Efficient enumeration of fixed points in complex Boolean networks using answer set programming

Van-Giang Trinh[1], Belaid Benhamou[1], and Sylvain Soliman[2]

[1]LIRICA team, LIS, Aix-Marseille University, Marseille, France
[2]Lifeware team, Inria Saclay center, Palaiseau, France

November 20, 2023

Van-Giang Trinh, Belaid Benhamou, & Sylvain Soliman (2023). Efficient enumeration of fixed points in complex Boolean networks using answer set programming. In *International Conference on Principles and Practice of Constraint Programming* (pp. 35:1–35:19).

Boolean network

$\mathcal{N} = (V, F)$, where $V = \{v_1, ..., v_n\}$ is a set of nodes and $F = \{f_1, ..., f_n\}$ is a set of associated Boolean functions.

At time t, node $v_i \in V$ can update its state by $s_{t+1}(v_i) = f_i(s_t)$.

$$\begin{cases} f_1 = (v_1 \land v_2) \lor (\neg v_1 \land \neg v_2) \\ f_2 = (v_1 \land v_2) \lor (\neg v_1 \land \neg v_2) \end{cases}$$

Boolean network

$$\begin{tikzcd}
01 \arrow[loop right] \arrow[d, bend left] & 11 \arrow[loop right] \\
00 \arrow[u, bend left] \arrow[r, bend right] & 10 \arrow[loop right] \arrow[l, bend right]
\end{tikzcd}$$

Fully asynchronous dynamics

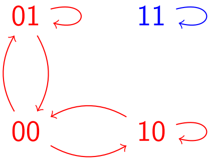
Update schemes:

- Fully asynchronous: only one node is non-deterministically selected to update at each time step.
- Synchronous: all nodes are selected to update at each time step.
- . . .

Attractors

An *attractor* is a minimal non-empty set of states from which the system cannot escape once entered.

$$\begin{cases} f_1 = (v_1 \land v_2) \lor (\neg v_1 \land \neg v_2) \\ f_2 = (v_1 \land v_2) \lor (\neg v_1 \land \neg v_2) \end{cases}$$


$$\begin{tikzcd} 01 \arrow[loop right] \arrow[d, bend left] & 11 \arrow[loop right] \\ 00 \arrow[u, bend left] \arrow[r, bend right] & 10 \arrow[loop right] \arrow[l, bend right] \end{tikzcd}$$

State set	Attractor	Type
$\{11\}$	yes	fixed point
$\{00, 01\}$	no	-
$\{00, 01, 10\}$	yes	cyclic
$\{00, 01, 10, 11\}$	no	-

Application

Many applications in systems biology, since attractors correspond to biological *phenotypes*:

- new insights into the origins of diseases: cancers, SARS-CoV-2, HIV
- aid the development of new drugs
- starting point for many control approaches for biological systems, which play an important role in systems medicine

Applications in many other fields:

- computer science
- mathematics
- theoretical physics
- complex systems
- ...

Fixed points vs. cyclic attractors

To date, the analysis of fixed points remains a very useful/standard tool in understanding the behavior of complex biological models.

- in some cases the full computation of cyclic attractors remains intractable
- for many biological systems, the expected long-term behavior is not cyclic (as in the Cell Cycle, or Circadian rhythms for instance) but rather a stabilization to an observable *phenotype*
- fixed points are independent of the update scheme, but cyclic attractors are not
- crucial starting point for the state-of-the-art for computing cyclic attractors of BNs [Trinh et al., 2022]

More applications: coding theory, control theory, neural networks.

Fixed point enumeration

Characterization and complexity

A state s is a *fixed point* of $\mathcal{N}$ if and only if $s(v_i) = f_i(s)$ for every $v_i \in V$.

The problems of detecting a fixed point and enumerating all fixed points of a general Boolean network have been shown to be respectively NP-hard and #P-hard [Akutsu et al., 1998].

Limitations

The fixed point enumeration problem has attracted researchers from various communities and many methods have been proposed [Mori and Akutsu, 2022].

With the constant increase in model size and complexity of Boolean update functions, the existing methods show their limitations.

State-of-the-art	Bottleneck	Remark
[Klarner et al., 2017]	prime implicants	hard to obtain + large number
[Paulevé et al., 2020]	DNF + locally-monotonic	sometimes hard to obtain + not handle general models
[Abdallah et al., 2017]	transition-based representation	# transitions may be exponential in the number of input nodes

Answer set programming and systems biology

Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988] has been widely applied to the field of systems biology [Videla et al., 2015].

Naturally ASP has been quickly applied to modeling and analysis of Boolean networks.

- fixed point enumeration [Klarner et al., 2017, Abdallah et al., 2017, Paulevé et al., 2020]
- attractor enumeration [Mushthofa et al., 2014, Klarner et al., 2017, Abdallah et al., 2017, Paulevé et al., 2020]
- inference from biological data [Rocca et al., 2014, Videla et al., 2015, Videla et al., 2017, Chevalier et al., 2020]
- control [Kaminski et al., 2013, Videla et al., 2017]

Answer set programming and systems biology

The most recent and most efficient fixed point enumeration methods all rely on answer set programing [Klarner et al., 2017, Abdallah et al., 2017, Paulevé et al., 2020].

