



DFAMiner: An efficient tool for learning minimal separating DFAs from labelled samples

Daniele Dell'Erba^a, Yong Li^{b,*}, Sven Schewe^c, Andrea Turrini^{b,d}

^a Department of Computer Science, Middlesex University, London, United Kingdom

^b Key Laboratory of System Software (Chinese Academy of Sciences), Institute of Software Chinese Academy of Sciences, Beijing, China

^c Department of Computer Science, University of Liverpool, Liverpool, United Kingdom

^d Institute of Intelligent Software Guangzhou, Guangzhou, China

ARTICLE INFO

Keywords:

Separating DFAs
Three-valued automata
SAT
Learning

ABSTRACT

We introduce DFAMiner, an efficient tool for learning minimal separating deterministic finite automata (DFA) from a set of labelled samples. The significant improvement of DFAMiner over existing tools is the use of an intermediate representation called *three-valued* automaton for the given set of labelled samples. This three-valued automaton has accepting and rejecting states as well as don't-care states, so that it can exactly recognise the labelled samples. The minimal separating DFA for the labelled samples is then learned by minimising the constructed three-valued automata via a reduction to SAT solving. Separating automata are an interesting class of automata that occurs generally in regular model checking and has raised interest in foundational questions of parity game solving. Therefore, DFAMiner has the potential to further advance these fields.

1. Introduction

The task of inferring a minimum-size separating automaton from two disjoint sets of samples has gained much attention from various fields, including computational biology [1], inference of network invariants [2], regular model checking [3], and reinforcement learning [4]. More recently, this problem has also arisen in the context of parity game solving [5], where separating automata can be used to decide the winner. These applications can be formalised as seeking the minimum-sized DFAs, known as the Min-DFA inference problem, from positive and negative samples. In passive learning, all samples not seen can take any value.

Finding the minimal DFA can be done in two steps. We can first learn a concise representation of an almost deterministic finite automaton with *three* values: *accept*, *reject*, and *don't care*. That is, we first represent them as a 3-valued almost deterministic finite automaton (3ADFA), and finally reduce the problem of finding a minimal DFA that agrees with the 3ADFA outside of don't care entries to a Boolean Satisfiability (SAT) problem.

While we have developed our tool around passive learning, it is also possible to obtain a 3ADFA directly from specifications. In regular model checking, for example, we can make all runs that cannot be created by an abstraction (or that violate assumptions in assume-guarantee reasoning) as *don't care*. In the case of parity games, words that contain both winning and losing cycles, as well as short words, can be marked as *don't care*.

* Corresponding author.

E-mail addresses: d.dellerba@mdx.ac.uk (D. Dell'Erba), liyong@ios.ac.cn (Y. Li), Sven.Schewe@liverpool.ac.uk (S. Schewe), turrini@ios.ac.cn (A. Turrini).

<https://doi.org/10.1016/j.scico.2026.103506>

Received 3 October 2025; Received in revised form 13 March 2026; Accepted 3 May 2026

Available online 12 May 2026

0167-6423/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Related work. The Min-DFA inference problem was first explored in [6,7]. Due to its high complexity (NP-complete), researchers initially focused on either finding local optima through state merging techniques [8–10], or investigating theoretical aspects such as reduction to graph colouring problems [11]. Notably, it has been shown that there is no efficient algorithm to find approximate solutions [12].

With the increase in computational power and efficiency of SAT solvers, research has shifted towards practical and *exact* solutions to the Min-DFA inference problem. Several tools have emerged in the literature, including ed-beam/exbar [13], FlexFringe [14], DFA-Inductor [15,16], and DFA-Identify [4].

The learned Min-DFA can be seen as a witness proof that separates the sets of good and bad behaviours for a given system. Therefore, our work can be directly applied to the problems that look for those proofs, such as regular model checking [3] and reinforcement learning [4]. Another standard application is in the active learning of minimal DFAs using equivalence queries [17]. We remark that non-incremental and incremental constructions were proposed in [18] to find small and even minimal APTAs that separate the positive and negative samples. These two constructions are based on regular positive negative inference (RPNI) state merging techniques [9]. Their algorithms are approximate constructions. As a consequence, their constructed APTAs cannot be used to extract a minimal separating DFA.

