DFAMiner: Mining minimal separating DFAs from labelled samples

Daniele Dell'Erba   dde@liverpool.ac.uk
Yong Li   liyong@liverpool.ac.uk
Sven Schewe   svens@liverpool.ac.uk
University of Liverpool, UK

arXiv:2405.18871v1 [cs.FL] 29 May 2024

Abstract

We propose DFAMiner, a passive learning tool for learning minimal separating deterministic finite automata (DFA) from a set of labelled samples. 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. We first propose a simple and linear-time algorithm that incrementally constructs a three-valued DFA (3DFA) from a set of labelled samples given in the usual lexicographical order. This 3DFA has accepting and rejecting states as well as don't-care states, so that it can exactly recognise the labelled examples. We then apply our tool to mining a minimal separating DFA for the labelled samples by minimising the constructed automata via a reduction to solving SAT problems. Empirical evaluation shows that our tool outperforms current state-of-the-art tools significantly on standard benchmarks for learning minimal separating DFAs from samples. Progress in the efficient construction of separating DFAs can also lead to finding the lower bound of parity game solving, where we show that DFAMiner can create optimal separating automata for simple languages with up to 7 colours. Future improvements might offer inroads to better data structures.

Keywords: Separating DFAs, Passive Learning, Three-valued DFAs, Parity Game Solving

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 (de la Higuera, 2005), inference of network invariants (Grinchtein et al., 2006), regular model checking (Neider, 2012), and reinforcement learning (Lauffer et al., 2022). More recently, this problem has also arisen in the context of parity game solving (Bojańczyk and Czerwiński, 2018), where separating automata can be used to decide the winner. The breakthrough quasi-polynomial algorithm (Calude et al., 2017), for example, can be viewed as producing such a separating automaton, and under additional constraints, quasi-polynomial lower bounds can be established, too (Czerwinski et al., 2019; Calude et al., 2017). These applications can be formalised as seeking the minimum-size of DFAs, known as the Min-DFA inference problem, from positive and negative samples.

The Min-DFA inference problem was first explored in (Biermann and Feldman, 1972; Gold, 1978). Due to its high (NP-complete) complexity, researchers initially focused on either finding local optima through state merging techniques (Lang et al., 1998; Oncina and Garcia, 1992; Bugalho and Oliveira, 2005), or seeking theoretical aspects such as reduction

to graph colouring problems (Coste and Nicolas, 1997). Notably, it has been shown that there is no efficient algorithm to find approximate solutions (Pitt and Warmuth, 1993).

With the increase in computational power and efficiency of Boolean Satisfiability (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 (Heule and Verwer, 2010), FlexFringe (Verwer and Hammerschmidt, 2017), DFA-Inductor (Ulyantsev et al., 2015; Zakirzyanov et al., 2019) and DFA-Identify (Lauffer et al., 2022).

The current practical and exact solutions to the Min-DFA inference problem typically involve two steps: (1) Construct the augmented prefix tree acceptor (APTA (Coste and Nicolas, 1998)) that recognises the given samples, and (2) minimise the APTA to a Min-DFA by a reduction to SAT (Heule and Verwer, 2010). Recent enhancements of this approach focus on the second step, including techniques like symmetry breaking (Heule and Verwer, 2010; Ulyantsev et al., 2015) and compact SAT encoding (Heule and Verwer, 2010; Zakirzyanov et al., 2019). Additionally, there is an approach on the incremental SAT solving technique specialised for the Min-DFA inference problem, where heuristics for assigning free variables have also been proposed (Avellaneda and Petrenko, 2019). However, their implementation relies heavily on MiniSAT (Eén and Sörensson, 2003). We believe that, in order to take advantage of future improvements of SAT solvers, it is better to use a SAT solver as a black-box tool. We note that the second step has also been encoded as a Satisfiability Modulo Theories problem (Smetsers et al., 2018), which can also be improved by our contribution to the first step.

The second step is typically the bottleneck in the workflow. It is known that the number of boolean variables used in the SAT problem is polynomial in the number of states of the APTA. Smaller APTAs naturally lead to easier SAT problems. This motivates our effort to improve the first step of the inference problem to obtain simpler SAT instances. While previous attempts have aimed at reducing the size of APTAs (Lang et al., 1998; Oncina and Garcia, 1992; Bugalho and Oliveira, 2005), we introduce a new and incremental construction of the APTAs that comes with a *minimality* guarantee for the acceptor of the given samples.

Contributions. We propose employing the (polynomial-time) incremental construction of minimal acyclic DFA learning algorithm (Daciuk et al., 2000) for minimal DFAs from a given set of *positive* samples. We offer two constructions based on it. The first consists of building two minimal DFAs, $\mathcal{D}^+$ and $\mathcal{D}^-$, for the positive samples S^+ and the negative ones S^-, respectively. When composing them, we set as rejecting the accepting states of $\mathcal{D}^-$ and use the DFA pair $(\mathcal{D}^+, \mathcal{D}^-)$ as the acceptor of $S = (S^+, S^-)$. For the second construction, our algorithm directly learns an APTA from S, hence considering both sets of positive and negative samples at the same time. As a consequence, we extend the algorithm to support APTA learning from the pair of labelled samples S. The obtained APTA is guaranteed to be the *minimum-size deterministic* acceptor for S.

We have implemented these techniques in our new tool DFAMiner and compared it with the state-of-the-art tools DFA-Inductor (Ulyantsev et al., 2015; Zakirzyanov et al., 2019) and DFA-Identify (Lauffer et al., 2022), on the benchmarks generated as described in (Ulyantsev et al., 2015; Zakirzyanov et al., 2019). Our experimental results demonstrate that DFAMiner builds smaller APTAs and is therefore significantly faster at minimising the DFAs than both DFA-Inductor and DFA-Identify.

To test the limitation of our technique, we employed it to extract deterministic safety or reachability automata as witness automata for parity game solving. With DFAMiner, we have established the lower bounds on the size of deterministic safety automata for parity games with up to 7 colours. In this case, the main bottleneck is no longer solving the Min-DFA inference problem, but the generation of the labelled samples, whose number is exponential in the length and the number of colours. To the best of our knowledge, this is the first time that Min-DFA inference tools have been applied to parity game solving. If they eventually scale, this may lead to new insights into the actual size of the minimal safety automata for solving parity games.

2. Preliminaries

In the whole paper, we fix a finite *alphabet* Σ of letters. A *word* is a finite sequence of letters in Σ. We denote with ε the empty word and with Σ^* the set of all finite words. When discarding the empty word, we restrict the set of words to $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A subset of Σ^* is a *finitary language*. Given a word u, we denote by $u[i]$ the i-th letter of u. For two given words u and w, we denote by $u \cdot w$ (uw, for short) the concatenation of u and w. We say that u is a *prefix* of u', denoted as $u \preceq u'$, if $u' = u \cdot v$ for some word $v \in \Sigma^*$. We denote by prefixes(u) the set of the prefixes of u, i.e., $\{\, v \in \Sigma^* \mid v \preceq u \,\}$. We also extend function prefixes to sets of words, i.e., we have $\mathsf{prefixes}(S) = \bigcup_{u \in S} \mathsf{prefixes}(u)$.

Automata. An automaton on finite words is called a 3-valued (deterministic) *finite automaton* (3DFA). A 3DFA $\mathcal{A}$ is formally defined as a tuple (Q, ι, δ, F, R), where Q is a finite set of states, $\iota \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and F, R and $D = Q \setminus (F \cup R)$ form a partition of Q where $F \subseteq Q$ is the set of *accepting* states, $R \subseteq Q$ is the set of *rejecting* states and D is the set of *don't-care* states. 3DFAs map all words in Σ^* to three values, i.e., accepting $(+)$, rejecting $(-)$ and don't-care $(?)$. This is why we call it a 3DFA.

A *run* of a 3DFA $\mathcal{A}$ on a not empty finite word u of length n is a sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \le i < n$, $q_{i+1} \in \delta(q_i, u[i])$. As usual, a finite word $u \in \Sigma^*$ is *accepted* (respectively, *rejected*) by a 3DFA $\mathcal{A}$ if there is a run $q_0 \cdots q_n$ over u such that $q_n \in F$ (respectively, $q_n \in R$). Naturally, a 3DFA $\mathcal{A}$ can be seen as a classification function in $\Sigma^* \to \{+, -, ?\}$. The usual deterministic finite automaton (DFA) is a 3DFA with $R = Q \setminus F$, i.e., a word is mapped to either $+$ or $-$. Both the classes of words *accepted* and *rejected* by 3DFAs are known to be regular languages.

