SAF: SAT-based Attractor Finder in Asynchronous Automata Networks

Takehide Soh, Morgan Magnin, Daniel Le Berre, Mutsunori Banbara, Naoyuki Tamura

SAF: SAT-based Attractor Finder in Asynchronous Automata Networks

Takehide Soh[1][0000−0001−5897−9192], Morgan Magnin[2][0000−0001−5443−0506], Daniel Le Berre[3][0000−0003−3221−9923], Mutsunori Banbara[4][0000−0002−5388−727X], and Naoyuki Tamura[5][0000−0002−5466−1010]

[1] Kobe University, Information Infrastructure and Digital Transformation Initiatives Headquarters, 1-1, Rokko-dai, Nada, Kobe, Hyogo 657-8501 Japan soh@lion.kobe-u.ac.jp

[2] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France morgan.magnin@ec-nantes.fr

[3] Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France leberre@cril-lab.fr

[4] Nagoya University, Graduate School of Informatics, Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan banbara@nagoya-u.jp

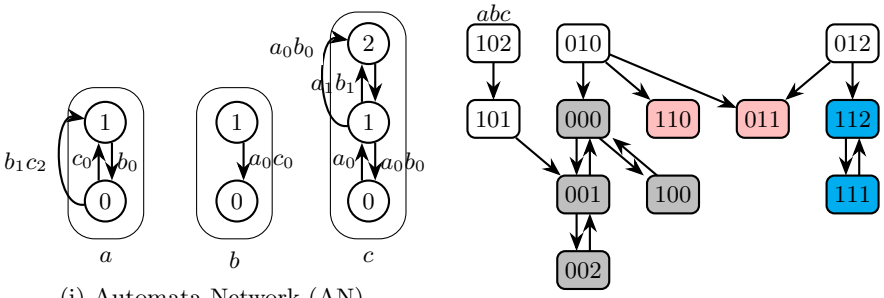
[5] Kobe University, 1-1, Rokko-dai, Nada, Kobe, Hyogo 657-8501 Japan tamura@kobe-u.ac.jp

Abstract. In this paper, we present a SAT-based Attractor Finder (SAF) which computes attractors in biological regulatory networks modelled as asynchronous automata networks. SAF is based on translating the problem of finding attractors of a bounded size into a satisfiability problem to take advantage of state-of-the-art SAT encodings and solvers. SAF accepts an automata network and outputs attractors in ascending size order until the bound is reached. SAF's main contribution is providing an alternative to existing attractor finders. There are cases where it is able to find some attractors while other techniques fail to do so. We observed such capability on both automata networks and Boolean networks. SAF is simple to use: it is available as a command line tool as well as a web application. Finally, SAF being written in Scala, it can run on any operating system with a Java virtual machine when combined with the SAT solver Sat4j.

Keywords: Automata Networks · Attractor · Boolean Networks · SAT Solvers .

1 Introduction

Asynchronous automata networks (hereafter abridged as automata networks or AN) [12,16] are a multi-valued mathematical model widely studied for the qualitative analysis of the dynamical behavior of biological regulatory networks. Figure 1 (i) is an example of automata network which contains three automata a, b and c whose possible state values are $a, b \in \{0, 1\}$, $c \in \{0, 1, 2\}$. The arcs



(i) Automata Network (AN) (ii) State Transition Graph (STG)

Fig. 1. Example of AN and its corresponding STG in asynchronous semantics

and their labels denote transitions that change the state value of an automaton with its labeled condition, e.g., a transition will change the state value of the automaton a from 0 to 1 with the condition $b = 1$ and $c = 2$. Among dynamical properties, *Attractors* raise a specific interest by their capacity to help validate the design of a biological model and predict possible asymptotic behaviors (e.g., capturing differentiation processes). *Attractors* can be defined from the state transition graph built from the automata network. Formally, a *trap domain* is a set of states that do not have outgoing arcs, meaning they cannot be escaped and thus loop indefinitely. An attractor is a subset minimal trap domain. Figure 1 (ii) represents the state transition graph of the automata network (i). It contains four attractors of sizes $1, 1, 2$, and 4, corresponding to the colored states. However, few software tools are available to the general public despite the importance of attractors in automata networks. Pint [17] is a popular tool, but it cannot compute complex attractors. A method based on Answer Set Programming (ASP) presented in the paper [6,7] can compute complex attractors, but sometimes fails to do so for large models.

