For example, p_1 is marked, p_2 and p_3 are unmarked.

We shall write $pred(x)$ (resp. $succ(x)$) to represent the set of vertices that have a (non-zero weighted) arc leading to (resp. coming from) x. For example, $pred(p_2) = \{t_1\}$, $succ(p_2) = \{t_2, t_3\}$, $pred(t_2) = \{p_2\}$, and $succ(t_2) = \{p_1\}$.

Siphons

Siphon

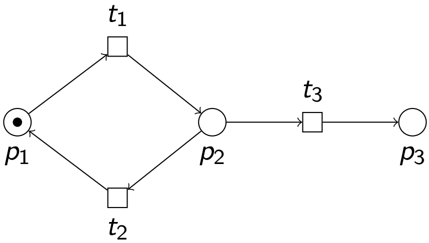
A siphon of a Petri net (P, T, W) is a set of places S such that:

$$\forall t \in T, S \cap succ(t) \neq \emptyset \Rightarrow S \cap pred(t) \neq \emptyset.$$

Once a siphon is unmarked, it remains unmarked.

$S = \{p_1, p_3\}$ is not a siphon because $S \cap succ(t_3) = \{p_1, p_3\} \cap \{p_3\} = \{p_3\} \neq \emptyset$ but $S \cap pred(t_3) = \{p_1, p_3\} \cap \{p_2\} = \emptyset$.

Here: $\emptyset$, $\{p_1, p_2\}$, $\{p_1, p_2, p_3\}$



Traps

Trap

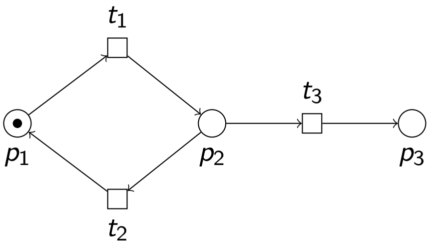
A trap of a Petri net (P, T, W) is a set of places S such that:

$$\forall t \in T, S \cap pred(t) \neq \emptyset \Rightarrow S \cap succ(t) \neq \emptyset.$$

Once a trap is marked (i.e., at least one of its places is marked), it remains marked.

$S = \{p_1, p_3\}$ is not a trap because $S \cap pred(t_1) = \{p_1, p_3\} \cap \{p_1\} = \{p_1\} \neq \emptyset$ but $S \cap succ(t_1) = \{p_1, p_3\} \cap \{p_2\} = \emptyset$.

Here: $\emptyset$, $\{p_1, p_2, p_3\}$



Petri net of a Boolean model

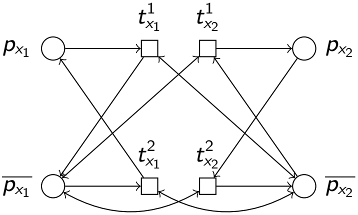
The original encoding was established in [Chaouiya et al., 2004].

Two places for each gene: $v \rightsquigarrow p_v, \overline{p_v}$

Solutions of $f_v \not\leftrightarrow v \rightsquigarrow$ transitions from p_v to $\overline{p_v}$ (and back)

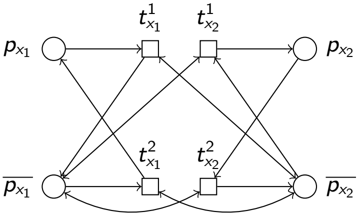
At any marking m of the Petri net encoding a Boolean model, it always holds that $m(p_v) + m(\overline{p}_v) = 1$.

$$\begin{cases} f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \\ f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \end{cases}$$



Conflict-free siphons/traps

A siphon/trap is called **conflict-free** if it does not contain both p_v and $\overline{p_v}$ for all $v \in V$.



Siphon	Conflict-free?
$\emptyset$	yes
$\{\overline{p_{x_1}}, \overline{p_{x_2}}\}$	yes
$\{p_{x_1}, \overline{p_{x_1}}\}$	no
$\{p_{x_2}, \overline{p_{x_2}}\}$	no

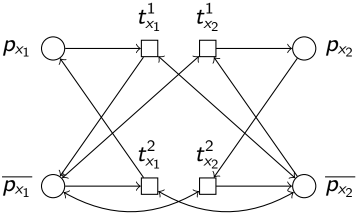
A conflict-free siphon (resp. trap) is maximal if it is not a subset of any other conflict-free siphon (resp. trap).