2. Problem description

Before introducing our problem, we first give some basic notation and terminologies. We fix an alphabet Σ in the paper. A word w is a sequence of letters in Σ ; in particular, ϵ is the empty word. A regular language is a set of words that can be recognised by a DFA. A DFA D is a tuple (Q, ι, δ, A) over an alphabet Σ where Q is a set of states, ι is the initial state, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function and A is the set of accepting states. We also extend δ to words by letting $\delta(q, \epsilon) = q$ and $\delta(q, au) = \delta(\delta(q, a), u)$ for any $q \in Q$, $a \in \Sigma$ and $u \in \Sigma^*$. Hence, D can be seen as a function such that $D(u) = +$ if $\delta(\iota, u) \in A$ and $D(u) = -$ otherwise.

Let $S = (S^+, S^-)$ be the given set of labelled samples in the whole paper where S^+ is the set of *positive* samples that must be accepted while S^- contains all samples that need to be rejected. Naturally, we can regard S as a function such that, for a word $u \in \Sigma^*$, we have $S(u) = \$$ if $u \in S^\$$ where $\$ \in \{+, -\}$ and $S(u) = ?$ if $u \notin (S^+ \cup S^-)$. Our problem here is to construct a *separating* DFA having the minimal number of states for a given set of labelled samples. We say a DFA D is *separating* (consistent with) the samples S if, for all $u \in \Sigma^*$, if $S(u) = \$$, then $D(u) = \$$, where $\$ \in \{+, -\}$. Our goal is to find a *minimal* separating DFA (Min-DFA) D for S , i.e., a separating DFA having the minimal number of states.

To address the separating Min-DFA problem, DFAMiner makes use of succinct intermediate representations for S . First, DFAMiner uses a 3-valued DFA (3DFA) S with a minimum number of states to represent S , where S classifies the words in Σ^* into three different types such that, for all $u \in \Sigma^*$, $S(u) = S(u)$ [19,20]. To this end, the 3DFA S is defined as a tuple (Q, ι, δ, A, R) where (Q, ι, δ, A) is the same as a DFA, R is the set of *rejecting* states (with $A \cap R = \emptyset$) and thus $D = Q \setminus (A \cup R)$ is the set of remaining *don't-care* states. DFAMiner also uses another representation of S called 3-valued almost deterministic finite automaton (3ADFA)¹: a tuple $(Q, \iota_a, \iota_r, \delta, A, R)$ where $(Q, \iota_a, \delta, A, \emptyset)$ is a 3DFA separating $(S^+, \emptyset)$ and $(Q, \iota_r, \delta, \emptyset, R)$ is a 3DFA separating $(\emptyset, S^-)$. 3ADFAs are more general than 3DFAs. In the following, 3ADFAs refer both to 3ADFAs and 3DFAs unless specified otherwise.

3. Tool architecture

DFAMiner is written in Python and available publicly at <https://github.com/liyong31/DFAMiner> under MIT license; the main code metadata is summarised in Table 1. Its architecture, shown in Fig. 1, comprises two components: the 3ADFA constructor, which builds a 3ADFA to recognise the sample set S and the 3ADFA minimiser, which uses a SAT solver to obtain from the 3ADFA a minimal separating DFA for S .

3.1. DFAMiner core

3ADFA constructor. The *Constructor* implements the algorithms described in [20] to construct a 3ADFA or a 3DFA S for the set S of samples. If S is a 3DFA, it is further guaranteed that S has the minimal number of states among all possible 3DFAs recognising S [20].

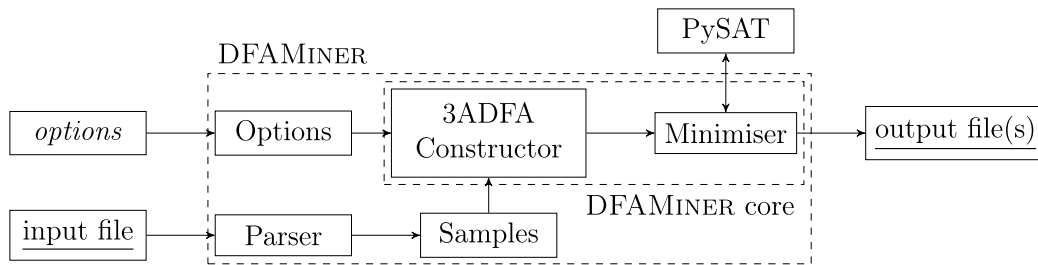
3ADFA minimiser. The *Minimiser* encodes the problem of finding a separating Min-DFA from a 3ADFA as a SAT problem. More precisely, given an integer $k > 0$, and a 3ADFA S , *Minimiser* would guess a DFA D of k states by using Boolean variables to encode the existence of the transitions and whether a state is accepting or not. Furthermore, to enforce a consistent DFA of S , *Minimiser* also adds the corresponding clauses as constraints. It is guaranteed that there is a separating DFA D with k states for S if, and only if, the SAT problem is satisfiable. To ensure minimality, *Minimiser* starts by setting $k = 1$ and then increases k until the SAT problem is satisfiable. So, the first satisfiable DFA D of k states would be a minimal DFA we look for. We refer to [19,20] for the details.

