



DFAMiner: Mining Minimal Separating DFAs from Labelled Samples



Daniele Dell’Erba¹, Yong Li^{1,2}, and Sven Schewe¹



¹ Department of Computer Science, University of Liverpool, Liverpool, UK
 {Daniele.Dell-Erba,Yong.Li3,Sven.Schewe}@liverpool.ac.uk

² Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

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 SAT solving. 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: Passive learning · Separating Automata · Three-valued DFA · 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 [21], inference of network invariants [19], regular model checking [26], and reinforcement learning [24]. More recently, this problem has also arisen in the context of parity game solving [6], where separating automata can be used to decide the winner. The breakthrough quasi-polynomial algorithm [8], for example, can be viewed as producing such a separating automaton, and under additional constraints, quasi-polynomial lower bounds can be established,

too [8, 13]. 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 [5, 18]. Due to its high (NP-complete) complexity, researchers initially focused on either finding local optima through state merging techniques [7, 23, 29], 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 [31].

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 [20], FlexFringe [35], DFA-Inductor [34, 36], and DFA-Identify [24].

The current practical and exact solutions to the Min-DFA inference problem typically involve two steps: First, construct the augmented prefix tree acceptor (APTA [12]) that recognises the given samples, and then minimise the APTA to a Min-DFA by a reduction to SAT [20]. Recent enhancements of this approach focus on the second step, including techniques like symmetry breaking [20, 34] and compact SAT encoding [20, 36]. 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 [3]. However, their implementation relies heavily on MiniSAT [17]. 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 can be encoded as a Satisfiability Modulo Theories problem [32], which also benefits from 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 [7, 23, 29], 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 minimal acyclic DFA learning algorithm [14] to extract minimal DFAs from a given set of *positive* samples. More precisely, we extend their algorithm to support the APTA construction from a set of positive samples and a set of negative samples. Notably, the obtained APTA is guaranteed to be the *minimum-size deterministic* acceptor for the labelled sample set S .

We have implemented these techniques in our new tool DFAMiner and compared it with the state-of-the-art tools DFA-Inductor [34, 36] and DFA-Identify [24], on the benchmarks generated as described in [34, 36]. Our experimental results demonstrate that DFAMiner builds *smaller* APTAs and is therefore significantly faster at finding the Min-DFAs than both DFA-Inductor and DFA-Identify.

To test our technique, we have 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. 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.

Related work. The learned Min-DFA can be seen as a witness proof that separates the set of good behaviours and the set of 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 [26] and reinforcement learning [24]. Another standard application is in the active learning of minimal DFAs by equivalence queries [2]. We remark that in [1], non-incremental and incremental constructions were proposed to find small and even minimal APTAs that separate the positive and negative samples. These two constructions are based on state merging techniques of RPNI [29]. Their algorithms are approximate constructions. As a consequence, their constructed APTAs can be smaller (or even larger) than our APTAs, and can no longer be used to extract the minimal separating DFA for S in the second step.

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. As usual, we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A subset of Σ^* is a *finitary language*. Given a word u , we denote by $u[i]$ the i -th letter of u . We denote by $u[i, k]$ the subword starting at the i -th element and ending at the $(k - 1)$ -th element when $0 \leq i < k$, and the empty sequence ε when $i \geq k$ or $k = 0$. We denote by $u[i \cdots]$ the word of u starting at the i -th element when $i < |u|$, and the empty sequence ε when $i \geq |u|$. For two given words u and v , we denote by $u \cdot v$ (uv , for short) the concatenation of u and v . We say that u is a *prefix* of w if $w = u \cdot v$ for some word $v \in \Sigma^*$. We denote by $\text{prefixes}(u)$ the set of the prefixes of u . We also extend function prefixes to a set of words S , i.e. we have $\text{prefixes}(S) = \bigcup_{u \in S} \text{prefixes}(u)$.

Transition system. A *deterministic* transition system (TS) is a tuple $\mathcal{T} = (Q, \iota, \delta)$, where Q is a finite set of states, $\iota \in Q$ is initial state, and $\delta : Q \times \Sigma \rightarrow Q$ is a transition function. We also extend δ from letters to words in a usual way, by letting $\delta(q, \varepsilon) = q$ and $\delta(q, a \cdot u) = \delta(\delta(q, a), u)$, where $u \in \Sigma^*$ and $a \in \Sigma$.