$\Longrightarrow$ We propose two new ASP-based methods for efficiently enumerating fixed points in a Boolean network.

ASP-based methods for enumerating fixed points

Core ASP encoding

We intend to build a logic program (say $\mathcal{P}$) for $\mathcal{N}$ such that its set of stable models one-to-one corresponds to the set of fixed points of $\mathcal{N}$.

For each node v_i, we introduce two atoms p_i and n_i.

The below ASP rules ensure that a stable model of $\mathcal{P}$ corresponds to a state of $\mathcal{N}$:

$$\leftarrow p_i \wedge n_i \tag{1}$$

and

$$p_i \vee n_i \leftarrow \tag{2}$$

The translation from a stable model A of $\mathcal{P}$ to a state x of $\mathcal{N}$ is that for every $v_i \in V$,

$$\begin{cases} x(v_i) = 1 \text{ iff } p_i \in A, \\ x(v_i) = 0 \text{ iff } n_i \in A. \end{cases}$$

Core ASP encoding

Fixed points can be characterized by the conjunction of $v_i \leftarrow f_i$ and $\neg v_i \leftarrow \neg f_i$. We encode the two parts for every $v_i \in V$ as ASP rules.

To avoid the presence of negation, we use the Negative Normal Form (NNF) of a Boolean function.

The NNF is obtained by recursively applying De Morgan laws until all negations that remain are on only literals.

$$\neg(v_3 \vee \neg(v_1 \wedge v_2)) \Rightarrow \neg(v_3 \vee \neg v_1 \vee \neg v_2) \Rightarrow \neg v_3 \wedge v_1 \wedge v_2$$

NNF is much more efficient to obtain than DNF, CNF, or BDD.

Core ASP encoding

$$v_i \leftarrow f_i$$
$$\Rightarrow$$
$$\gamma(v_i) \leftarrow \gamma(\mathsf{NNF}(f_i))$$

where we define function γ as

$$\gamma(v_i) = p_i$$
$$\gamma(\neg v_i) = n_i$$
$$\gamma(\bigwedge_{1\leq j\leq J} \alpha_j) = \gamma(\alpha_1) \wedge \ldots \wedge \gamma(\alpha_J)$$
$$\gamma(\bigvee_{1\leq j\leq J} \alpha_j) = aux_k$$

where aux_k is a new auxiliary atom and for each j add the rule $aux_k \leftarrow \gamma(\alpha_j)$.

Core ASP encoding

$$\neg v_i \leftarrow \neg f_i$$

$$\Rightarrow$$

$$\gamma(\neg v_i) \leftarrow \gamma(\mathrm{NNF}(\neg f_i))$$

Theorem

The set of stable models of $\mathcal{P}$ one-to-one corresponds to the set of fixed points of $\mathcal{N}$.

Example (written in Clingo's syntax)

$$\begin{tikzcd}
01 \arrow[loop right] \arrow[d, bend left] & 11 \arrow[loop right] \\
00 \arrow[u, bend left] \arrow[r, bend right] & 10 \arrow[loop right] \arrow[l, bend right]
\end{tikzcd}$$

$$\begin{cases} f_1 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2) \\ f_2 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2) \end{cases}$$

```
:- p1, n1.                              :- p2, n2.
p1, n1.                                 p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                         aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                             aux2 :- n2.
aux3 :- p1.                             aux3 :- p2.

p2 :- aux4.
aux4 :- p1, p2.                         aux4 :- n1, n2.
```

Example (written in Clingo's syntax)

```
:- p1, n1.                            :- p2, n2.
p1, n1.                               p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                       aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                           aux2 :- n2.
aux3 :- p1.                           aux3 :- p2.

p2 :- aux4.
aux4 :- p1, p2.                       aux4 :- n1, n2.
n2 :- aux5, aux6.
aux5 :- n1.                           aux5 :- n2.
aux6 :- p1.                           aux6 :- p2.

#show p1/0.  #show n1/0.              #show p2/0.  #show n2/0.
```

Example (written in Clingo's syntax)

```
:- p1, n1.                              :- p2, n2.
p1, n1.                                 p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                         aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                             aux2 :- n2.
aux3 :- p1.                             aux3 :- p2.

p2 :- aux4.
aux4 :- p1, p2.                         aux4 :- n1, n2.
n2 :- aux5, aux6.
aux5 :- n1.                             aux5 :- n2.
aux6 :- p1.                             aux6 :- p2.

#show p1/0. #show n1/0.                 #show p2/0. #show n2/0.
```

Example (written in Clingo's syntax)

```
:- p1, n1.                        :- p2, n2.
p1, n1.                           p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                   aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                       aux2 :- n2.
aux3
```