We remark that the 3DFAs are very standard model for representing positive and negative samples in the literature. In (Alquezar and Sanfeliu, 1995), 3DFAs are called deterministic unbiased finite state automata.

For a given regular language, the Myhill-Nerode theorem (Myhill, 1957; Nerode, 1958) helps to obtain the minimal DFA. Similarly, it is suggested in (Chen et al., 2009) that we can identify equivalent words that reach the same state in the minimal 3DFA of a given function $L : \Sigma^* \to \{+, -, ?\}$. Let x, y be two word in Σ^* and $L \in (\Sigma^* \to \{+, -, ?\})$ be a function. We define an equivalence relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ as below:

$$x \sim_L y \text{ if, and only if, } \forall v \in \Sigma^*, L(xv) = L(yv).$$

We denote by $|\sim_L|$ the index of $\sim_L$, i.e., the number of equivalence classes defined by L. Let S be a given finite set of labelled samples. We can also see S as a classification function and it also induces an equivalence relation $\sim_S$. Further, we define $S^+ = \{u \in S : S(u) = +\}$, $S^- = \{u \in S : S(u) = -\}$ and $S^? = \Sigma^* \setminus S = \{u \in S : S(u) = ?\}$. Finally, we conclude with a straightforward proposition that follows from the fact that $|\sim_S|$ is bounded by the number of prefixes of S, i.e., $|\mathsf{prefixes}(S)|$.

FACT 1:
Let S be a finite set of labelled samples. Then the index of $\sim_S$ is also finite.

3. Overview of **DFAMiner**

Problem definition. Let $S = (S^+, S^-)$ be the given set of labelled samples. Our goal in this paper is to find a *minimal* DFA (Min-DFA) $\mathcal{D}$ for S such that for all $u \in \Sigma^*$, if $S(u) = \$$, where $\$ \in \{+, -\}$, then $\mathcal{D}(u) = \$$. We call the target DFA a minimal *separating* DFA[1] for S.

The passive learner for separating DFAs (Heule and Verwer, 2010; Zakirzyanov et al., 2019) usually works as follows: (1) Construct a 3DFA $\mathcal{M}$ recognising S and (2) Minimise the 3DFA $\mathcal{M}$ to a Min-DFA using a SAT solver. Our work mainly differs from the APTA construction for the first step. We will first describe the APTA construction in Section 3.1 and then give our proposal for the tool architecture in Section 3.2. In the remainder of the paper, we let $S = (S^+, S^-)$ be the given labelled sample set.

3.1. The construction of 3DFAs

Prior works (Zakirzyanov et al., 2019; Ulyantsev et al., 2015; Heule and Verwer, 2010) construct an automaton called *augmented prefix tree acceptor* (APTA) (Alquezar and Sanfeliu, 1995; Coste and Nicolas, 1998; Coste and Fredouille, 2003)[2] $\mathcal{P}$ that recognises S. The APTA $\mathcal{P} = (Q, \varepsilon, \delta, F, R)$ is formally defined as a 3DFA where $Q = \mathsf{prefixes}(S)$ is the set of states, ε is the initial state, $F = S^+$ is the set of accepting states, $R = S^-$ is the set of rejecting states, and $\delta(u, a) = ua$ for all $u, ua \in Q$ and $a \in \Sigma$. As mentioned in the introduction, the number of boolean variables and clauses used in the SAT problem is polynomial in the number of states of $\mathcal{P}$. The main issue is that the size of $\mathcal{P}$ increases dramatically with the growth of the number of samples in S and the length of the samples, This is not surprising, given that $\mathcal{P}$ maps every word in $\mathsf{prefixes}(S)$ to a unique state. To show this growth, we considered samples from parity game solving. Table 1 shows the size comparison between the APTA and its minimal 3DFA (Min-3DFA) representation. With 5 and 6 letters (in this case colours), we can observe that the Min-3DFAs can be much smaller than their corresponding APTA counterparts. In other words, there are a lot of equivalent and redundant states in APTAs that can be merged together.

In fact, since APTAs are acyclic (i.e., there are no cycles), we can minimise them with a linear-time backward traversal (Daciuk et al., 2000). The crucial step is how to identify whether two states are equivalent in the backward traversal (Daciuk et al., 2000). Our

---

1. In (Chen et al., 2009), the 3DFA that recognises S is called separating DFA for S.

2. APTAs are called prefix tree unbiased finite state automata in (Alquezar and Sanfeliu, 1995) and they are also similar to the prefix-tree Moore Machines in (Trakhtenbrot and Barzdin, 1973).

$|\Sigma|$	Length	Min-3DFA	APTA
5	7	438	53,277
5	8	541	209,721
5	9	644	835,954
5	10	747	3,369,694
6	7	1279	199,397
6	8	1807	930,870
6	9	2170	4,369,362
6	10	2533	20,689,546

Table 1: Size comparison between Min-3DFA and APTA on part of benchmarks for parity game solving.

solution is to use the equivalence relation $\sim_S$. Based on the definition of $\sim_S$, we define that two states $p, q \in Q$ are equivalent, denoted $p \equiv r$ if, and only if,

1. they have the same acceptance status, i.e., they are both accepting, rejecting or don't-care states;

2. for each letter $a \in \Sigma$, either they both have no successors or their successors are equivalent.

In the implementation, since we only store one representative state for each equivalence class, the second requirement can be changed as follows:

2'. for each letter $a \in \Sigma$, either they both have no successors or the same successor.

Therefore, it is easy to come up with an algorithm to minimise the given APTA tree $\mathcal{P}$ by doing the following:

1. We first collapse all accepting (respectively, rejecting) states without outgoing transitions to one accepting (respectively, rejecting) state without outgoing transitions, and put the two states in a map called *Register* that stores equivalence relation of states.

2. Then we perform *backward* traversal of states and check if there is a state whose successors are *all* in *Register*. For such states, we identify equivalent states by Rule 2', replace all equivalent states with their representative and put their representative in *Register*.

3. We repeat step 2 until all states, including the initial state, are in *Register*.

In this way, we are guaranteed to obtain the minimal 3DFA $\mathcal{M}$ that correctly recognises the given set S.

Further, we do not have to construct the APTA $\mathcal{P}$ in order to obtain the minimal 3DFA for the given samples. We have shown that it is possible to construct the minimal 3DFA on the fly, given that the samples are in the usual lexicographic order. If the samples are not in the lexicographic order, we will first order them and then proceed to the construction of its 3DFA. The details of the construction are deferred to Appendix A.
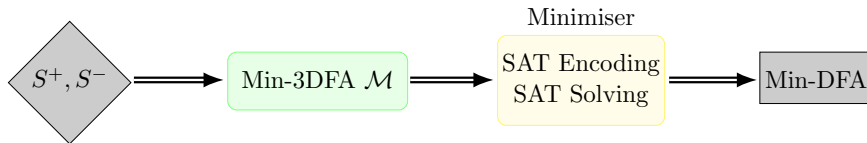
Figure 1: Workflow of DFAMiner with 3DFAs

Theorem 1 *The incremental construction produces the minimal 3DFA recognising exactly the sample set S.*

3.2. Our proposal

Our tool DFAMiner follows the classic two-step workflow. The advantage of DFAMiner over prior works is that DFAMiner has an access to an *incremental* construction that produces the minimal 3DFA $\mathcal{M}$ of S with respect to $\sim_S$. With the incremental construction, a natural workflow of DFAMiner is to first construct the minimal 3DFA $\mathcal{M}$ from $S = (S^+, S^-)$ and then find a minimal separating DFA $\mathcal{D}$ from $\mathcal{M}$ using a SAT solver. This approach is depicted in Figure 1. All the components labelled in green or blue are novel contributions made in our tool. We use the standard SAT-based minimal separating DFA extractions from APTAs (and hence 3DFAs) described in (Ulyantsev et al., 2015).

We observe that the minimal separating DFA finding algorithm (Ulyantsev et al., 2015) does not necessarily work only on 3DFAs, but also on multiple 3DFAs; see Appendix B for our formulation. This motivates us to ask a question: can we construct multiple 3DFAs for S? We give a affirmative answer to this question.