Automata. An automaton on finite words is called a deterministic *finite automaton* (DFA). A DFA $\mathcal{A}$ is formally defined as a tuple $(\mathcal{T}, F)$, where $\mathcal{T}$ is a TS, and $F \subseteq Q$ is the set of *accepting* states. DFAs map all words in Σ^* to two values, accepting (+) and rejecting (−).

A *run* of an DFA $\mathcal{A}$ on a finite word u of length $n \geq 0$ is a sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \leq i < n$, $q_{i+1} = \delta(q_i, u[i + 1])$. We

write $q_0 \xrightarrow{u} q_n$ if there is a run from q_0 to q_n over u . DFAs have at most one run for each word. A run is *accepting* if it ends in an accepting state $q_n \in F$. A finite word $u \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if it has an accepting run. The set of words accepted by an automaton is called its *language*. The class of words *accepted* by DFAs is known to be regular languages. For a given regular language, the Myhill-Nerode theorem [25, 28] helps to obtain the minimal DFA.

DFAs are easy to extend to languages with “don’t-care” words.

Definition 1. A 3-valued DFA (3DFA)¹ is defined as a triple $(\mathcal{T}, A, R)$, where $\mathcal{T}$ is a deterministic TS, and A, R , and $D = Q \setminus (A \cup R)$ partition the set of states Q , where $A \subseteq Q$ is the set of accepting states; $R \subseteq Q$ is the set of rejecting states; and the remaining states D are called don’t-care states.

3DFAs map all words in Σ^* to *three* values: accepting (+), rejecting (−), and don’t-care (?), where they are accepting if they have an accepting run, rejecting if they have a rejecting run (which is a run ending in a rejecting state), and don’t-care otherwise.

It is possible to identify equivalent words that reach the same state in the minimal 3DFA of a given function $L : \Sigma^* \rightarrow \{+, -, ?\}$ [9]. Let x, y be two words in Σ^* and $L \in (\Sigma^* \rightarrow \{+, -, ?\})$ be a function. We define an equivalence relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ as: $x \sim_L y$ 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 $S = (S^+, S^-)$ be a given finite set of labelled samples in Σ^* . We can also see S as a classification function that induces an equivalence relation $\sim_S$. That is, if we set $S^? = \Sigma^* \setminus S$, then $S(u) = \$$ if $u \in S^{\$}$, where $\$ \in \{+, -, ?\}$. Finally, we conclude with a straightforward proposition that follows from the fact that $|\sim_S|$ is bounded by $|\text{prefixes}(S)|$.

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

3 DFAMiner

3.1 Main Problem

Let $S = (S^+, S^-)$ be the given set of labelled samples in the whole paper. 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 $S(u) = \$$, then $\mathcal{D}(u) = \$$, where $\$ \in \{+, -\}$. We call the target DFA a minimal *separating* DFA² for S , abbreviated as separating Min-DFA.

Recall that the passive learners for separating Min-DFAs [20, 36] usually first construct the APTA $\mathcal{P}$ (and thus a 3DFA) recognising S and then minimise the APTA $\mathcal{P}$ to a Min-DFA using a SAT solver. Our tool DFAMiner follows a similar workflow. The main advantage of DFAMiner compared to prior work is that it has access to an *incremental* construction that produces the minimal 3DFA $\mathcal{M}$

¹ 3DFAs are a standard model for representing positive and negative samples in the literature. In [1], 3DFAs are called deterministic unbiased finite state automata.

² The 3DFA that recognises S is called separating DFA for S in [9].

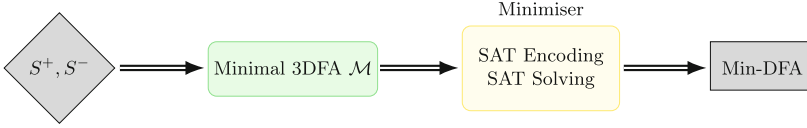


Fig. 1. Workflow of DFAMiner with 3DFAs

of S with respect to $\sim_S$. Furthermore, DFAMiner also supports the use of a DFA pair $(\mathcal{D}^+, \mathcal{D}^-)$ to obtain possibly further reduction on the state space. We call such pair double DFAs.