$v_1 \leftarrow f_1$ with $f_1 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2)$

```
p2 :- aux4.
aux4 :- p1, p2.                   aux4 :- n1, n2.
n2 :- aux5, aux6.
aux5 :- n1.                       aux5 :- n2.
aux6 :- p1.                       aux6 :- p2.

#show p1/0.  #show n1/0.          #show p2/0.  #show n2/0.
```

Example (written in Clingo's syntax)

```
:- p1, n1.                          :- p2, n2.
p1, n1.                             p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                     aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                         aux2 :- n2.
aux3 :- p1.                         aux3 :- p2.

p2 :- aux4
aux4
n2 :- aux5, aux6.
aux5 :- n1.                         aux5 :- n2.
aux6 :- p1.                         aux6 :- p2.

#show p1/0.  #show n1/0.            #show p2/0.  #show n2/0.
```

$\neg v_1 \leftarrow \neg f_1$ with $\neg f_1 = (\neg v_1 \vee \neg v_2) \wedge (v_1 \vee v_2)$

Example (written in Clingo's syntax)

```
:- p1, n1.                            :- p2, n2.
p1, n1.                               p2, n2.

p1 :- aux1
aux1
n1 :- aux2, aux3.
aux2 :- n1.                           aux2 :- n2.
aux3 :- p1.                           aux3 :- p2.

p2 :- aux4.
aux4 :- p1, p2.                       aux4 :- n1, n2.
n2 :- aux5, aux6.
aux5 :- n1.                           aux5 :- n2.
aux6 :- p1.                           aux6 :- p2.

#show p1/0.  #show n1/0.              #show p2/0.  #show n2/0.
```

$f_2 = f_1 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2) \Rightarrow$ similar ASP rules for node v_2

Example (written in Clingo's syntax)

```
:- p1, n1.                          :- p2, n2.
p1, n1.                             p2, n2.

p1 :- aux1.
aux1 :- p1, p2.                     aux1 :- n1, n2.
n1 :- aux2, aux3.
aux2 :- n1.                         aux2 :- n2.
aux3 :- p1.                         aux3 :- p2.

p2 :-
aux4 :- p1, p2.                     aux4 :- n1, n2.
n2 :- aux5, aux6.
aux5 :- n1.                         aux5 :- n2.
aux6 :- p1.                         aux6 :- p2.
```

Exclude auxiliary atoms from stable models.

```
#show p1/0.  #show n1/0.            #show p2/0.  #show n2/0.
```

Example (written in Clingo's syntax)

```
:- p1, n1.
p1, n1.

p1 :- aux1.
aux1 :- p1, p2.
n1 :- aux2, aux3.
aux2 :- n1.
aux3 :- p1.

p2 :-
aux4 :- p1, p2.
n2 :- aux5, aux6.
aux5 :- n1.
aux6 :- p1.

#show p1/0.  #show n1/0.
```

```
:- p2, n2.
p2, n2.


aux1 :- n1, n2.

aux2 :- n2.
aux3 :- p2.


aux4 :- n1, n2.

aux5 :- n2.
aux6 :- p2.

#show p2/0.  #show n2/0.
```

One stable model $\{p_1, p_2\} \sim$ fixed point 11

Problem with source nodes

Node $v_i \in V$ is called a *source* node if and only if $f_i = v_i$.

The number of fixed points of a Boolean network may be extremely large if it has many source nodes. Might be exponential in the number of source nodes.

In the core encoding as well as those of the state-of-the-art methods, a resulting stable model always corresponds to a single fixed point.

A bottleneck in number of source nodes $\Longrightarrow$ new method to overcome this

New method

$$\begin{cases} f_1 = v_1 \\ f_2 = v_1 \lor v_2 \end{cases}$$

$$\begin{tikzcd}
01 \arrow[loop right] & 11 \arrow[loop right] \\
00 \arrow[loop right] & 10 \arrow[u] \arrow[loop right]
\end{tikzcd}$$

Fixed point	**Stable model**
00	$\{n_1, n_2\}$
01	$\{n_1, p_2\}$
11	$\{p_1, p_2\}$

New method

Fixed point	Stable model
00	$\{n_1, n_2\}$
01	$A_1 = \{n_1, p_2\}$
11	$A_2 = \{p_1, p_2\}$
$\Rightarrow$ 01, 11	$A = \{p_1, n_1, p_2\}$

Our main idea is to group two stable models A_1 and A_2 of $\mathcal{P}$ into a Herbrand model A if they only differ in the atoms corresponding to a source node.

We add A to the set of stable models of $\mathcal{P}$, and then repeat the grouping process until there is no new stable model.

A covers all the fixed points represented by the two stable models constituting it. $\Rightarrow$ maximal set-inclusion stable models.

New method

We adjust the core encoding to make the above approach fully automated in the ASP solver.