Conflict-free siphons are trap spaces

Theorem 1

Let $\mathcal{M}$ be a Boolean model and $\mathcal{P}$ be its Petri net encoding. There is a one-to-one correspondence between the set of **trap spaces** of $\mathcal{M}$ and the set of **conflict-free siphons** of $\mathcal{P}$.

$$\begin{cases} f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \\ f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \end{cases}$$



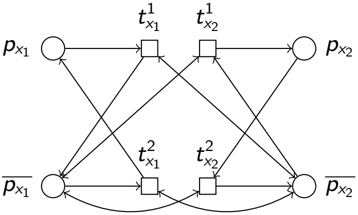
Trap space	Conflict-free siphon
$\star\star$	$\emptyset$
11	$\{\overline{p_{x_1}}, \overline{p_{x_2}}\}$

Maximal conflict-free siphons are minimal trap spaces

Theorem 2

Let $\mathcal{M}$ be a Boolean model and $\mathcal{P}$ be its Petri net encoding. There is a one-to-one correspondence between the set of **minimal trap spaces** of $\mathcal{M}$ and the set of **maximal conflict-free siphons** of $\mathcal{P}$.

$$\begin{cases} f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \\ f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2) \end{cases}$$



Trap space	Conflict-free siphon
$\star\star$	$\emptyset$
11	$\{\overline{p_{x_1}}, \overline{p_{x_2}}\}$

Proposed method for minimal trap space computation

From Theorem 2, we propose a new method for computing minimal trap spaces of a Boolean model $\mathcal{M}$.

- Build the Petri net encoding $\mathcal{P}$ of $\mathcal{M}$.
- Compute all maximal conflict-free siphons of $\mathcal{P}$.
- Convert the obtained maximal conflict-free siphons into the corresponding minimal trap spaces.

Petri net transformation

Transforming a Boolean model into its Petri net encoding can be done via computing Disjunctive Normal Forms (DNF) of each Boolean function [Chatain et al., 2014]. This transformation is implemented in the bioLQM[3] library using BDDs.

Though this might appear quite computationally intensive it is important to remark first that contrary to the prime-implicants case, there is no need to find minimal DNFs.

We use the above transformation in our proposed method.

[3] http://www.colomoto.org/biolqm/

Maximal conflict-free siphon computation

Characterize all siphons of the encoded Petri net as a system of Boolean rules.

$$p \in S \Rightarrow \bigvee_{p' \in pred(t)} p' \in S, p \in P, t \in T, t \in pred(p)$$

Add to the system the Boolean rules representing the conflict-freeness.

$$p_v \in S \Rightarrow \overline{p_v} \notin S \wedge \overline{p_v} \in S \Rightarrow p_v \notin S, v \in V$$

Encode the system as an ASP.

Use an ASP solver (e.g., clingo [Gebser et al., 2011]) to compute all set-inclusion maximal answer sets of the ASP.

Set-maximality through "heuristics"

```
clingo --heuristic=Domain --enum-mod=domRec --dom-mod=3
```

References I

Akutsu, T. (2018).
Algorithms for analysis, inference, and control of Boolean networks.
World Scientific, Singapore.

Benes, N., Brim, L., Pastva, S., and Safránek, D. (2021).
Computing bottom SCCs symbolically using transition guided reduction.
In International Conference on Computer Aided Verification, pages 505–528. Springer.

Biane, C. and Delaplace, F. (2019).
Causal reasoning on Boolean control networks based on abduction: Theory and application to cancer drug discovery.
IEEE/ACM Transactions on Computational Biology and Bioinformatics, 16(5):1574–1585.

References II

Brim, L., Pastva, S., Šafránek, D., and Šmijáková, E. (2021).
Parallel one-step control of parametrised boolean networks.
Mathematics, 9(5):560.

Chaouiya, C., Remy, E., Ruet, P., and Thieffry, D. (2004).
Qualitative modelling of genetic networks: From logical regulatory graphs to standard Petri nets.
In Cortadella, J. and Reisig, W., editors, Applications and Theory of Petri Nets 2004, 25th International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004, Proceedings, volume 3099 of Lecture Notes in Computer Science, pages 137–156. Springer.

References III

Chatain, T., Haar, S., Jezequel, L., Paulevé, L., and Schwoon, S. (2014).
Characterization of reachable attractors using Petri net unfoldings.
In Mendes, P., Dada, J. O., and Smallbone, K., editors, Computational Methods in Systems Biology - 12th International Conference, CMSB 2014, Manchester, UK, November 17-19, 2014, Proceedings, volume 8859 of Lecture Notes in Computer Science, pages 129–142. Springer.

Chatain, T. and Paulevé, L. (2016).
Goal-driven unfolding of Petri nets.

Cifuentes-Fontanals, L., Tonello, E., and Siebert, H. (2021).
Control in Boolean networks with model checking.
arXiv preprint arXiv:2112.10477.