Our construction of double 3DFAs (dDFAs) from $S = (S^+, S^-)$ is formalised as follows: (1) Construct the minimal 3DFAs $\mathcal{D}^+$ and $\mathcal{D}^-$ for $(S^+, \emptyset)$ and $(S^-, \emptyset)$, respectively; (2) Make sure that $\mathcal{D}^+$ and $\mathcal{D}^-$ do not share the same state names; (3) Combine the two 3DFAs into a dDFA $\mathcal{N}$ where the initial states of $\mathcal{S}$ are the initial states of both $\mathcal{D}^+$ and $\mathcal{D}^-$, the transitions between states remain unchanged and we make the accepting states of $\mathcal{D}^-$ as rejecting states. The workflow of this construction is depicted in Figure 2. In this way, we obtain a dDFA $\mathcal{N}$ that recognises exactly the given set S. The empirical evaluation shows that the two types of workflows are incomparable and both have their place in the learning procedure.

The SAT encoding of the minimiser component is fairly standard and has been deferred to Appendix B. We can then gradually increase the number of states in the proposed separating DFA for S until the constructed SAT formula is satisfiable. It immediately follows that:

Theorem 2 *Our tool DFAMiner will output a minimal separating DFA for S.*

We remark that in (Alquezar and Sanfeliu, 1995), non-incremental and incremental constructions were also proposed to find small and even minimal 3DFAs M such that for each $u \in S^{\$}$, it holds that $M(u) = \$$ where $\$ \in \{+, -\}$. These two constructions are based on state merging techniques given in (Oncina and Garcia, 1992), with the worst-time complexity being $\mathcal{O}(|\mathsf{prefixes}(S)|^3)$, while the size of the intermediate 3DFA constructed by our incremental construction will not exceed $|\mathsf{prefixes}(S)|$. Furthermore, the resultant 3DFA M
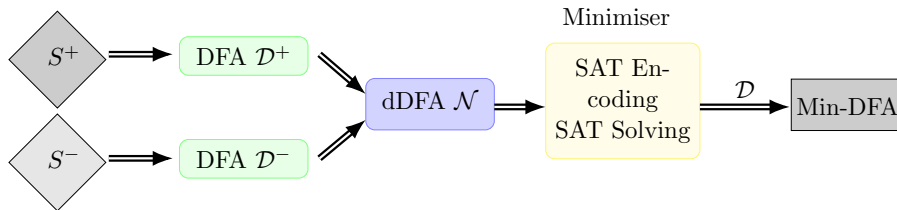
Figure 2: Workflow of DFAMiner with dDFAs

may accept (respectively, reject) more words than S^+ (respectively, S^-), while our incremental construction produces a minimal 3DFA recognising S exactly. As a consequence, their constructed 3DFA can be smaller (or even larger) than ours, and can no longer be used to extract the minimal separating DFA for S.

4. Evaluation

To further demonstrate the improvements of DFAMiner[3] over the state of the art, we conducted comprehensive experiments on standard benchmarks (Ulyantsev et al., 2015; Zakirzyanov et al., 2019). We compared with DFA-Inductor (Zakirzyanov et al., 2019) and DFA-Identify [4] (Lauffer et al., 2022), the state of the art tools publicly available for passive learning tasks. Unlike DFAMiner and DFA-Inductor, DFA-Identify uses a SAT encoding of graph coloring problems (Heule and Verwer, 2010) and the representative DFAs in the second step (Ulyantsev et al., 2015). Like DFA-Inductor, DFAMiner is also implemented in Python with PySAT (Ignatiev et al., 2018). We delegate all SAT queries to the SAT solver CaDical 1.5.3 in all tools (Biere et al., 2020). DFAMiner accepts the samples formalised in Abbadingo format[5].

The experiments of Table 2 were carried on an Intel i7-4790 3.60 GHz processor. Each index N, reports the results of 100 benchmark instances of random samples. Each benchmark has $50 \times N$ samples. For every index, we show the average time and the percentage of instances solved within 1200 seconds. The alphabet for the samples has two symbols while the size of the generated DFA is N. We compare four approaches to inferring Min-DFAs: DFA-Inductor, DFA-Identify, DFAMiner with 3DFA (DFA-MIN), and DFAMiner with double DFA (dDFA-MIN). An extended version of the table is reported in Appendix A.3. Both dDFA-MIN and 3DFA-MIN perform better than DFA-Inductor and DFA-Identify, on average they are three times faster. DFA-Inductor can minimise within 20 minutes instances up to level 13, while the two variants of DFAMiner can scale one more level and minimise one third of the instances of level 15. On these random samples the double DFA approach is slightly faster than the 3DFA one.

Figures 3 and 4 reports the comparison on the size of the APTA/dDFA (on the left) and minimisation time (on the right) for the previous benchmark. In these two figures, instead of the mean data, we show the individual data of each sample. Both DFA-Inductor and DFA-Identify build the same APTA (they differ for the encoding step), and as shown in

---

	DFA-Inductor		DFA-Identify		dDFA-MIN		3DFA-MIN	
N	avg	%	avg	%	avg	%	avg	%
4	0.12	100	0.09	100	0.03	100	0.02	100
5	0.29	100	1.38	100	0.06	100	0.05	100
6	0.67	100	2.33	100	0.30	100	0.18	100
7	1.81	100	4.12	100	0.80	100	0.73	100
8	3.57	100	9.70	100	1.29	100	1.25	100
9	10.84	100	20.76	100	3.83	100	3.78	100
10	50.91	100	44.57	100	17.88	100	16.80	100
11	154.73	100	128.69	100	55.12	100	59.46	100
12	399.52	96	373.65	99	144.27	100	162.39	100
13	850.04	74	785.93	82	390.10	99	418.62	97
14	1125.59	19	1099.92	23	809.88	76	861.10	69
15	1182.98	6	1197.61	1	1060.18	37	1062.02	34
16	1188.17	1	1184.82	3	1167.58	4	1164.02	5

Table 2: Comparison for the minimisation of DFAs from random samples of DFAMiner with DFA inductor. For each approach we report the mean minimisation time and the percentage of DFAs minimised within the time limit.
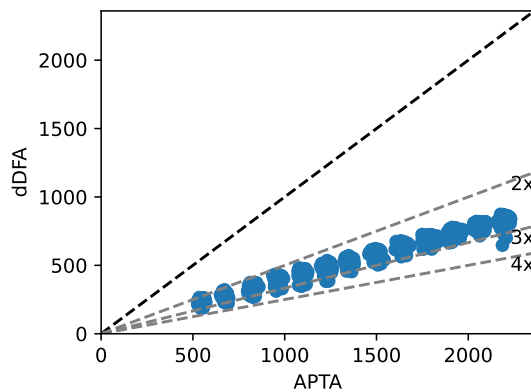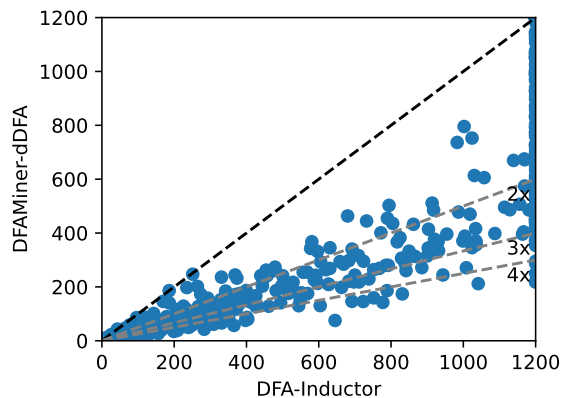

Figure 3: figure
Scatter plot on automata size

Figure 4: figure
Scatter plot on runtime (secs)

Figure 3, its size is three times larger than the dDFA built by DFAMiner, no matter how big is the final DFA. Figure 4, instead, shows that when using a double DFA, DFAMiner always performs better than DFA Inductor, on average three times faster with peaks of more than four times faster. We provide additional details and comparisons on minimisation time and 3DFA size in Appendix A.3.

Colours	2	3	4	5	6
DFA Size	3	3	5	5	9
Length	3	5	7	11	15
#Pos	3	130	1,645	9,375,269	4,399,883,736
#Neg	5	31	5,235	1,009,941	38,871,920,470

Table 3: Samples required to learn the minimal separating automata for solving parity games.