DFAMiner delegates all SAT queries to the Python library PySAT [21].

Table 1

Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	https://github.com/liyong31/DFAMiner
C3	Permanent link to Reproducible Capsule	https://doi.org/10.5281/zenodo.17204729
C4	Legal Code License	MIT
C5	Code versioning system used	Git and GitHub
C6	Software code languages, tools, and services used	Python, PySAT
C7	Compilation requirements, operating environments and dependencies	Python 3.7 +
C8	If available, link to developer documentation/manual	https://github.com/liyong31/DFAMiner/blob/main/README.md
C9	Support email for questions	turrini@ios.ac.cn

**Fig. 1.** The architecture of DFAMiner.

```

usage: dfaminer.py [-h] --file path --out path [--output-format {dot,
    textual}] [--intermediate path] [--solver {lingeling,mergesat3,
    cadical195,gluecard4,cadical103,maplechrono,minisat22,glucose4,
    glucose42,cadical153}] [--lower LOWER] [--upper UPPER] [--3dfa]
    [--nobfs] [--safety] [--verify]

```

Mining a minimal DFA consistent with samples

options:

```

-h, --help                show this help message and exit
--file path               path to input sample file
--out path                path to output DFA
--output-format {dot,textual}
                           the format for the output
--intermediate path       path to output the intermediate 3DFA
--solver {lingeling,mergesat3,cadical195,gluecard4,cadical103,
    maplechrono,minisat22,glucose4,glucose42,cadical153}
                           choose the SAT solver
--lower LOWER             the lower bound for the DFA
--upper UPPER             the upper bound for the DFA
--3dfa                    use three valued DFA for inference
--nobfs                   disable the constraints for BFS tree
--safety                  construct safety DFA for solving parity games
--verify                  verify resultant DFA

```

Fig. 2. DFAMiner usage.

8 2	
1 3 0 0 0	
1 3 0 0 1	
0 3 0 1 0	
0 3 0 1 1	
1 3 1 0 0	
0 3 1 0 1	
0 3 1 1 0	
0 3 1 1 1	

```

{
  "alphabet": ["0", "1"],
  "accepting": ["000", "001", "100"],
  "rejecting": ["010", "011", "101", "110", "111"]
}

```

Fig. 3. Input file format: Abbadingo (left) and JSON (right).

3.2. DFAMiner usage

DFAMiner is a command line tool whose main file, `dfaminer.py`, offers a number of options to control its behaviour, listed in Fig. 2. Regarding input and output, there are three command options: `file` (mandatory) for the input file with the samples, `out` (mandatory) for the output file where to save the learned DFA, and `intermediate` that allows to save the constructed 3DFA/3ADFA (which depends on the `3dfa` option) before minimisation. Regarding controlling the behaviour of DFAMiner, the option `solver` allows the user to choose which SAT solver to use while mining the minimal DFA. The pair of options `lower` and `upper` allow to set the minimum and maximum number of states the output DFA can have. The default minimum number of states is 1 while the default maximum matches the number of states in the 3DFA built from the input samples. To reduce the search space, *Minimiser* by default uses the technique proposed in [15] to look for only a representative DFA for all isomorphic DFAs differing only in their state names. The idea underlying this approach is that the breath-first search (BFS) tree for each isomorphic DFA is unique. So, we can restrict the BFS tree in the encoding. When using the option `nobfs`, we indicate that this anti-symmetric technique is disabled in the encoding. The `safety` option tries to restrict our search to a separating safety automaton for parity game solving specially described in [19,20]. Lastly, the `verify` options makes `dfaminer.py` verify the correctness of the mined DFA with respect to the input words. That is, it checks whether the output DFA is really separating the given set S of labelled samples.