- removing the condition $\leftarrow p_i \wedge n_i$
- adding *choice* rules for only atoms corresponding to source nodes (i.e., $p_i \leftarrow$ not not p_i and $n_i \leftarrow$ not not n_i) $\Rightarrow$ making A to be a stable model

Theorem

The set of maximal set-inclusion stable models of $\mathcal{P}$ fully covers all fixed points of the Boolean network.

Post-processing

A stable model can be group-able with multiple ones, thus one fixed point can belong to multiple maximal set-inclusion stable models.

A binary decision diagram to symbolically represent the set of maximal set-inclusion stable models.

Meta result for further analysis based on symbolic operators:

- list all fixed points if needed
- count the number of fixed points
- return the set of fixed points of the BN restricted by a given combination of values on source nodes
- ...

Experiments

Python tool `fASP`[1]. ASP solver = Clingo[2]

Our methods:

- `fASP-conj`: the core encoding
- `fASP-src`: modification to handle the case of many source nodes, cannot control the maximum number of resulting fixed points

State-of-the-art methods:

- `PyBoolNet` [Klarner et al., 2017]
- `mpbn` [Paulevé et al., 2020]
- `AN-ASP` [Abdallah et al., 2017]
- `FPCollector` [Aracena et al., 2021]: cannot control the maximum number of resulting fixed points

[1] https://github.com/giang-trinh/fASP

[2] https://github.com/potassco/clingo

Datasets

BBM repository[3]:

- a collection of real-world Boolean models from various sources used in systems biology
- 211 models, peaking at 321 nodes and 133 source nodes

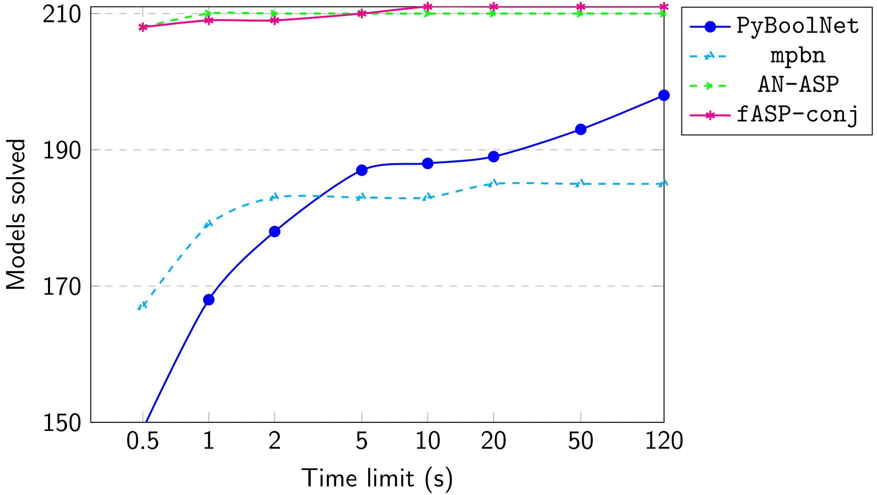
Pseudo-random models:

- structurally similar to the real-world models in the BBM repository
- 400 pseudo-random models ranging from 1000 to 5000 nodes and 127 to 1171 source nodes

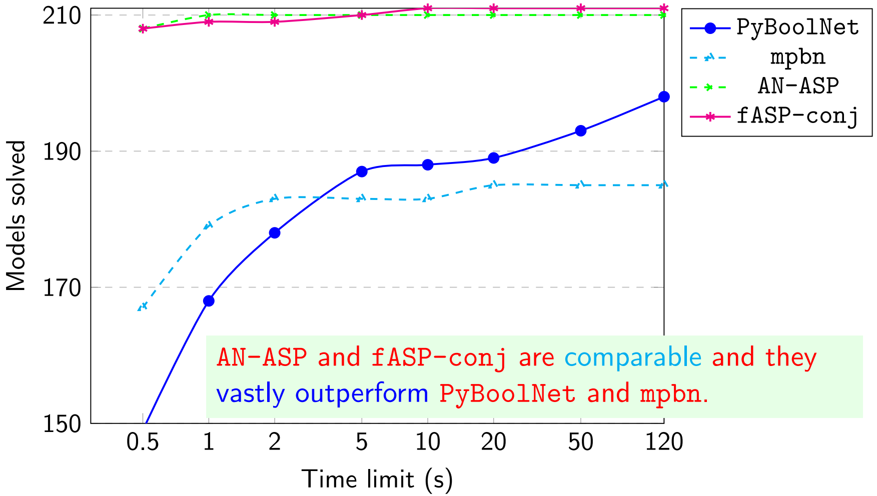
[3] `https://github.com/sybila/biodivine-boolean-models`