5. Application in parity game solving

It has been shown that the first quasi-polynomial parity game solving algorithm (Calude et al., 2017) essentially builds a separating automaton of quasi-polynomial size to distinguish runs with only winning cycles (cycles closed in a parity game where the highest colour occurring is *even*) from runs that contain only odd cycles (where the highest colour occurring is *odd*) (Czerwinski et al., 2019).

Without going into detail, we note that the hardest case occurs when the colours are unique (occur only once, hence, the colour identifies a node, thing that helps to detect cycles), and have implemented this as follows: we fix an alphabet with **c** different colours, a length $\ell > $ **c**, and a highest colour **c**. We must accept a word if all cycles are winning (e.g. 001212), rejecting it if all cycles are losing (e.g. 13123312). Words with winning and losing cycles (e.g. 21232) are don't-care words. A cycle occurs when a colour has repeated at least twice.

The resulting automata are always safety that reject all words that have not seen a winning cycle after (at most) ℓ steps, as well as some words that have seen both, winning and losing cycles (don't-care word), or, alternatively, reachability automata that accept all words that have not seen a losing cycle after at most ℓ steps (again, except don't-care ones). Thus, the size of the Min-DFA falls when increasing the sample length ℓ, and *eventually stabilises*. Using such a separating automaton reduces solving the parity game to solving a safety game (Bojańczyk and Czerwiński, 2018).

Separating automata build with the current state-of-the-art construction (Calude et al., 2017) grow quasi-polynomially, and since it is not known whether these constructions are optimal, we applied DFAMiner to learn the most succinct separating automata for the parity condition.

Table 3 shows the application of DFAMiner to the parity condition up to 7 colours (from 0 to 6). For each maximal colour we report the length required to build the minimal separating automaton, the size of the obtained DFA, and the number of all the positive and negative samples generated. Although most words have both wining and losing cycles (don't-care words), the positive and negative samples grow *exponentially*, too, which is why we stopped at 7 colours.

While the APTA size constructed by DFA-Inductor grows exponentially, the sizes of dDFAs and 3DFAs seem to grow only constantly when increasing the length of the samples for a fixed colour number. We report the details in Table 6 in Appendix C.1. Consequently, *all* versions of DFA-Inductor were only able to solve cases with at most 4 colours, while

DFAMiner can manage to solve cases up to 6 colours and length 16. To further push the limit of DFAMiner for parity game solving, we have also provided an efficient SAT encoding for parity games (in Appendix C.2). With the constructions for both 3DFAs and dDFAs and the efficient encoding, the bottleneck of the whole procedure is no longer solving the Min-DFA inference problem, but the generation of samples. With a better sample generation approach, we believe that this application can give insights on the structure of minimal safety automata for an arbitrary number of colours.

6. Discussion and Future Work

We propose a novel and more efficient way to build APTAs for the Min-DFA inference problem. Our contribution focuses on a compact representation of the positive and negative samples and, therefore, provides the leeway to benefit from further enhancements in solving the encoded SAT problem.

Natural future extensions of our approach include implementing the tight encoding of symmetry breaking (Zakirzyanov et al., 2019). Another easy extension of our construction is to learn a set of decomposed DFAs (Lauffer et al., 2022), thus improving the overall performance as well. A more complex future work is to investigate whether or not one can similarly construct a deterministic Büchi automaton based on ω-regular sets of accepting, rejecting, and don't-care words that provides a minimality guarantee for a given set of labelled samples.

References

R Alquezar and 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*, volume 94, pages 291–300, 1995.

Florent Avellaneda and Alexandre Petrenko. Learning minimal DFA: taking inspiration from RPNI to improve SAT approach. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2019. doi: 10.1007/978-3-030-30446-1_13. URL https://doi.org/10.1007/978-3-030-30446-1_13.

Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. doi: 10.1109/TC.1972.5009015. URL https://doi.org/10.1109/TC.1972.5009015.

M. Bojańczyk and W. Czerwiński. *An Automata Toolbox*. unpublished, 2018.

Miguel M. F. Bugalho and Arlindo L. Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognit.*, 38(9):1457–1467, 2005. doi: 10.1016/J.PATCOG.2004.03.027. URL https://doi.org/10.1016/j.patcog.2004.03.027.

C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding Parity Games in Quasipolynomial Time. In *Symposium on Theory of Computing 17*, pages 252–263. Association for Computing Machinery, 2017.

Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Learning minimal separating dfa's for compositional verification. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2009. doi: 10.1007/978-3-642-00768-2_3. URL https://doi.org/10.1007/978-3-642-00768-2_3.

François Coste and Daniel Fredouille. Unambiguous automata inference by means of state-merging methods. In Nada Lavrac, Dragan Gamberger, Ljupco Todorovski, and Hendrik Blockeel, editors, *Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, volume 2837 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2003. doi: 10.1007/978-3-540-39857-8_8. URL https://doi.org/10.1007/978-3-540-39857-8_8.

François Coste and Jacques Nicolas. Regular inference as a graph coloring problem. In *IWGI*, 1997.

François Coste and Jacques Nicolas. How considering incompatible state mergings may reduce the DFA induction search tree. In Vasant G. Honavar and Giora Slutzki, editors, *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, volume 1433 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 1998. doi: 10.1007/BFB0054076. URL https://doi.org/10.1007/BFb0054076.

W. Czerwinski, L. Daviaud, N. Fijalkow, M. Jurdzinski, R. Lazic, and P. Parys. Universal Trees Grow Inside Separating Automata: Quasi-Polynomial Lower Bounds for Parity Games. In *Symposium on Discrete Algorithms 19*, page 2333–2349. SIAM, 2019.

Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard E. Watson. Incremental construction of minimal acyclic finite state automata. *Comput. Linguistics*, 26(1):3–16, 2000. doi: 10.1162/089120100561601. URL https://doi.org/10.1162/089120100561601.

Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognit.*, 38(9):1332–1348, 2005. doi: 10.1016/J.PATCOG.2005.01.003. URL https://doi.org/10.1016/j.patcog.2005.01.003.

Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi: 10.1007/978-3-540-24605-3_37. URL https://doi.org/10.1007/978-3-540-24605-3_37.

E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37 (3):302–320, 1978. doi: 10.1016/S0019-9958(78)90562-4. URL https://doi.org/10.1016/S0019-9958(78)90562-4.

Olga Grinchtein, Martin Leucker, and Nir Piterman. Inferring network invariants automatically. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 483–497. Springer, 2006. doi: 10.1007/11814771_40. URL https://doi.org/10.1007/11814771_40.

Marijn Heule and Sicco Verwer. Exact DFA identification using SAT solvers. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2010. doi: 10.1007/978-3-642-15488-1_7. URL https://doi.org/10.1007/978-3-642-15488-1_7.

Alexey Ignatiev, António Morgado, and João Marques-Silva. Pysat: A python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. doi: 10.1007/978-3-319-94144-8_26. URL https://doi.org/10.1007/978-3-319-94144-8_26.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In Vasant G. Honavar and Giora Slutzki, editors, *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998. doi: 10.1007/BFB0054059. URL https://doi.org/10.1007/BFb0054059.

Niklas Lauffer, Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, Ameesh Shah, and Sanjit A. Seshia. Learning deterministic finite automata decompositions from examples and demonstrations. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 1–6. IEEE, 2022. doi: 10.34727/2022/ISBN.978-3-85448-053-2_39. URL https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_39.

John Myhill. Finite automata and the representation of events. In *Technical Report WADD TR-57-624*, page 112–137, 1957.

Daniel Neider. Computing minimal separating dfas and regular invariants using SAT and SMT solvers. In Supratik Chakraborty and Madhavan Mukund, editors, *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2012. doi: 10.1007/978-3-642-33386-6_28. URL https://doi.org/10.1007/978-3-642-33386-6_28.

Daniel Neider and Nils Jansen. Regular model checking using solver technologies and automata learning. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, volume 7871 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2013. doi: 10.1007/978-3-642-38088-4_2. URL https://doi.org/10.1007/978-3-642-38088-4_2.

Anil Nerode. Linear automaton transformations. In *American Mathematical Society*, page 541–544, 1958.