We now explain in detail the format of the files to be used with the input `file` and the output `out` options.

Input format. DFAMiner accepts two types of input files: an Abbadingo file or a JSON file. The Abbadingo file is a plain text document with a very strict format. The first line of the file contains two numbers: the total number of samples in the file and the number of letters in the alphabet. Each of the following lines in the file encodes one sample as a sequence $m \ n \ l_1 \ l_2 \ \dots \ l_n$, where $m \in \{0, 1\}$ is a Boolean flag indicating whether the sample is positive, n is the length of the sample, and $l_1, \dots, l_n$ are the n letters composing the sample. The JSON file has a more flexible structure: it contains three keys (“alphabet”, “accepting”, and “rejecting”), each one being an array of strings. The “alphabet” strings have length 1 and are thus single letters, and each string in the “accepting” and “rejecting” arrays must be composed of letters of the alphabet. Fig. 3 shows the same samples provided in Abbadingo (on the left) and JSON (on the right) format. The format of the input file is derived from the extension of the input file: if the extension is “json”, then the input is assumed to be a JSON file; otherwise it is assumed to be an Abbadingo file.

Output format. The DFA generated by DFAMiner is saved in the file specified by means of the `out` option; its format is controlled by the `output-format` option, which allows to choose between the default “textual” format à la Abbadingo and the “dot” format of Graphviz. The “textual” format is quite straight-forward and strict: the first line gives the number of states and the size of the alphabet. Each of the following lines is either of the form `i n` (denotes that state n is initial), `a n` (n is accepting), `r n` (n is rejecting), or `t s l d` (there is a transition from state s to state d by letter l). Alternatively, when `output-format` is set to “dot”, `dfaminer.py` generates a file containing a Graphviz’s digraph, which can be automatically visualized by means of Graphviz. For example, Fig. 4 shows the content of the “textual” file created by `dfaminer.py` with the `3dfa` option on the input shown in Fig. 3 as well as the graph drawn by Graphviz from the “dot” file. Fig. 5 shows the analogous output for the final learned minimal separating DFAs.

4. Evaluation

The scatter plots in Fig. 6 show the comparison between the size of the intermediate ADFAs generated by DFAMiner with and without the `3dfa` option and DFA-Inductor [15,16], which uses the same representation of samples as DFA-Identify [4]. As we can see from the plots, DFAMiner outperforms DFA-Inductor by a factor of 3 (the dashed line on the plots), with the 3DFA construction performing slightly better than the 3ADFA one, as indicated by the plot on the right. The DFAs produced as output by the different tools have all matching sizes, since each tool produces the minimal separating DFA for the given input samples.

The scatter plots in Fig. 7 show the comparison of DFAMiner without the `3dfa` option against DFAMiner with it (right plot) as well as with DFA-Inductor (left plot) and DFA-Identify (centre plot). As we can see from the plots, DFAMiner is usually faster than

¹ In [20], it is called double DFA.