Results on real-world models

1000 first fixed points

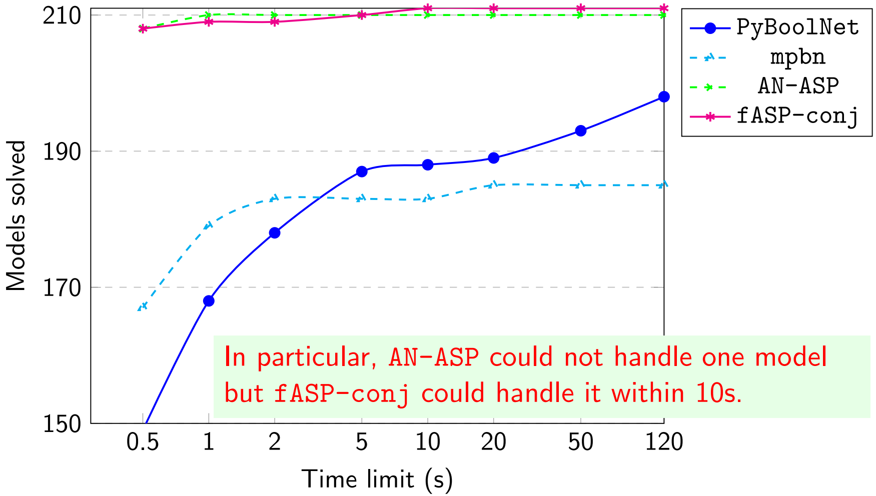


1000 first fixed points



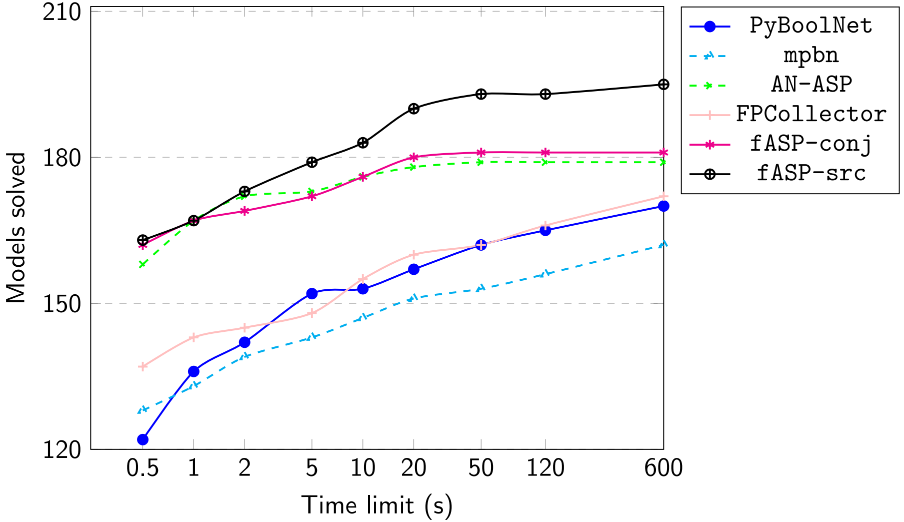
AN-ASP and fASP-conj are comparable and they vastly outperform PyBoolNet and mpbn.

1000 first fixed points

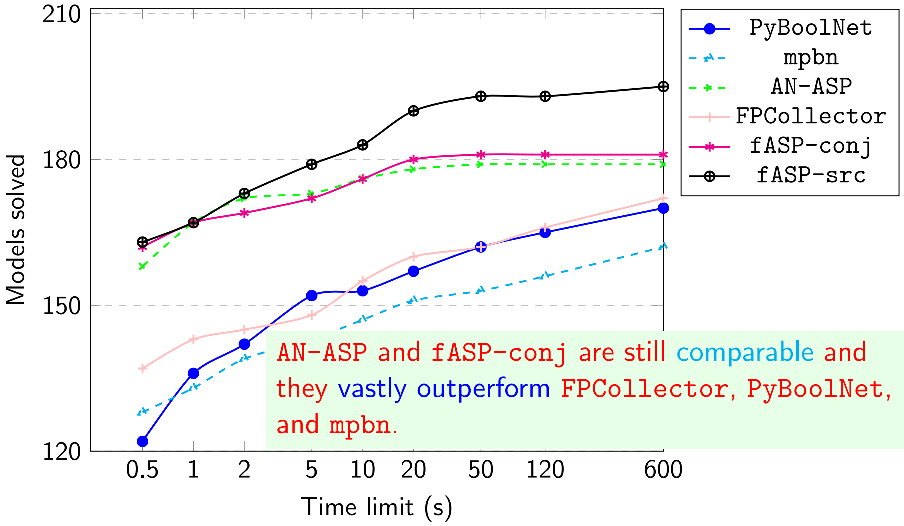


In particular, `AN-ASP` could not handle one model but `fASP-conj` could handle it within 10s.