Jos'e Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

Charles P. Pfleeger. State reduction in incompletely specified finite-state machines. *IEEE Trans. Computers*, 22(12):1099–1102, 1973. doi: 10.1109/T-C.1973.223655. URL https://doi.org/10.1109/T-C.1973.223655.

Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *J. ACM*, 40(1):95–142, 1993. doi: 10.1145/138027.138042. URL https://doi.org/10.1145/138027.138042.

Rick Smetsers, Paul Fiterau-Brostean, and Frits W. Vaandrager. Model learning as a satisfiability modulo theories problem. In Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira, editors, *Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings*, volume 10792 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2018. doi: 10.1007/978-3-319-77313-1_14. URL https://doi.org/10.1007/978-3-319-77313-1_14.

Boris Avraamovich Trakhtenbrot and Ya M Barzdin. *Finite automata: Behavior and synthesis*. Elsevier, 1973.

Vladimir Ulyantsev, Ilya Zakirzyanov, and Anatoly Shalyto. Bfs-based symmetry breaking predicates for DFA identification. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 611–622. Springer, 2015. doi: 10.1007/978-3-319-15579-1_48. URL https://doi.org/10.1007/978-3-319-15579-1_48.

Sicco Verwer and Christian 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, pages 638–642. IEEE Computer Society, 2017. doi: 10.1109/ICSME.2017.58. URL https://doi.org/10.1109/ICSME.2017.58.

Ilya Zakirzyanov, António Morgado, Alexey Ignatiev, Vladimir Ulyantsev, and João Marques-Silva. Efficient symmetry breaking for sat-based minimum DFA inference. In Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira, editors, *Language and Automata Theory and Applications - 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings*, volume 11417 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2019. doi: 10.1007/978-3-030-13435-8_12. URL https://doi.org/10.1007/978-3-030-13435-8_12.

Appendix A. Minimal APTA construction

The algorithm can be seen as the combination of the construction of the APTA and its minimisation based on the backward traversal of the APTA tree. We will enumerate the input samples S one by one in their lexicographical order; this is important for identifying the states/nodes in the APTA tree that have been completely traversed and will remain unchanged after some point, in order to obtain an on-the-fly minimisation. We will highlight the two fundamental components: the minimisation and the backward traversal/construction of the APTA tree in the sequel.

Let us consider a simpler situation where the *full* APTA tree $\mathcal{P}$ is already given. The most important thing in the minimisation component is to decide whether two states p and q are equivalent and it has already been described.

By using the equivalence relation we can obtain the minimal 3DFA $\mathcal{M}$ that correctly recognises the given set S. Moreover, if we use a hash map for storing all representative states in *Register*, the minimisation algorithm above runs in linear time with respect to the number of states in $\mathcal{P}$. However, as we can see in Table 1, the APTAs can be *significantly larger* than the corresponding minimal 3DFAs. Hence, it is vital to avoid the full construction of the APTA tree $\mathcal{P}$ of S. The key to the on-the-fly construction is to know when a state has been *completely* traversed during construction.

To this end, we can just assume that the samples are already ordered in the usual lexicographical order. The comparison works as follows. Assume that we already have a default order on the letters in Σ. For two words u and u', we first compare their first $\min(|u|, |u'|)$ letters: (1) if we find a word that has a smaller letter than the other at the same position, then that word is smaller, (2) if all letters are the same and u has the same length as u', then they are equal, otherwise (3) the word that is longer than the other word is greater.

Assume that $S = u_1, u_2, \cdots, u_\ell$ is in order. We describe below how to tell when a state is *impossible* to have more successors and it is ready to find its representative state. Assume that current APTA is $\mathcal{P}_i = (Q_i, \{\iota\}, \delta_i, F_i, R_i)$ and we now input the next sample u_{i+1}, where $i \geq 0$.

When $i = 0$, $\mathcal{P}_0$ is of course minimal since $\mathcal{P}_0$ only has a state ι without any outgoing transitions. For technical reason, we let $u_0 = \varepsilon$, which may not appear in the sample set

S. (Note that if there is an empty word ε in S, ι will be set to accepting or rejecting accordingly.)

We first run u_{i+1} on $\mathcal{P}_i$. We let $u_{i+1} = x \cdot y_{i+1}$ and assume that $x \in \mathsf{prefixes}(u_{i+1})$ be the *longest* word such that $\delta_i(\iota, x) \neq \emptyset$. Let $p = \delta_i(\iota, x)$. Then, all states along the run of $\mathcal{P}_i$ over x are not ready to find their representatives, as p needs to add more reachable states to run the suffix y_{i+1}. Moreover, we observe that x must be a prefix of u_i, i.e., $x \in \mathsf{prefixes}(u_i)$. This is because every word that has a *complete* run in $\mathcal{P}_i$ must *not* be greater than u_i. By definition of the lexicographic order, if x is not a prefix of u_i, then x must be smaller than $u_i[0 \cdots |x|]$. This leads to the contradiction that u_i is greater than u_{i+1}. Let $u_i = x \cdot y_i$ and $\rho = p_0 \cdots p_{|u_i|}$ where $p_0 = \iota$ and $p_{|x|} = p$. We can show that all the states p_k with $k > |x|$ in the run of $\mathcal{P}_i$ over $u_i = x \cdot y_i$ are ready to merge with their representative, as they must not have more reachable states. Assume that there is a state p_ℓ with $\ell > |x|$ reached over a sample u_h with $h > i$, then it is easier to lead the contradiction that u_h is smaller than u_{i+1}. Therefore, we can similarly identify the representatives for those states and merge them in the usual backward manner. It follows that all the states except the ones in the run of u_{i+1} in the 3DFA $\mathcal{P}_{i+1}$ are already consistent with respect to $\sim_S$; thus, there is no need to modify them afterwards. After we have input all the samples, we only need to merge all states in the run over u_ℓ with their equivalent states. In this way, we are guaranteed to obtain the minimal 3DFA $\mathcal{S}$ for S in the end.

The formal procedure of the above incremental construction of the minimal 3DFA from S is given in Algorithm 1. Note that, when looking for the run from p over the *last* input sample, we only need to find the successors over the maximal letter by the last_child function. In this way, when we reach the last state of the run, we then can begin to identify equivalent states in a backward manner, as described in the subprocedure replace_or_register. Moreover, in the function add_suffix(p, y), we just create the run from p over y and set the last state to be accepting or rejecting depending on the label of u. In fact, we only extend the equivalence relation $\equiv$ of (Daciuk et al., 2000) in replace_or_register to support the accepting, rejecting and don't-care states, as described before.

Theorem 3 *Let S be a finite labelled set of ordered samples. Algorithm 1 returns the correct and minimal 3DFA recognising S.*

Proof The proof is basically an induction on the number of input samples and has been overlapped with the intuition described above. We thus omit it here. ∎

Appendix B. SAT Encoding

In this section, we show that how to obtain the minimal separating DFAs from 3DFAs/3NFAs. It is known that minimising DFAs with don't-care words is NP-complete (Pfleeger, 1973). We will take the advantage of the current powerful solvers for Boolean Satisfiability (SAT) problems to look for minimal DFAs.

B.1. SAT-based encoding of minimisation

We assume that we are given a dDFA $\mathcal{N} = (\mathcal{T}, A, R)$, where $\mathcal{T} = (Q, I, \delta)$ is obtained from two DFAs $\mathcal{D}^+$ and $\mathcal{D}^-$. We look for a separating DFA $\mathcal{D}$ of n states for $\mathcal{N}$ such that for each

Algorithm 1: Incremental construction of the minimal 3DFA from S

procedure MAIN PROCEDURE(Sample Set U)

    $Register := \emptyset$

    while U *has next sample* u **do**

    $x := \mathsf{common\_prefix}(u)$

    $p := \delta(\iota, x)$ $\triangleright$ the last state over the common prefix x

    $y := u[|x|\ldots]$ $\triangleright$ the remaining suffix of u

    if *has_children*(p) **then**

        $\mathsf{replace\_or\_register}(p)$ $\triangleright$ merge/register all states after p

     end

    $\mathsf{add\_suffix}(p, y)$ $\triangleright$ create run to accept suffix y from p

    end

    $\mathsf{replace\_or\_register}(\iota)$ $\triangleright$ merge the run over the last sample

procedure REPLACE_OR_REGISTER(p)

    $r := \mathsf{last\_child}(p)$ $\triangleright$ obtain the successor over the maximal letter

    if *has_children*(r) **then**

    $\mathsf{replace\_or\_register}(r)$ $\triangleright$ recursively obtain the run over last sample

    end

    if $\exists q \in Q.(q \in Register \wedge q \equiv r)$ **then**

    $\mathsf{last\_child}(p) := q$ $\triangleright$ merge with its representative

    end

    else

    $Register := Register \cup \{r\}$ $\triangleright$ set the first state of each class as representative

    end

$u \in \Sigma^*$, if $\mathcal{N}(u) = \$$, then $\mathcal{D}(u) = \$$, where $\$ \in \{+, -\}$. Clearly the size of $\mathcal{D}$ is bounded by the size of the TS, i.e. $0 < n \le |Q|$, since we can obtain a DFA from the dDFA by simply using $\mathcal{D}^+$ (or the complement of $\mathcal{D}^-$). Nevertheless, we aim at finding the minimal such integer n.