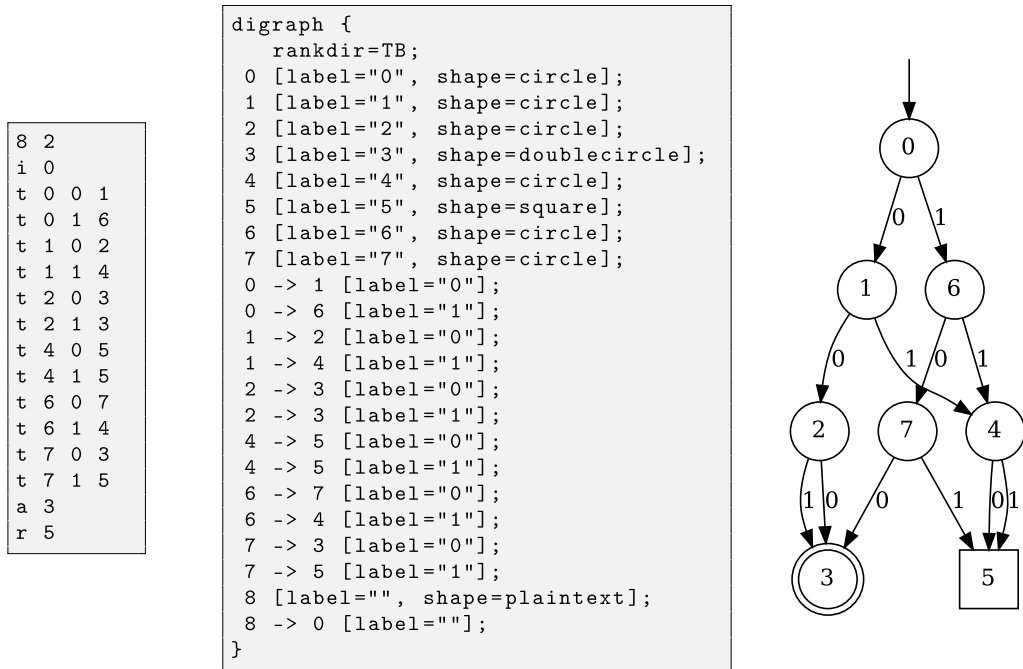


Fig. 4. DFAMiner's intermediate 3DFA as "textual" (left) and "dot" (centre) output format, with its graph version on the right.

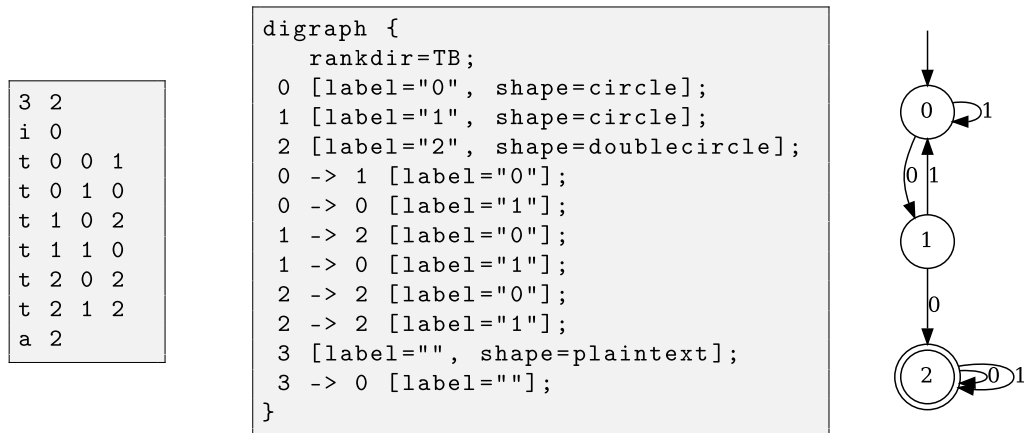


Fig. 5. DFAMiner's learned DFA as "textual" (left) and "dot" (centre) output format, with its graph version on the right.

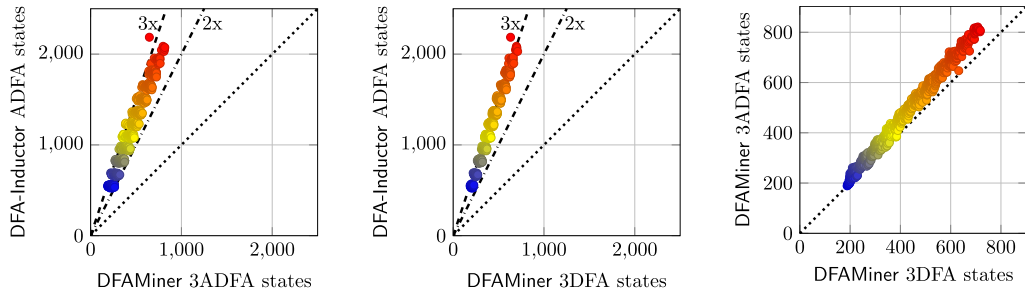


Fig. 6. Size comparison of the intermediate automata.

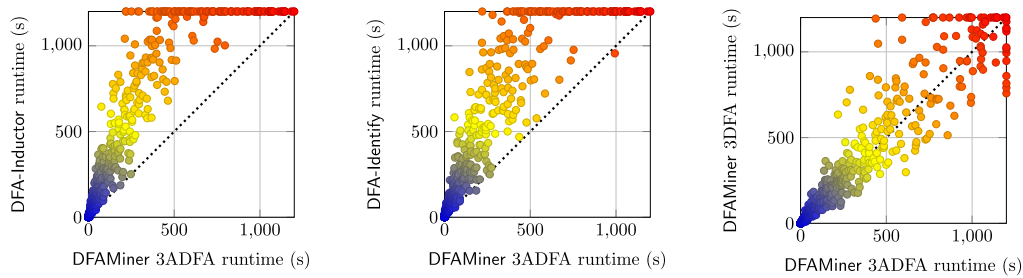


Fig. 7. Running time of DFAMiner against other tools.