All fixed points

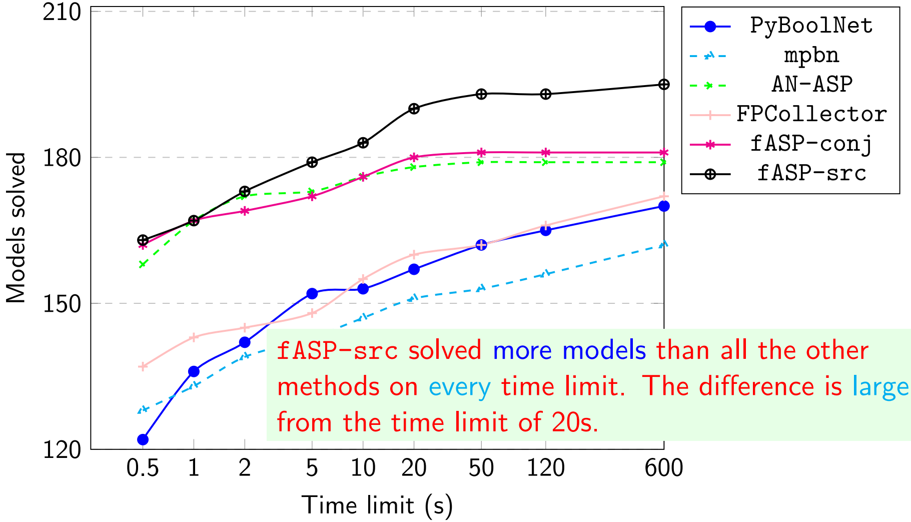


All fixed points



AN-ASP and fASP-conj are still comparable and they vastly outperform FPCollector, PyBoolNet, and mpbn.

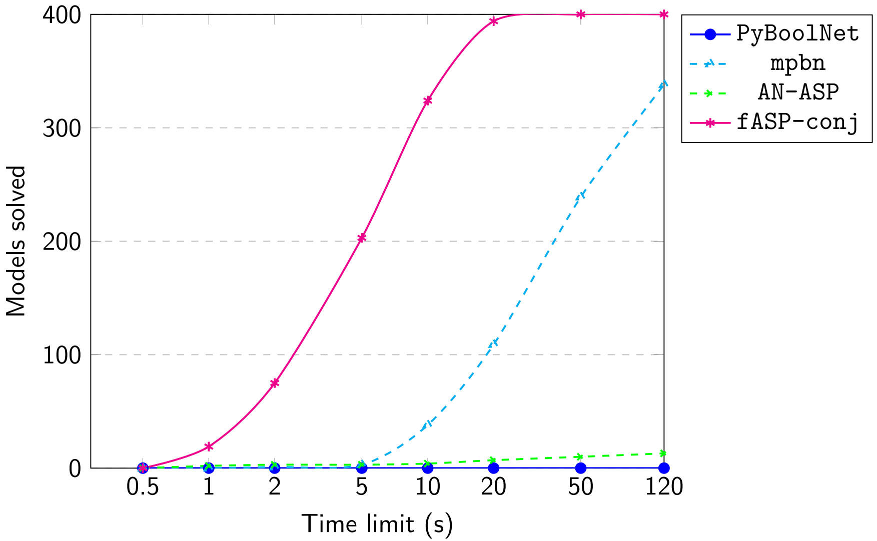
All fixed points



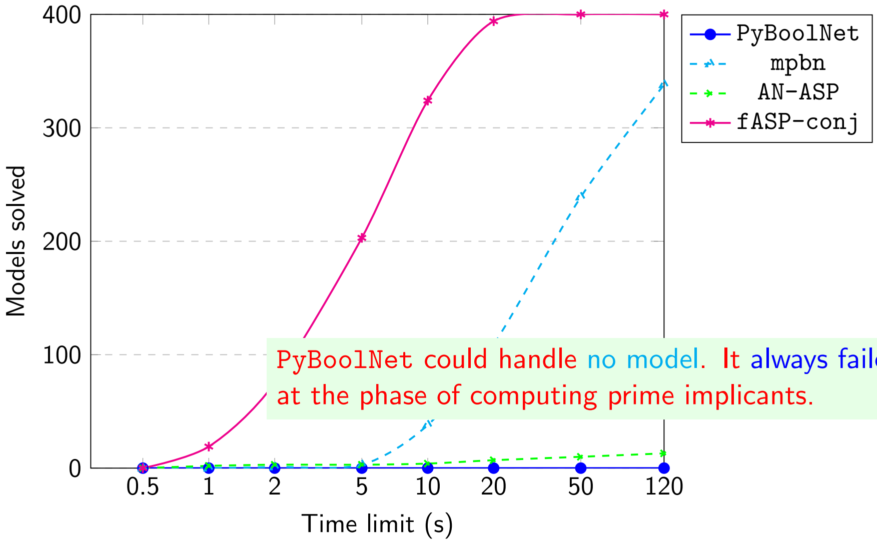
fASP-src solved more models than all the other methods on every time limit. The difference is large from the time limit of 20s.