Definition 2. A double DFA (*dDFA*) is a tuple $(\mathcal{T} = \{\mathcal{T}^+, \mathcal{T}^-\}, A, R)$, where $\mathcal{T}$ is the union of two disjoint TSs, and A, R and $D = Q \setminus (A \cup R)$ partition the states Q of $\mathcal{T}$, such that the languages of $L^+ = (\mathcal{T}^+, A)$ and $L^- = (\mathcal{T}^-, R)$ are disjoint. We call the words accepted by L^+ accepting, the words accepted by L^- rejecting, and all other words don’t-care words.

Note that since L^+ and L^- are disjoint, every word on $\mathcal{T}$ can have only one accepting run and one rejecting run, although $\mathcal{T}$ has two initial states.

3.2 Workflow Description

Assume that we have an incremental construction of 3DFAs from the given set of samples $S = (S^+, S^-)$. A natural workflow of DFAMiner is to first construct the *minimal* 3DFA $\mathcal{M}$ (which is also a directed acyclic graph) from S and then minimise it using a SAT solver. This approach is depicted in Fig. 1. The components labelled in green or blue in Figs. 1–2 are novel contributions made in our tool. We use the standard SAT-based minimisation approaches of 3DFAs as a black-box [34].

We observe that the minimisation algorithm [34] does not necessarily work only on 3DFAs, but also on dDFAs and even on a pair of nondeterministic finite automata (the encoding will be discussed in Sect. 5). This motivates us to ask the following question: can we construct a dDFA for the pair of samples S ? We give a positive answer to this question.

Our construction of dDFAs $\mathcal{N}$ from S is formalised as follows. We construct the minimal 3DFAs $\mathcal{D}^+$ and $\mathcal{D}^-$ that recognise the languages $(S^+, \emptyset)$ and $(S^-, \emptyset)$, respectively, making sure that $\mathcal{D}^+$ and $\mathcal{D}^-$ do not share the same state names. We then combine the two DFAs into a dDFA $\mathcal{N}$, where the initial states of $\mathcal{N}$ are the initial states of both $\mathcal{D}^+$ and $\mathcal{D}^-$, while the transitions between states remain unchanged and we make the accepting states of $\mathcal{D}^+$ and $\mathcal{D}^-$ the accepting and rejecting states of $\mathcal{N}$, respectively. All other states are don’t-care states. Note that, although such a dDFA corresponds to two TSs, we can see them as one, since their languages are disjoint. Therefore, even if there are now two initial states, every word will be accepted or rejected by only one of them. The workflow of this construction is depicted in Fig. 2. In this way, we obtain a dDFA $\mathcal{N}$ that recognises exactly the given set S . The empirical evaluation shows

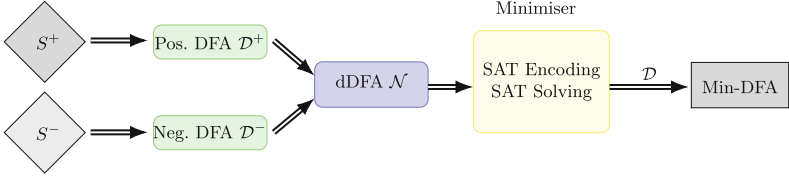


Fig. 2. Workflow of DFAMiner with dDFAs

that the two types of workflows are incomparable (none of them dominates the other in terms of size or speed), hence, both have their place in the learning procedure.

We note that the algorithm for producing dDFAs can be adjusted to produce proper double nondeterministic finite automata (NFA) when we first translate $\mathcal{D}^+$ and $\mathcal{D}^-$ to NFAs N^+ and N^- , respectively, using standard tools (e.g. [10]) to reduce their size (similar to Fig. 2). The way N^+ and N^- are merged into $\mathcal{N}$ is the same as for $\mathcal{D}^+$ and $\mathcal{D}^-$, and adjusting the SAT encoding to have NFAs (and thus potentially many successors) is straight forward.

For details of the components of DFAMiner, our incremental construction for 3DFAs is reported in Sect. 4, while the SAT-based minimisation algorithm is described in Sect. 5. In Sect. 6 we propose a possible application of DFAMiner in learning minimal separating DFAs by equivalence queries and to parity game solving. We close with an experimental evaluation on standard benchmarks in Sect. 7. A full version of the paper with supplementary materials can be found in [16].