both DFA-Inductor (left plot) and DFA-Identify (centre plot), with just few isolated cases where the opposite occurs. While using the 3dfa option usually helps to reduce the size of the intermediate representation of the sample set S (cf. Fig. 6, right plot), there is no clear indication regarding the running time: as shown in the plot on the right of Fig. 7, the marks are rather equally distributed above and below the diagonal.

5. Impact

First of all, according to [19], DFAMiner already proves the lower bound of the separating safety automata used in the parity games that involve parity condition up to 7 colours. Moreover, one can imagine that our results can immediately improve the regular invariants extraction algorithm proposed in [3], as it relies on computing the minimal separating DFAs. The main idea underlying DFAMiner is to reduce the number of Boolean variables needed in the SAT problem by leveraging smaller 3DFA representations of the labelled samples, which naturally can be an inspiration to improve [4]. We believe that DFAMiner can inspire and even be applied to solving many future problems that involve succinct representations of labelled samples.

6. Conclusions and future plans

We have introduced DFAMiner, an efficient tool for learning minimal separating DFA for labelled samples. From the empirical evaluation, we can see that DFAMiner is much faster and more scalable. The main reason behind the efficiency of DFAMiner is that it needs fewer Boolean variables in the SAT problem due to the succinct representation of 3ADFAs for samples. As future work, we plan to investigate whether a more succinct representation of labelled samples than 3ADFAs, e.g., three-valued nondeterministic finite automata, can further accelerate the SAT problem solving.

CRediT authorship contribution statement

Daniele Dell'Erba: Writing – original draft, Methodology, Conceptualization; **Yong Li:** Writing – original draft, Software, Methodology, Conceptualization; **Sven Schewe:** Writing – original draft, Methodology, Conceptualization; **Andrea Turrini:** Writing – original draft, Software, Methodology, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Andrea Turrini reports financial support was provided by European Union. Sven Schewe reports financial support was provided by Engineering and Physical Sciences Research Council. Yong Li reports financial support was provided by Chinese Academy of Sciences. Yong Li reports financial support was provided by Institute of Software Chinese Academy of Sciences. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported in part by the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-040), ISCAS Basic Research (Grant Nos. ISCAS-JCZD-202406, ISCAS-JCZD-202302), ISCAS New Cultivation Project ISCAS-PYFX-202201, the [National Natural Science Foundation of China](#) (Grant No. 62102407), and by the EPSRC through grants EP/X03688X/1 and EP/X042596/1.

This project is part of the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant no. 101008233.