To do this, we encode our problem as a SAT problem such that there is a separating complete DFA $\mathcal{D}$ with n states if, and only if, the SAT problem is satisfiable. We apply the standard propositional encoding (Neider, 2012; Neider and Jansen, 2013; Ulyantsev et al., 2015; Zakirzyanov et al., 2019). For simplicity, we let $\{0, \cdots, n-1\}$ be the set of states of $\mathcal{D}$, such that 0 is the initial one. To encode the target DFA $\mathcal{D}$, we use the following variables:

- the transition variable $e_{i,a,j}$ denotes that $i \overset{a}{\to} j$ holds, i.e. $e_{i,a,j}$ is true if, and only if, there is a transition from state i to state j over $a \in \Sigma$, and

- the acceptance variable f_i denotes that $i \in F$, i.e. f_i is true if, and only if, the state i is an accepting one.

Once the problem is satisfiable, from the values of the above variables, it is easy to construct the DFA $\mathcal{D}$. To that end, we need to tell the SAT solver how the DFA should look like by giving the constraints encoded as clauses. For instance, to make sure the result DFA is indeed deterministic and complete, we need following constraints:

D1 Determinism: For every state i and a letter $a \in \Sigma$ in $\mathcal{D}$, we have that $\neg e_{i,a,j} \vee \neg e_{i,a,k}$ for all $0 \le j < k < n$.

D2 Completeness: For every state i and a letter $a \in \Sigma$ in $\mathcal{D}$, $\bigvee_{0 \le j < n} e_{i,a,j}$ holds.

Moreover, to make sure the obtained DFA $\mathcal{D}$ is separating for $\mathcal{N}$, we also need to perform the product of the target DFA $\mathcal{D}$ and $\mathcal{N}$. In order to encode the product, we use extra variables $d_{p,i}$, which indicates that the state p of $\mathcal{N}$ and the state i of $\mathcal{D}$ can both be reached on some word u. The constraints we need to enforce that $\mathcal{D}$ is separating for $\mathcal{N}$ are formalised as below:

D3 Initial condition: $d_{\iota,0}$ is true for all $\iota \in I$. (0 is the initial state of $\mathcal{D}$.

D4 Acceptance condition: for each state i of $\mathcal{D}$,

    D4.1 Accepting states: $d_{p,i} \Rightarrow f_i$ holds for all $p \in A$

    D4.2 Rejecting states: $d_{p,i} \Rightarrow \neg f_i$ holds for all $p \in R$;

D5 Transition relation: for a pair of states i, j in $\mathcal{D}$, $d_{p,i} \wedge e_{i,a,j} \Rightarrow d_{p',j}$ where $p' = \delta(p, a)$ for all $p \in Q$ and $a \in \Sigma$.

Let $\phi_n^{\mathcal{N}}$ be the conjunction of all these constraints. Then, we obtain the following theorem.

Theorem 4 *Let $\mathcal{N}$ be a dDFA of S and $n \in \mathbb{N}$. Then $\phi_n^{\mathcal{N}}$ is satisfiable if, and only if, there exists a complete DFA $\mathcal{D}$ with n states that is separating for $\mathcal{N}$.*

The formula $\phi_n^{\mathcal{N}}$ contains $\mathcal{O}(n^3 \cdot |\Sigma| + n^2 \cdot |Q| \cdot |\Sigma|)$ constraints.

When looking for separating DFAs, the SAT solver may need to inspect multiple isomorphic DFAs that only differ in their state names for satisfiability. If those isomorphic DFAs are not separating for $\mathcal{N}$, then the SAT solver still has to prove this for each DFA. To reduce the search space, it suffices to check only a representative DFA for all isomorphic DFAs (Ulyantsev et al., 2015). We will describe the representative DFA in the following section.

B.2. SAT encoding of the representative DFA

The representative DFA $\mathcal{D}$ is induced by restricting the structure of its breath-first search (BFS) tree τ. In our setting, an edge of the BFS tree is a directed connection from one node to another and it is labelled by a letter in Σ. In this section, we need to enforce an order on the letters in Σ. For simplicity, we let $\Sigma = \{0, \cdots, \mathbf{c}-1\}$ where $\mathbf{c} > 0$. Recall that the set of nodes in the tree is the set of states of $\mathcal{D}$. Below we list the requirements of the BFS tree.

A1 Minimal parent:

    A1.1 If there is an edge from node i to node j in the BFS tree, then $i < j$ and i is the minimal state that reaches j via a transition in $\mathcal{D}$, and

    A1.2 If j is a child node of i and $j+1$ is a child node of k, then, $i < k$. This is because we enforce that smaller children must have smaller parents.

A2 Minimal letter edge:

    A2.1 If the edge from i to j is labelled with letter a in the BFS tree, a must be the minimal letter from i to j in $\mathcal{D}$, and

    A2.2 If there are edges from i to j over a_1 and to k over a_2 in the BFS tree, and $a_1 < a_2$, then $j < k$. Note that it is impossible for a_1 and a_2 to be equal since $\mathcal{D}$ is deterministic.

In order to encode the above requirements, we need the following three types of boolean variables. For a pair of states $0 \leq i, j < n$ and a letter $0 \leq a < \mathbf{c}$:

- The edge variable $m_{i,a,j}$ of the BFS tree denotes that $m_{i,a,j}$ is true if, and only if, the BFS tree has an edge from node i to node j labelled with the letter a.

- The parent variable $p_{j,i}$ indicates that $p_{j,i}$ is true if, and only if, node j is a child node of node i in the BFS tree.

- The transition variable $t_{i,j}$ indicates that $t_{i,j}$ is true if, and only if, there is a transition from state i to state j in $\mathcal{D}$.

Now we can give the following constraints that are needed to represent the requirements of the BFS tree below.

B1 Minimal parent:

	DFA-Inductor				dDFA-MIN				DFA-MIN			
N	min	avg	max	solved%	min	avg	max	solved%	min	avg	max	solved%
4	0.01	0.12	0.20	100	0.01	0.03	0.49	100	0.01	0.02	0.09	100
5	0.01	0.29	0.62	100	0.01	0.06	1.01	100	0.01	0.05	0.22	100
6	0.02	0.67	1.43	100	0.01	0.30	1.82	100	0.01	0.18	1.91	100
7	0.02	1.81	5.86	100	0.03	0.80	7.09	100	0.04	0.73	3.61	100
8	0.02	3.57	10.26	100	0.01	1.29	9.06	100	0.01	1.25	4.53	100
9	0.02	10.84	39.84	100	0.02	3.83	11.94	100	0.01	3.78	15.71	100
10	3.85	50.91	199.04	100	0.69	17.88	62.40	100	0.63	16.80	57.27	100
11	1.90	154.73	691.79	100	0.37	55.12	263.82	100	0.33	59.46	288.10	100
12	15.86	399.52	1200	96	4.30	144.27	536.74	100	5.12	162.39	705.19	100
13	84.84	850.04	1200	74	28.61	390.10	1200	99	15.86	418.62	1200	97
14	214.76	1125.59	1200	19	77.56	809.88	1200	76	68.03	861.10	1200	69
15	587.66	1182.98	1200	6	225.56	1060.18	1200	37	218.22	1062.02	1200	34
16	17.22	1188.17	1200	1	9.55	1167.58	1200	4	5.06	1164.02	1200	5

Table 4: Full experimental results on generated benchmarks

B1.1 for two nodes $0 \le i < j < n$ in the BFS tree, we have $p_{j,i} \Leftrightarrow t_{i,j} \land \bigwedge_{0 \le k < i} \neg t_{k,j}$, and

B1.2 for every triple $0 \le k < i < j < n$, we have $p_{j,i} \Rightarrow \neg p_{j+1,k}$.