4 Incremental Construction of 3DFAs

4.1 Prior Construction of 3DFAs

Let S be the given labelled sample set and $\mathcal{P}$ the APTA³ that recognises S constructed with standard procedures [20, 24, 34, 36]. The APTA $\mathcal{P} = (Q, \varepsilon, \delta, F, R)$ is formally defined as a 3DFA where $Q = \text{prefixes}(S)$ 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 $a \in \Sigma$.

Table 1. Size of Min-3DFA and APTA on parity game solving.

$ \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

³ APTAs are called prefix tree unbiased finite state automata in [1] and they are also similar to the prefix-tree Moore Machines in [33].

The main issue is that the size of $\mathcal{P}$ increases dramatically with the growth of the number of samples in S and their length. This is not surprising given that $\mathcal{P}$ maps every word in $\text{prefixes}(S)$ to a unique state.

To show this growth, we have 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 states in APTAs that can be merged. To identify equivalent states in $\mathcal{P}$, we can use the equivalence relation $\sim_S$. In fact, since APTAs are acyclic, we can minimise them via a linear-time backward traversal [14]. Further, we show next that we do not have to construct the full APTA $\mathcal{P}$ in order to obtain the Min-3DFA for the given samples.

We will subsequently refer to APTAs constructed by the existing approaches and use 3DFAs for the acceptors constructed by our new technique.

4.2 Incremental Construction of 3DFAs

In [14], an incremental construction of a minimal DFA that accepts a given set of positive samples has been proposed. We extend their algorithm to 3DFAs from a pair $S = (S^+, S^-)$ of sets of labelled samples.

Our algorithm can be seen as the *on-the-fly* version of the combination of the construction of the APTA and its minimisation to the Min-3DFA based on the backward traversal of the APTA. We first describe the minimisation of the APTA tree and then the on-the-fly construction of the Min-3DFAs in the sequel.

For simplicity, let us assume that the *full* APTA tree $\mathcal{P}$ is already given. The crucial step in the minimisation component is to decide whether two states p and q are equivalent. Based on the definition of $\sim_S$, we define that two states $p, q \in Q$ are equivalent, denoted $p \equiv q$ if, and only if:

1. they have the same acceptance status, i.e. they are both accepting, rejecting or don’t-care states; and
2. for each letter $a \in \Sigma$, they either 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 simplified as follows:

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

Therefore, it is easy to outline an algorithm to minimise the given APTA tree $\mathcal{P}$ by applying these steps:

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 *Register*, which allows fast access to their representative states for all 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 one, are in *Register*.

In this way, we are guaranteed to obtain the Min-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 outlined 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 Min-3DFAs. Hence, it is vital to avoid the full construction of the APTA tree $\mathcal{P}$ of S . The key of the *on-the-fly* construction is to identify when a state has been *completely* traversed during construction.

To this end, we need to assume that the samples are already ordered in the usual *lexicographical* order that we will also use to compare the words. That is, the input samples will be first ordered as follows. For two words u and u' , we first compare their prefixes of length $\min(|u|, |u'|)$. Then, three cases may arise: one of the two words has a smaller letter than the other at the same position, then that word is smaller; otherwise the two prefixes coincide, and then if the two words have the same length, u and u' are equal; otherwise one word is longer than the other, then it is greater.

Assume that $S = \{u_1, u_2, \dots, u_\ell\}$ is ordered. In the process of the creation of the states, we need to detect when a state *cannot* have further successors and then it is ready to be merged with its representative state. Assume that the current 3DFA is $\mathcal{P}_i = (Q_i, \{\iota\}, \delta_i, F_i, R_i)$ and we now input the next sample u_{i+1} . When $i = 0$, $\mathcal{P}_0$ is trivially minimal since $\mathcal{P}_0$ has only a state ι without any outgoing transition. For technical reasons, 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).