References

- [1] C. de la Higuera, A bibliographical study of grammatical inference, *Pattern Recognit.* 38 (9) (2005) 1332–1348. <https://doi.org/10.1016/J.PATCOG.2005.01.003>
- [2] O. Grinchtein, M. Leucker, N. Piterman, Inferring network invariants automatically, in: U. Furbach, N. Shankar (Eds.), *Automated Reasoning, Third International Joint Conference, IJCAR 2006*, Seattle, WA, USA, August 17–20, 2006, *Proceedings*, 4130 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 483–497. https://doi.org/10.1007/11814771_40
- [3] D. Neider, Computing minimal separating DFAs and regular invariants using SAT and SMT solvers, in: S. Chakraborty, M. Mukund (Eds.), *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012*, Thiruvananthapuram, India, October 3–6, 2012, *Proceedings*, 7561 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 354–369. https://doi.org/10.1007/978-3-642-33386-6_28
- [4] N. Lauffer, B. Yalcinkaya, M. Vazquez-Chanlatte, A. Shah, S.A. Seshia, Learning deterministic finite automata decompositions from examples and demonstrations, in: A. Griggio, N. Rungta (Eds.), *22nd Formal Methods in Computer-Aided Design, FMCAD 2022*, Trento, Italy, October 17–21, 2022, *IEEE*, 2022, pp. 1–6. https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_39
- [5] M. Bojańczyk, W. Czerwiński, *An Automata Toolbox*, Unpublished, 2018.
- [6] A.W. Biermann, J.A. Feldman, On the synthesis of finite-state machines from samples of their behavior, *IEEE Trans. Comput.* 21 (6) (1972) 592–597. <https://doi.org/10.1109/TC.1972.5009015>
- [7] E.M. Gold, Complexity of automaton identification from given data, *Inf. Control.* 37 (3) (1978) 302–320. [https://doi.org/10.1016/S0019-9958\(78\)90562-4](https://doi.org/10.1016/S0019-9958(78)90562-4)
- [8] K.J. Lang, B.A. Pearlmutter, R.A. Price, Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm, in: V.G. Honavar, G. Slutzki (Eds.), *Grammatical Inference, 4th International Colloquium, ICGI-98*, Ames, Iowa, USA, July 12–14, 1998, *Proceedings*, 1433 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 1–12. <https://doi.org/10.1007/BFB0054059>
- [9] J. Oncina, P. Garcia, Inferring regular languages in polynomial updated time, in: *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*, World Scientific, 1992, pp. 49–61.
- [10] M.M.F. Bugalho, A.L. Oliveira, Inference of regular languages using state merging algorithms with search, *Pattern Recognit.* 38 (9) (2005) 1457–1467. <https://doi.org/10.1016/J.PATCOG.2004.03.027>
- [11] F. Coste, J. Nicolas, Regular inference as a graph coloring problem, in: IWGI, 1997.
- [12] L. Pitt, M.K. Warmuth, The minimum consistent DFA problem cannot be approximated within any polynomial, *J. ACM* 40 (1) (1993) 95–142. <https://doi.org/10.1145/138027.138042>
- [13] M. Heule, S. Verwer, Exact DFA identification using SAT solvers, in: J.M. Sempere, P. García (Eds.), *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010*, Valencia, Spain, September 13–16, 2010, *Proceedings*, 6339 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 66–79. https://doi.org/10.1007/978-3-642-15488-1_7
- [14] S. Verwer, C.A. Hammerschmidt, Flexfringe: a passive automaton learning package, in: *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, Shanghai, China, September 17–22, 2017, *IEEE Computer Society*, 2017, pp. 638–642. <https://doi.org/10.1109/ICSME.2017.58>
- [15] V. Ulyantsev, I. Zakirzyanov, A. Shalyto, BFS-based Symmetry breaking predicates for DFA identification, in: A. Dediu, E. Formenti, C. Martín-Vide, B. Truthe (Eds.), *Language and Automata Theory and Applications - 9th International Conference, LATA 2015*, Nice, France, March 2–6, 2015, *Proceedings*, 8977 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 611–622. https://doi.org/10.1007/978-3-319-15579-1_48
- [16] I. Zakirzyanov, A. Morgado, A. Ignatiev, V. Ulyantsev, J. Marques-Silva, Efficient symmetry breaking for SAT-based minimum DFA inference, in: C. Martín-Vide, A. Okhotin, D. Shapira (Eds.), *Language and Automata Theory and Applications - 13th International Conference, LATA 2019*, St. Petersburg, Russia, March 26–29, 2019, *Proceedings*, 11417 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 159–173. https://doi.org/10.1007/978-3-030-13435-8_12
- [17] D. Angluin, Negative results for equivalence queries, *Mach. Learn.* 5 (1990) 121–150. <https://doi.org/10.1007/BF00116034>
- [18] R. Alquezar, A. Sanfeliu, Incremental grammatical inference from positive and negative data using unbiased finite state automata, in: *Shape, Structure and Pattern Recognition, Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR*, 94, 1995, pp. 291–300.
- [19] D. Dell'Erba, Y. Li, S. Schewe, DFAMiner: mining minimal separating DFAs from labelled samples, 2024, [arXiv:2405.18871](https://arxiv.org/abs/2405.18871)
- [20] D. Dell'Erba, Y. Li, S. Schewe, DFAMiner: mining minimal separating DFAs from labelled samples, in: A. Platzer, K.Y. Rozier, M. Pradella, M. Rossi (Eds.), *Formal Methods - 26th International Symposium, FM 2024*, Milan, Italy, September 9–13, 2024, *Proceedings, Part II*, 14934 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 48–66. https://doi.org/10.1007/978-3-031-71177-0_4
- [21] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: a python toolkit for prototyping with SAT oracles, in: *SAT*, 2018, pp. 428–437. https://doi.org/10.1007/978-3-319-94144-8_26