To fill these gaps, we propose the software SAF using a SAT-based method for finding attractors of bounded size in asynchronous automata networks. SAF has been developed based on a recently proposed SAT-based method [19]. The existing ASP-based method [7] encodes attractors as a not-simple cycle between states, while our SAT encoding models attractors as a set of states. The complexity of our encoding is the attractor's size, while the ASP encoding's complexity lies in the cycle's length. In our running example, the attractors of size four will be computed as $\{000, 001, 002, 100\}$ by the SAT encoding while it will be computed as $(100, 000, 001, 002, 001, 000, 100)$ by the ASP encoding. SAF benefits from the following characteristics: **Efficiency.** SAF can take advantage of the wide availability of state-of-the-art SAT solvers and SAT enumerators, have received considerable attention and saw impressive progress during the last two decades [10]. The efficiency of such an approach is comparable and sometimes better than existing ASP-based methods [19]. **Portability.** The current version of SAF adopts Sat4j [15] as default back-end SAT solver. The combination of SAF and Sat4j provides a portable software tool that runs on any platform sup-

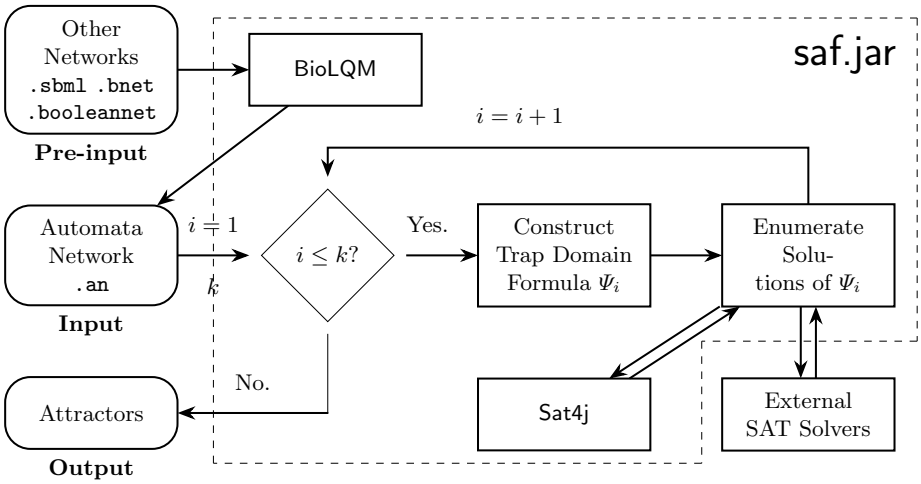


Fig. 2. SAF Architecture. The parts enclosed in dashed lines are included in saf.jar.

porting Java. **Easy-to-use.** SAF is available as a command line tool as well as a web application https://saf-app.herokuapp.com/. Those tools using Sat4j can be used without installing any other dependency.

2 Architecture and Implementation

The architecture of SAF is summarized in Fig. 2. This section explains in detail its input, output, and the design of the core-engine.

I/O. SAF takes as input an automata network, but also supports other formats, e.g., `.sbml`, `.bnet`, `.booleannet`, which will be translated into automata networks using BioLQM [11]. In addition, a bound k can be given, limiting the computation to attractors of size at most or equal to k. The output is a set of attractors of increasing size, printed to the standard output as soon as the calculation is completed.

Core-engine. The computation of attractors of a bounded size is delegated to a SAT solver, i.e., we build a formula Ψ_k such that each model of Ψ_k is a trap domain of size less than or equal to k. After finding a trap domain, we modify the formula Ψ_k to prevent finding it again. The procedure enumerates that way all trap domains of size k. Applying the procedure with an increasing value of k, it is possible to guarantee that all trap domains found are minimal and thus are also attractors of size bounded by k. The main challenges in SAT-based approaches are minimizing encoded SAT problems and enumerating numerous attractors. We address the former by employing the set-based encoding discussed above and tackle the latter by utilizing low-level integration with SAT solvers through the incremental IPASIR interface [1] and dedicated model enumerators for large attractor counts.