References IV

Fontanals, L. C., Tonello, E., and Siebert, H. (2020).
Control strategy identification via trap spaces in Boolean networks.
In Abate, A., Petrov, T., and Wolf, V., editors, Computational Methods in Systems Biology - 18th International Conference, CMSB 2020, Konstanz, Germany, September 23-25, 2020, Proceedings, volume 12314 of Lecture Notes in Computer Science, pages 159–175. Springer.

Garg, A., Cara, A. D., Xenarios, I., Mendoza, L., and Micheli, G. D. (2008).
Synchronous versus asynchronous modeling of gene regulatory networks.
Bioinform., 24(17):1917–1925.

References V

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., and Schneider, M. (2011).
Potassco: The Potsdam answer set solving collection.
AI Commun., 24(2):107–124.

Giang, T. V. and Hiraishi, K. (2021).
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.

Klarner, H., Streck, A., and Siebert, H. (2017).
PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks.
Bioinform., 33(5):770–772.

References VI

Mizera, A., Pang, J., Qu, H., and Yuan, Q. (2019).
Taming asynchrony for attractor detection in large Boolean networks.
IEEE ACM Trans. Comput. Biol. Bioinform., 16(1):31–42.

Moon, K., Lee, K., Chopra, S., and Kwon, S. (2022).
Bilevel integer programming on a Boolean network for discovering critical genetic alterations in cancer development and therapy.
European Journal of Operational Research, 300(2):743–754.

Naldi, A., Monteiro, P. T., and Chaouiya, C. (2012).
Efficient handling of large signalling-regulatory networks by focusing on their core control.
In Gilbert, D. R. and Heiner, M., editors, Computational Methods in Systems Biology - 10th International Conference, CMSB 2012, London, UK, October 3-5, 2012. Proceedings, volume 7605 of Lecture Notes in Computer Science, pages 288–306. Springer.

References VII

Paulevé, L. (2017).
Pint: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface.
In CMSB 2017 - 15th conference on Computational Methods for Systems B
volume 10545 of Lecture Notes in Computer Science, pages 370 – 316. Springer.

Rozum, J. C., Deritei, D., Park, K. H., Gómez Tejeda Zañudo, J., and Albert, R. (2021a).
pystablemotifs: Python library for attractor identification and control in Boolean networks.
Bioinform.

References VIII

Rozum, J. C., Zañudo, J. G. T., Gan, X., Deritei, D., and Albert, R. (2021b).
Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks.
Sci. Adv., 7(29).

Saadatpour, A., Albert, R., and Reluga, T. C. (2013).
A reduction method for Boolean network models proven to conserve attractors.
SIAM J. Appl. Dyn. Syst., 12(4):1997–2011.

Schwab, J. D., Kühlwein, S. D., Ikonomi, N., Kühl, M., and Kestler, H. A. (2020).
Concepts in Boolean network modeling: What do they all mean?
Comput. Struct. Biotechnol. J., 18:571–582.

References IX

Su, C. and Pang, J. (2021a).
Cabean 2.0: Efficient and efficacious control of asynchronous Boolean networks.
In International Symposium on Formal Methods, pages 581–598. Springer.

Su, C. and Pang, J. (2021b).
CABEAN: a software for the control of asynchronous boolean networks.
Bioinform., 37(6):879–881.

Trinh, G. V., Akutsu, T., and Hiraishi, K. (2020).
An FVS-based approach to attractor detection in asynchronous random Boolean networks.
IEEE/ACM Trans. Comput. Biol. Bioinf.
in press.

References X

Trinh, V., Benhamou, B., Hiraishi, K., and Soliman, S. (2022a).
Minimal trap spaces of logical models are maximal siphons of their petri net encoding.
In Petre, I. and Paun, A., editors, Computational Methods in Systems Biology - 20th International Conference, CMSB 2022, Bucharest, Romania, September 14-16, 2022, Proceedings, volume 13447 of Lecture Notes in Computer Science, pages 158–176. Springer.

Trinh, V., Hiraishi, K., and Benhamou, B. (2022b).
Computing attractors of large-scale asynchronous boolean networks using minimal trap spaces.
In BCB '22: 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Northbrook, Illinois, USA, August 7 - 10, 2022, pages 13:1–13:10. ACM.

References XI

Zañudo, J. G. T. and Albert, R. (2015).
Cell fate reprogramming by control of intracellular network dynamics.
PLOS Computational Biology, 11(4):e1004193.

Zheng, D., Yang, G., Li, X., Wang, Z., Liu, F., and He, L. (2013).
An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks.
PloS One, 8(4):e60593.