B2 Minimal letter edge:

B2.1 for every pair $0 \le i < j < n$ in the BFS tree and a letter $a \in \Sigma$, we have $m_{i,a,j} \Leftrightarrow (e_{i,a,j} \land \bigwedge_{0 \le b < a} \neg e_{i,b,j})$.

B2.2 for every pair $0 \le i < j < n$ of nodes and a pair $0 \le a < b < \mathbf{c}$ of letters, we have $p_{j,i} \land p_{j+1,i} \land m_{i,b,j} \Rightarrow \neg m_{i,a,j+1}$.

B3 Edge consistency: for each pair $0 \le i < j < n$, $t_{i,j} \Leftrightarrow \bigvee_{0 \le a < \mathbf{c}} e_{i,a,j}$ holds.

B4 Existence of a parent: for each node $0 \le i < n$ in the BFS tree, we have that $\bigvee_{0 \le j < i} p_{j,i}$ holds.

Let ϕ_n^τ be the conjunction of all these constraints. Then we obtain the following theorem.

Theorem 5 *Let $\mathcal{N}$ a dDFA of S and $n \in \mathbb{N}$. Then, $\phi_n^{\mathcal{N}} \land \phi_n^\tau$ is satisfiable if, and only if, there exists a separating DFA $\mathcal{D}$ with n states, with respect to $\mathcal{N}$.*

We remark that the formula ϕ_n^τ contains $\mathcal{O}(n^3 + n^2 \cdot |\Sigma|^2)$ constraints.

Appendix C. More experimental results

In this section we provide additional experiments. Table 4 extends Table 2 by reporting also the minimum and maximum running time for each technique. Since the samples are randomly generated, the minimum and maximum values can show irregular peaks, while the average tends to converge to the timeout value.

In Table 5 we report, on the same benchmark as the previous table, the minimum, maximal, and average size of the APTA for DFA-Inductor and DFA-Identify (they build the exact same APTA) and the size of the 3DFA for the two variants of DFAMiner, namely the double DFA and 3DFA.

	DFA-Inductor/DFA-Identify			dDFA-MIN			DFA-MIN		
N	min	avg	max	min	avg	max	min	avg	max
4	526	549	570	189	230	264	188	207	227
5	650	675	695	221	281	321	220	248	270
6	798	822	860	267	338	374	266	299	322
7	935	963	994	312	393	437	311	347	373
8	1073	1097	1126	347	439	481	346	389	418
9	1195	1225	1260	383	485	535	382	425	449
10	1322	1349	1378	454	528	571	427	461	485
11	1459	1496	1542	503	580	622	474	510	530
12	1602	1643	1683	563	638	679	520	559	586
13	1748	1791	1836	618	690	725	556	604	632
14	1887	1930	1968	664	741	777	605	648	670
15	2018	2063	2098	716	786	821	651	690	719
16	2159	2196	2230	647	827	873	632	726	755

Table 5: Full automata size results on generated benchmarks before the SAT minimisation

In the following figures we propose a pairwise comparison of all the techniques on the minimisation time. Figure 5 shows DFAMiner and DFA-Inductor. Interestingly, in Figure 8, the two variants of DFAMiner do not overlap, there are cases up to two times easier to minimise for the double DFA variant and cases up to three times easier for the 3DFA variant, which, as shown in Table 4, on average is slightly faster. In Figure 7 we compare the double DFA variant of DFAMiner and DFA-Inductor 2 which employs a different, and more efficient, encoding of DFA called tightDFS. This heuristic can also be applied to our tool. Although the application of tight encoding, DFA-Inductor 2 is still slower that DFAMiner up to four times slower. However, in some cases it can be slightly faster or solve cases on which dDFA hit the timeout (seven cases with N=14). Similarly, Figure 7 compares the 3DFA variant of DFAMiner and DFA-Inductor 2. In this case the number of DFAs not solved by DFAMiner are eight, all with N=14. Finally, Figures 9 and 10 show the comparison of DFA-Identify and the two variants of DFAMiner.

C.1. Size growth of APTA, dDFA and 3DFA in parity game solving

In Table 6, we can see that the number of positive and negatives samples grow exponentially, even if they eventually take up below 20% of all samples. On the other hand, we can see that for a fixed colour number, the sizes of dDFAs and 3DFAs grow constantly when increasing the word length by 1. For instance, for the colour number 6, the size of 3DFA increases by at most 528 and eventually by 363, and dDFA by 363 when increasing the length by 1. A huge save in the number of states representing the samples thus leads to a significantly better performance in solving the Min-DFA inference problems for parity games.
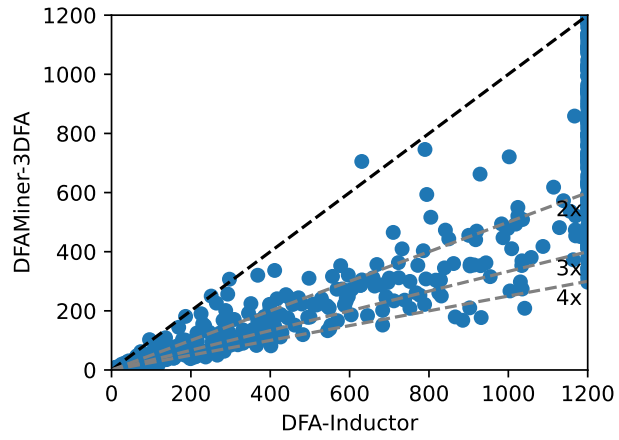
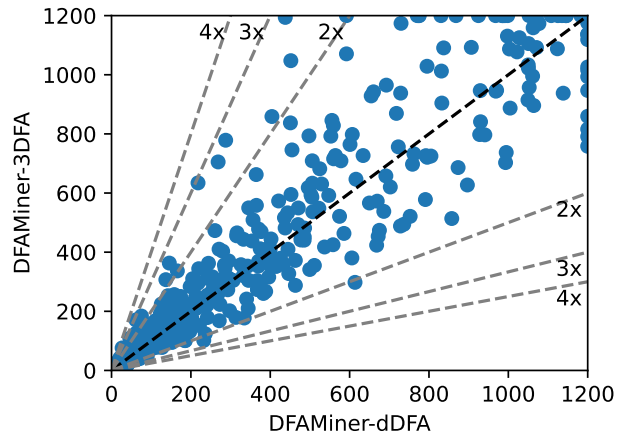Figure 5: Minimisation time of DFAMiner with double DFA and DFA inductor on 1,300 DFAs.

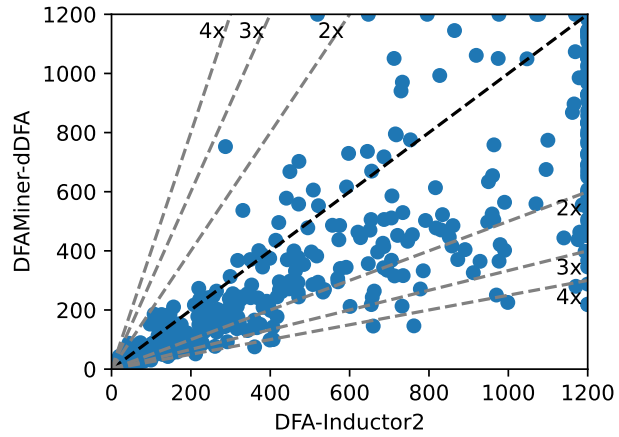Figure 6: Minimisation time of the two DFAMiner variants: the double DFA and 3DFA.