Results on pseudo-random models

1000 first fixed points

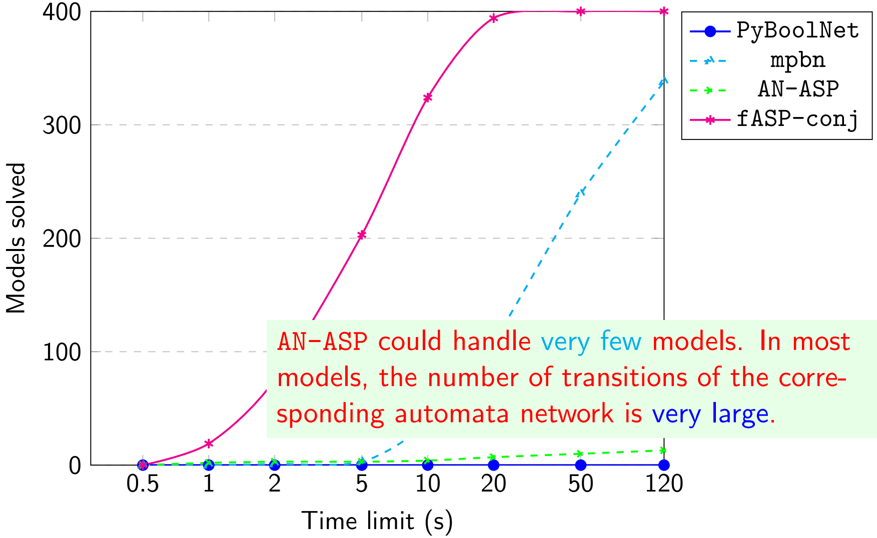


1000 first fixed points



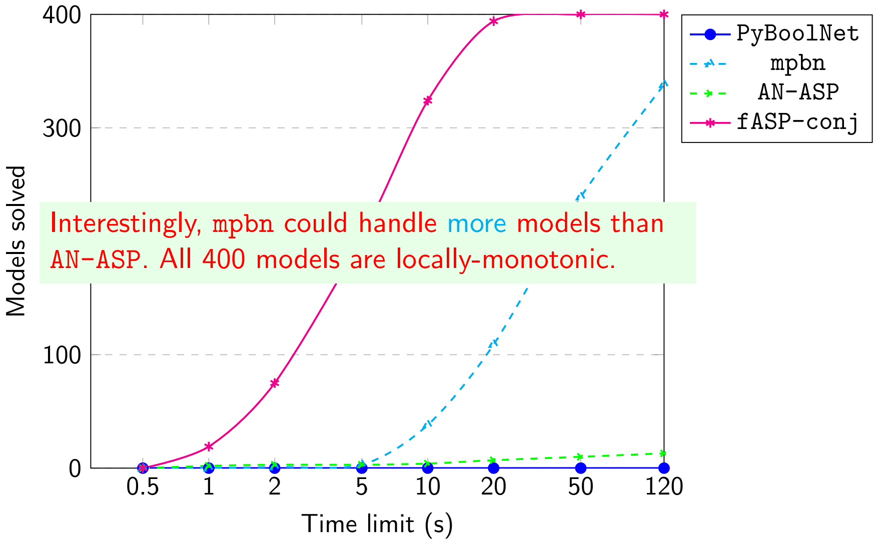
PyBoolNet could handle no model. It always failed at the phase of computing prime implicants.

1000 first fixed points



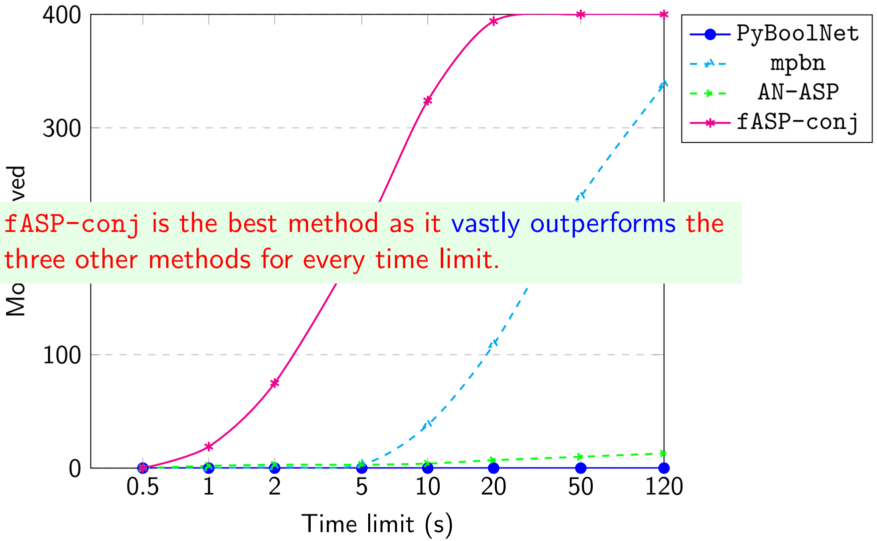
AN-ASP could handle very few models. In most models, the number of transitions of the corresponding automata network is very large.

1000 first fixed points



Interestingly, mpbn could handle more models than AN-ASP. All 400 models are locally-monotonic.

1000 first fixed points



fASP-conj is the best method as it vastly outperforms the three other methods for every time limit.