Table 1. Comparisons between SAF and ASP-based Method

	Statictics					Existing	SAF	
Instance	$\|\Sigma\|$	$\|T\|$	Attractors	k	$\|S_{\max}\|$	ASP [4]	Sat4j	Ext
Example	4	12	1(3):2(1):4(1)	4	3	**0.2**	1.7	1.7
Lambda phage	4	46	1(1):2(1)	2	4	**0.1**	1.8	1.9
Trp-reg	4	14	1(2):4(1)	4	3	**0.2**	1.7	1.8
Fission-yeast	9	43	1(1)	1	3	**0.0**	1.5	1.5
Mamm.	10	34	1(1)	1	2	**0.0**	1.3	1.3
Tcrsig	40	85	1(8)	1	2	**0.0**	1.8	1.7
FGF	59	102	1(1536)	1	3	**0.0**	4.2	2.0
T-helper (AN)	101	316	1(5875504)	1	3	148.3	T.O.	**32.9**
star05	5	10	6(1)	6	2	**1.6**	1.8	1.9
star10	10	20	11(1)	11	2	148.0	5.3	**4.0**
star12	12	24	13(1)	13	2	9959.3	7.8	**4.7**
star15	15	30	16(1)	16	2	T.O.	26.2	**8.8**
star20	20	40	21(1)	21	2	T.O.	117.0	**41.0**
star30	30	60	31(1)	31	2	T.O.	8252.9	**878.8**
star40	40	80	41(1)	41	2	T.O.	T.O.	**9329.9**
star50	50	100	51(1)	51	2	T.O.	T.O.	T.O.

Implementation. SAF is implemented in Scala. It can work on any system with a Java Virtual Machine (JVM) version 8 or greater using the Java SAT solver Sat4j [15]. For better performances, we also provide one state-of-the-art SAT solver, CaDiCaL [9] and SAT enumerator, BDD_Minisat_all [21], which need to be compiled on the host system. In the current implementation, by explicitly adding command line options, BDD_Minisat_all is called when $k = 1$ and CaDiCaL or Sat4j are called when $k \geq 2$.

3 Performance

We carried out experimental comparisons against the state-of-the-art tools to check the performance of SAF. We executed all experiments on a computer with a 3GHz CPU and 16GB of RAM. All benchmark instances and execution logs are available in https://doi.org/10.5281/zenodo.8062836.

Automata Network. We compared our system with the ASP-based tool presented in the literature [7]. Both use bounded search and are systems requiring an upper bound k. Benchmark instances are all instances from the literature [7,19], which consists in 8 biologically inspired networks, and artificial star networks that have increasing size attractors proposed in [19]. Those attractors have cycle sizes that are twice as large as their number of states (for detail, see Section 6.1 of [19]). The time limit is 3 hours. Table 1 shows the result. From left to right, columns denote the name of the automata networks, the number of automata included, the number of transitions included, the sizes of the attractors, the bound k given to both solvers, the maximum number of states of automata denoted as $S_{\max}$, the CPU time of the compared ASP-based method, SAF using Sat4j and

Table 2. Comparisons between SAF and Boolean network tools

Instance	#Var	Boolsim [13]	Cabean [20]	Pyboolnet [14]	Pystable. [18]	Aeon [8]	SAF Ext
T-Cell 2006	40	**7/1**	0/0	**7/1**	**7/1**	**7/1**	7/0
ABA	44	**16/12**	**16/12**	**16/12**	**16/12**	**16/12**	16/3
T-LGL	61	0/0	0/0	0/0	172/88[6]	**172/142**	172/92
EMT	69	**13/0**	**13/0**	0/0	**13/0**	**13/0**	**13/0**
T-Cell 2007	101	0/0	0/0	0/0	**104/24**	**104/24**	104/0
T-helper (BN)	103	0/0	0/0	0/0	0/0	0/0	**5875504/0**

external SAT solvers denoted as Ext. In the biologically inspired benchmarks, all methods solved them within a few seconds. The main reason is that they contain small-sized attractors and thus k is also small. One other difference between the ASP-based method and SAF comes from the implementation language: C++ and Scala. SAF is implemented on Scala which is running on JVM for the modeling part and thus there is a small disadvantage on this point. One exception is `T-helper` which contains a large number of attractors. On this problem, our approach takes advantage of the SAT solver `BDD_MINISAT_ALL` dedicated to the fast enumeration of solutions. On `star` benchmarks, the difference between each encoding is more obvious. Our set-based encoding outperforms ASP cycle-based encoding. The CPU time of the ASP-based method increases exponentially, and it cannot solve `star15` within 3 hours. On the other hand, although SAF+Sat4j cannot solve `star30`, SAF+external SAT solvers successfully solved them until `star40`. In summary, while the ASP-based method is faster at finding small-sized attractors, it may fail to detect some; on the other hand, SAF is more robust at finding attractors, but it faces difficulties when their size exceeds 50.