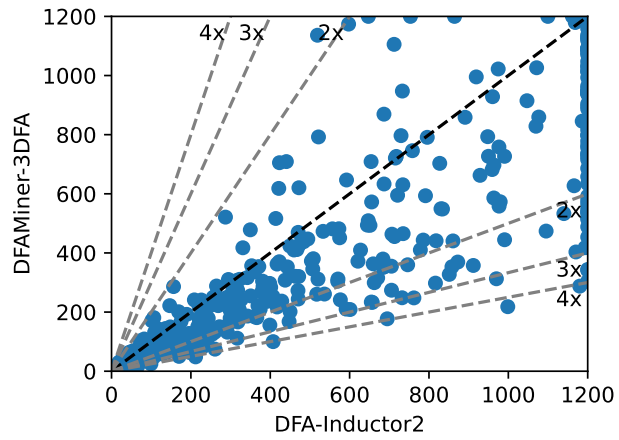Figure 7: Minimisation time of the DFA-Inductor 2 and DFAMiner with dDFA.

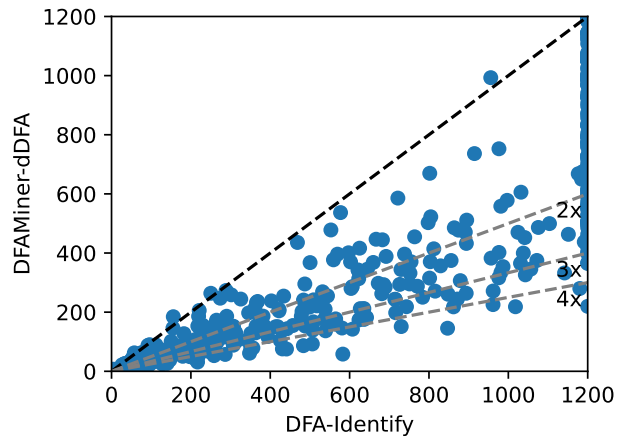Figure 8: Minimisation time of the DFA-Inductor 2 and DFAMiner with 3DFA.

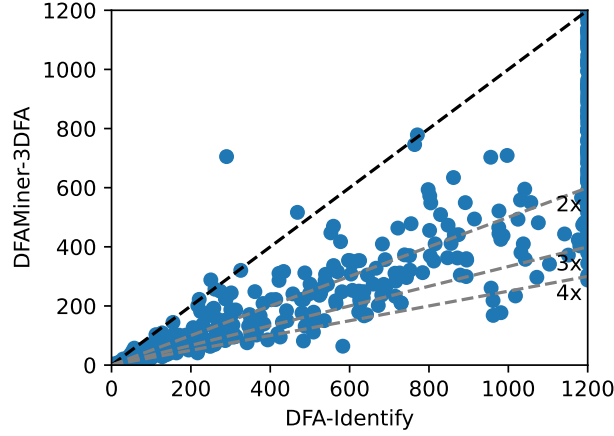Figure 9: Minimisation time of the DFA-Identify and DFAMiner with dDFA.

Figure 10: Minimisation time of the DFA-Identify and DFAMiner with 3DFA.

Colours	Length	dDFA	3DFA	APTA	Samples	#Pos	#Neg
2	3	12	8	15	2^3	3	5
3	4	28	23	111	3^4	51	20
3	5	38	33	266	3^5	130	31
4	5	84	82	1,083	4^5	274	488
4	6	117	122	3,311	4^6	669	1,599
4	7	150	155	10,076	4^7	1,645	5,235
5	6	269	301	13,634	5^6	7,233	3,067
5	7	372	438	53,277	5^7	30,332	9,625
5	8	475	541	209,721	5^8	127,194	30,456
5	9	578	644	835,954	5^9	533,305	97,228
5	10	681	747	3,369,694	5^{10}	2,236,023	312,568
5	11	784	850	13,704,486	5^{11}	9,375,269	1,009,941
6	7	986	1,279	199,397	6^7	53,556	104,095
6	8	1,349	1,807	930,870	6^8	216,298	517,590
6	9	1,712	2,170	4,369,362	6^9	878,823	2,573,621
6	10	2,075	2,533	20,689,546	6^{10}	3,595,591	12,795,642
6	11	2,438	2,896	-	6^{11}	14,799,059	63,616,339
6	12	2,801	3,259	-	6^{12}	61,192,124	316,286,133
6	13	3,164	3,622	-	6^{13}	253,881,602	1,572,522,807
6	14	3,527	3,985	-	6^{14}	1,055,948,048	7,818,368,374
6	15	3,890	4,348	-	6^{15}	4,399,883,736	38,871,920,470
6	16	4,253	4,711	-	6^{16}	18,357,865,115	193,266,275,998

Table 6: The size comparison between APTAs, 3DFAs and dDFAs for parity game solving. Some of the missing data for APTA is due to the fact that there is no enough RAM memory to build it.

C.2. Encoding of safety automata

In this section we outline the encoding employed for minimising safety automata from parity samples. In particular, we introduce additional contraints to reduce the space of the search of the DFA. Because of the huge number of samples, we have tweaked the encodings described in the Section B for efficiency, taking properties of separating automata for parity games into account.

Let $\mathbf{c}$ be the number of colours in the parity game, i.e. the set of the colours is $[0, \mathbf{c}-1]$. We use the previously defined transition variables from a state i to j reading a letter (colour) a as $e_{i,a,j}$ and the acceptance variables f_i, with $0 \leq i, j < n$ and $0 \leq a < \mathbf{c}$. Moreover, to make the SAT problem easier, we set 0 as the initial state and $n-1$ as the sink state (for safety or co-safety acceptance). Let ℓ be a positive integer. We denote by $\mathsf{odd}(\ell)$ (respectively, $\mathsf{even}(\ell)$) the set of odd (respectively, even) numbers in the set $[0, \ell-1]$. By $\mathsf{opp}(\mathbf{c})$ we denote the set of colours having the opponent priority as the highest colour, i.e. if $\mathbf{c}-1$ is even, then $\mathsf{opp}(\mathbf{c}) = \mathsf{odd}(\mathbf{c})$, and $\mathsf{opp}(\mathbf{c}) = \mathsf{even}(\mathbf{c})$ otherwise. The additional constraints are as follow:

P1 Initial state loops. Whenever the initial state 0 reads a colour of the same parity as the highest one, then it loops over itself. Therefore, if $\mathbf{c}-1$ is even, then we have $\bigwedge_{a \in \mathsf{even}(\mathbf{c})} e_{0,a,0}$, otherwise $\bigwedge_{a \in \mathsf{odd}(s)} e_{0,a,0}$ holds.

P2 Initial state outgoing transitions. Whenever the initial state 0 reads a colour of the opponent parity as $\mathbf{c}-1$, it reaches a different state than 0 and $n-1$. Hence, we have $\bigwedge_{a \in \mathsf{opp}(\mathbf{c})} (\bigvee_{0 < i < n-1} e_{0,a,i})$.

P3 Sink state incoming transitions[6]. Whenever a state $i \neq n-1$ reads a colour of the same parity as the highest one, then it cannot reach the sink state $n-1$. The constraint is formalised as $\bigwedge_{0 \leq i < n-1} (\bigwedge_{a \notin \mathsf{opp}(\mathbf{c})} \neg e_{i,a,n-1})$.

P4 Reset transitions. Whenever a state $i \neq n-1$ reads the highest colour, then it reaches the initial state 0. Then, we have that $\bigwedge_{0 \leq i < n-1} e_{i,\mathbf{c}-1,0}$ holds.

P5 Sink state loops. The sink state $n-1$ makes exception to the previous rule, it can only loop on itself. Formally, $\bigwedge_{0 \leq a < \mathbf{c}} e_{n-1,a,n-1}$.

P6 No loops on opponent colours. No states but the sink can loop whenever reading a colour of the opponent parity as the highest one. Then, we have $\bigwedge_{0 \leq i < n-1} (\bigwedge_{a \in \mathsf{opp}(\mathbf{c})} \neg e_{i,a,i})$.

P7 Acceptance. If the highest colour is even, we look for a safety DFA (the sink state is the only rejecting one), otherwise we build a co-safety DFA (the only accepting state is the sink). Formally, if $\mathbf{c}-1$ is even, then $(\bigwedge_{0 \leq i < n-1} f_i) \wedge \neg f_{n-1}$, otherwise $(\bigwedge_{0 \leq i < n-1} \neg f_i) \wedge f_{n-1}$.

For best performance of parity game solving problems, it is better to turn on the safety encoding option when running DFAMiner.

---

6. This constraint may conflict with the requirement R4 for the representative DFA. So, we need to drop those constraints of the representative DFA for state $n-1$ that require the incoming transition to state $n-1$ to be over the maximal letter.