All fixed points

For every model, all the compared methods failed to obtain all the fixed points as they quickly met the out of memory error.

The reason is that the number of all fixed points (even stable models for the `fASP-src` method) is actually too large due to a lot of source nodes (> 100).

Room for improvement.

Conclusion

Fixed points are important and standard in Boolean network analysis.

Two new methods based on ASP for enumerating fixed points in Boolean networks: `fASP-conj` and `fASP-src`.

Main advantages:

- Both rely on NNFs of Boolean functions, which are much more efficient to obtain than other representations used by previous methods (e.g., prime implicants, DNFs, automata networks).
- `fASP-src` provides a more compact representation of the results based on BDDs, which can give both memory and run-time benefits.

Conclusion

`fASP-conj` and `fASP-src` vastly outperform all the state-of-the-art methods.

In particular, `fASP-src` shows its superiority to all the other methods in enumerating all the fixed points of models with many source nodes.

Conclusion

Whereas `fASP-src` makes use of the unique characteristics of ASP (it doesn't map directly to SAT), it is possible to build an SAT version for `fASP-conj`.

Use a polynomial transformation like our conjunctive ASP encoding or Tseitin's transformation, but this introduces auxiliary variables.

Multiple redundant models may encode the same fixed point.

A step to eliminate redundant SAT models is therefore necessary to guarantee the correctness and this would add complexity to the SAT approach.

Future work

Boolean network models of biological systems usually contain many source nodes, which might be hard to avoid in the modeling process [Aghamiri et al., 2020]. Hence, improving `fASP-src` is necessary.

Implement the SAT version of `fASP-conj` and evaluate its performance on the set of models used in this work.

Extend the proposed methods to those for computing trap spaces of Boolean networks [Klarner et al., 2017], which are more general than fixed points and useful approximations for attractors in Boolean networks.

Thank you for your attention!

References I

Abdallah, E. B., Folschette, M., Roux, O. F., and Magnin, M. (2017).
ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks.
Algorithms Mol. Biol., 12(1):20:1–20:23.

Aghamiri, S. S., Singh, V., Naldi, A., Helikar, T., Soliman, S., Niarakis, A., and Xu, J. (2020).
Automated inference of Boolean models from molecular interaction maps using CaSQ.
Bioinform., 36(16):4473–4482.

Akutsu, T., Kuhara, S., Maruyama, O., and Miyano, S. (1998).
A system for identifying genetic networks from gene expression patterns produced by gene disruptions and overexpressions.
Genome Informatics, 9:151–160.

References II

Aracena, J., Cabrera-Crot, L., and Salinas, L. (2021).
Finding the fixed points of a Boolean network from a positive feedback vertex set.
Bioinform., 37(8):1148–1155.

Chevalier, S., Noël, V., Calzone, L., Zinovyev, A. Y., and Paulevé, L. (2020).
Synthesis and simulation of ensembles of Boolean networks for cell fate decision.
In *International Conference on Computational Methods in Systems Biology*, pages 193–209. Springer.

Gelfond, M. and Lifschitz, V. (1988).
The stable model semantics for logic programming.
In *International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press.

References III

Kaminski, R., Schaub, T., Siegel, A., and Videla, S. (2013).
Minimal intervention strategies in logical signaling networks with ASP.
Theory Pract. Log. Program., 13(4-5):675–690.

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.

Mori, T. and Akutsu, T. (2022).
Attractor detection and enumeration algorithms for Boolean networks.
Comput. Struct. Biotechnol. J., 20:2512–2520.

References IV

Mushthofa, M., Torres, G., de Peer, Y. V., Marchal, K., and Cock, M. D. (2014).
ASP-G: an ASP-based method for finding attractors in genetic regulatory networks.
Bioinform., 30(21):3086–3092.

Paulevé, L., Kolčák, J., Chatain, T., and Haar, S. (2020).
Reconciling qualitative, abstract, and scalable modeling of biological networks.
Nat. Commun., 11(1).

Rocca, A., Mobilia, N., Fanchon, E., Ribeiro, T., Trilling, L., and Inoue, K. (2014).
ASP for construction and validation of regulatory biological networks.
Logical Modeling of Biological Systems, pages 167–206.

References V

Trinh, V., Hiraishi, K., and Benhamou, B. (2022).
Computing attractors of large-scale asynchronous Boolean networks using minimal trap spaces.
In *ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 13:1–13:10. ACM.

Videla, S., Guziolowski, C., Eduati, F., Thiele, S., Gebser, M., Nicolas, J., Saez-Rodriguez, J., Schaub, T., and Siegel, A. (2015).
Learning Boolean logic models of signaling networks with ASP.
Theor. Comput. Sci., 599:79–101.

Videla, S., Saez-Rodriguez, J., Guziolowski, C., and Siegel, A. (2017).
caspo: a toolbox for automated reasoning on the response of logical signaling networks families.
Bioinform., 33(6):947–950.