Boolean Network. We also compared our system with state-of-the-art Boolean network tools: Boolsim [13], Cabean [20], Pyboolnet [14], Pystablemotifs [18] and Aeon [8]. All compared tools execute comprehensive searches, i.e., those tools run without specifying k. So, we run SAF with large enough k and compare the number of attractors computed within the time limit 2,400 seconds. We employ existing Boolean network instances to facilitate the comparison since no tool is currently available for converting automata networks into Boolean networks. Benchmark instances are all real cell model instances from the literature [18]. The specific instance T-LGL may vary from 58 to 61 variables depending on the source. Our instance is available from our dataset [3]. In addition, we compare one more instance, a Boolean Network version of `T-helper` [5] downloaded from URL [2], which has a large number of attractors. Table 2 shows the result[6]. From left to right, columns denote the name of the Boolean networks, the number of atoms included, the number of attractors computed by compared tools and SAF in the form of "n/m" where n denotes the number of singleton attractors and m denotes the number of complex attractors. Unlike automata-network results, SAF did not yield conclusive results on those benchmarks. The reasons for this

[6] In our environment, Pystablemotifs version 3.3 throws an error to T-LGL on our computer. Thus, we provide the number of attractors found in their paper [18]

could be twofold: i) SAF converts Boolean networks into richer automata networks instead of directly solving them, and ii) it struggles with calculating large attractors exceeding 50 due to its process of exploring attractors of increasing size. However, Boolean network version of `T-helper`, which contains many attractors, could not be enumerated by any other tool within the time limit.

4 Availability

Command line tool. The command line tool, sources, and instructions are available on https://github.com/TakehideSoh/SAF. The command line tool can be executed from users' terminal as follows.

```
$ java -jar saf.jar [options] [inputFile]
```

The minimum example using Sat4j and $k = 4$ on the automata network file `runningexample.an` is as follows.

```
$ java -jar saf.jar -k 4 runningexample.an
```

To run the above command, users do not need any installation on any platform supporting Java 8 or greater. Improved performance is available by adding options of BDD_Minisat_all and CaDiCaL (or any IPASIR compatible software). In this case, the installation of those SAT solvers in the system is necessary.
Web application. The web application version of SAF (see Fig. 3 in Appendix) is available at https://saf-app.herokuapp.com/. It currently accepts multivalued networks (`.an .sbml`), and Boolean networks (`.bnet .booleannet`). To use this application, users initially edit their network in a textbox or upload local files. Next, they specify a bound k and push "find attractors". The result is printed into another textbox. This service is hosted on HEROKU, one of the PaaS providers. Due to service limitations, each execution is limited to 20 seconds, but it has sufficient performance for simple use cases.

5 Conclusion

This paper presents a SAT-based Attractor Finder (SAF) for identifying attractors of biological regulatory networks modeled as dynamical multi-level discrete models. SAF was developed with the goal of being an efficient, portable, and easy-to-use tool. The core of the approach targets automata networks but, thanks to existing translators between modeling frameworks, SAF handles models specified in various Boolean network formats (Boolnet, Booleannet) and in SBML-qual. Source code, executable as a Jar file and a web application are available to make it easily usable by a wide range of users, from computer scientists to modelers. Future work includes making the output of SAF compatible with JSON format and graphical representations to make it more understandable for beginners. It is also important to apply it to challenging instances of Boolean Networks. In addition, extending SAF to other semantics and considering other attractor-related problems like bifurcation is interesting.

Acknowledgements This work was financially supported by the "PHC Sakura" program (43009SC, JPJSBP120193213), implemented by MESRI and JSPS. This work was also supported by JSPS KAKENHI (JP21K11828, JP22K11973, 23K11047), and by ROIS NII Open Collaborative Research 2023 (23FP04).

References

1. IPASIR. https://www.cs.utexas.edu/users/moore/acl2/manuals/current/manual/index-seo.php/IPASIR____IPASIR
2. Model of Boolean network version of T-helper. https://github.com/hklarner/pyboolnet/tree/master/pyboolnet/repository/jaoude_thdiff
3. Model of T-LGL used in Boolean netrwork experiments. https://github.com/TakehideSoh/SAF-Evaluation/blob/main/exp-boolean-network/benchmark/2176_T-LGL_Survival_Network_2008.booleannet.bnet
4. Abdallah, E.B., Folschette, M., Roux, O.F., Magnin, M.: ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. Algorithms for Molecular Biology **12**(1), 20:1–20:23 (2017)
5. Abou-Jaoudé, W., Monteiro, P.T., Naldi, A., Grandclaudon, M., Soumelis, V., Chaouiya, C., Thieffry, D.: Model checking to assess t-helper cell plasticity. Frontiers in bioengineering and biotechnology **2**, 86 (2015)
6. Ben Abdallah, E., Folschette, M., Magnin, M.: Analyzing long-term dynamics of biological networks with answer set programming. Systems Biology Modelling and Analysis: Formal Bioinformatics Methods and Tools pp. 251–303 (2022)
7. Ben Abdallah, E., Folschette, M., Roux, O., Magnin, M.: Asp-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. Algorithms for Molecular Biology **12**(1), 1–23 (2017)
8. Benes, N., Brim, L., Huvar, O., Pastva, S., Safránek, D., Smijáková, E.: Aeon.py: Python library for attractor analysis in asynchronous boolean networks. Bioinform. **38**(21), 4978–4980 (2022)
9. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021)
11. Chaouiya, C., Bérenguier, D., Keating, S.M., Naldi, A., van Iersel, M.P., Rodriguez, N., Dräger, A., Büchel, F., Cokelaer, T., Kowal, B.M., Wicks, B., Gonçalves, E.J.V., Dorier, J., Page, M., Monteiro, P.T., von Kamp, A., Xenarios, I., de Jong, H., Hucka, M., Klamt, S., Thieffry, D., Novère, N.L., Saez-Rodriguez, J., Helikar, T.: SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. BMC Syst. Biol. **7**, 135 (2013)
12. Folschette, M., Paulevé, L., Magnin, M., Roux, O.: Sufficient conditions for reachability in automata networks with priorities. Theoretical Computer Science **608**, 66–83 (2015)
13. Garg, A., Cara, A.D., Xenarios, I., Mendoza, L., Micheli, G.D.: Synchronous versus asynchronous modeling of gene regulatory networks. Bioinform. **24**(17), 1917–1925 (2008)

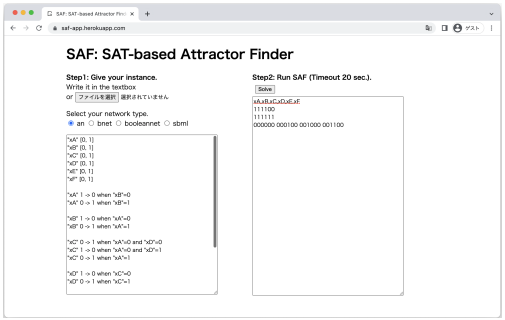
14. Klarner, H., Streck, A., Siebert, H.: Pyboolnet: a python package for the generation, analysis and visualization of boolean networks. Bioinform. **33**(5), 770–772 (2017)
15. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation **7**(2-3), 59–64 (2010)
16. Paulevé, L.: Goal-oriented reduction of automata networks. In: International Conference on Computational Methods in Systems Biology. Lecture Notes in Bioinformatics, vol. 9859, pp. 252–272. Springer (2016)
17. Paulevé, L.: Pint: A Static Analyzer for Transient Dynamics of Qualitative Networks with IPython Interface. In: CMSB 2017 - 15th conference on Computational Methods for Systems Biology. Lecture Notes in Computer Science, vol. 10545, pp. 370 – 316. Springer (2017)
18. Rozum, J.C., Deritei, D., Park, K.H., Zañudo, J.G.T., Albert, R.: pystablemotifs: Python library for attractor identification and control in boolean networks. Bioinform. **38**(5), 1465–1466 (2022)
19. Soh, T., Magnin, M., Berre, D.L., Banbara, M., Tamura, N.: Sat-based method for finding attractors in asynchronous multi-valued networks. In: Ali, H., Deng, N., Fred, A.L.N., Gamboa, H. (eds.) Proceedings of the 16th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2023, Volume 3: BIOINFORMATICS. pp. 163–174. SCITEPRESS (2023)
20. Su, C., Pang, J.: CABEAN: a software for the control of asynchronous boolean networks. Bioinform. **37**(6), 879–881 (2021)
21. Toda, T., Soh, T.: Implementing efficient all solutions SAT solvers. ACM J. Exp. Algorithmics **21**(1), 1.12:1–1.12:44 (2016)

A Input File and Web Interface of SAF

The automata network from Fig. 1 can be described by the following automata network `runningexample.an` input file as in Figure 3. The web application version of SAF is demonstrated in Figure 3 as well.

```
"a" [0, 1]
"b" [0, 1]
"c" [0, 1, 2]
"a" 0 -> 1 when "b"=1 and "c"=2
"a" 0 -> 1 when "c"=0
"a" 1 -> 0 when "b"=0
"b" 1 -> 0 when "a"=0 and "c"=0
"c" 0 -> 1 when "a"=0
"c" 1 -> 2 when "a"=0 and "b"=0
"c" 1 -> 2 when "a"=1 and "b"=1
"c" 2 -> 1
"c" 1 -> 0 when "a"=0 and "b"=0
```

`runningexample.an`



https://saf-app.herokuapp.com/

Fig. 3. Input File and Web Interface of SAF