Assume now that $i \geq 0$. We read u_{i+1} and run it on $\mathcal{P}_i$. The sample can be seen as $u_{i+1} = x \cdot y_{i+1}$ with the assumption that $x \in \text{prefixes}(u_{i+1})$ is 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 cannot be merged with their representatives, as $\mathcal{P}_i$ requires new states to run the suffix y_{i+1} . Note that x must be a prefix of u_i too, i.e. $x \in \text{prefixes}(u_i)$. This follows from the fact that there must be a run of $\mathcal{P}_i$ over u_i , which is the greatest sample in lexicographic order so far, and every word that has a *complete* run in $\mathcal{P}_i$ must *not* be greater than u_i . In fact, if we assume that x is not a prefix of u_i , then x must be smaller than $u_i[0, |x|]$. This leads to the contradiction that u_i is greater than u_{i+1} . Hence, in this case x must be the empty string. 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 states p_k with $k > |x|$ in the run of $\mathcal{P}_i$ over u_i can be merged with their representative, as they cannot have more (future) reachable states.

If we instead assume that there is a state p_j with $j > |x|$ reached over a future sample u_h with $h > i$, then u_h is smaller than u_{i+1} , which leads to the contradiction that the samples are ordered from the smaller to the bigger. Thus,

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
     $x := \text{common\_prefix}(u)$ 
     $p := \delta(\iota, x)$  ▷ the last state over the common prefix  $x$ 
     $y := u[|x| \dots]$  ▷ the remaining suffix of  $u$ 
    if  $\text{has\_children}(p)$  then
       $\text{replace\_or\_register}(p)$  ▷ merge/register all states after  $p$ 
    end if
     $\text{add\_suffix}(p, y)$  ▷ create run to accept suffix  $y$  from  $p$ 
  end while
   $\text{replace\_or\_register}(\iota)$  ▷ merge the run over the last sample
end procedure

procedure REPLACE_OR_REGISTER( $p$ )
   $r := \text{max\_child}(p)$  ▷ obtain the successor over the maximal letter
  if  $\text{has\_children}(r)$  then ▷  $r$  has a successor
     $\text{replace\_or\_register}(r)$  ▷ recursively obtain the run over last sample
  end if
  if  $\exists q \in Q. (q \in Register \wedge q \equiv r)$  then
     $\text{max\_child}(p) := q$  ▷ merge with its representative
  else
     $Register := Register \cup \{r\}$  ▷ set the 1st state of each class as representative
  end if
end procedure

```

we can identify the representatives for states p_k and merge them in the usual backward manner. It follows that all 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 samples, we only need to merge all states in the run over the last sample u_ℓ with their equivalent states. This way, we are guaranteed to obtain the Min-3DFA $\mathcal{M}$ for S in the end.

The formal procedure of the above incremental construction of the Min-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 `max_child` function. In this way, when we reach the last state r of the run over the last sample (i.e., `has_children( $r$ )` is false), we can begin to identify equivalent states and replace the successor of p with their representative state q in a backward manner or set the state r as the representative of its equivalent class, 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 the equivalence relation $\equiv$ in `replace_or_register` [14] to support the accepting, rejecting, and don’t-care states, as described before.

Figure 3 depicts all intermediate 3DFAs when running Algorithm 1 on the ordered set $S = \{(000, +), (001, +), (10, -)\}$. Initially, the 3DFA only has the initial state ι without outgoing transitions and *Register* is empty. The algorithm first creates states to accept 000. After receiving sample 001, the algorithm runs the common prefix 00 and merges r with its equivalent states. So, r is added to *Register*. When the sample 10 is read, the common prefix with 001 is ε , then all states after ι in the run over 001 can be merged with their equivalent

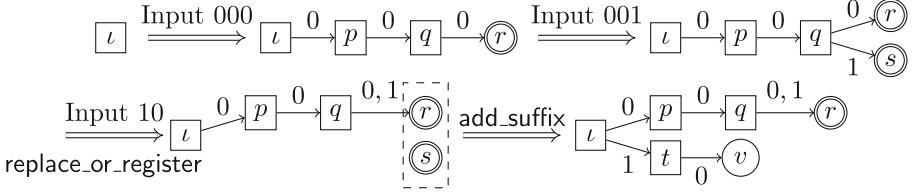


Fig. 3. An example run over $S = \{(000, +), (001, +), (10, -)\}$. Accepting, rejecting and don't-care states are denoted, respectively, by double circles, circles and squares. The dashed rectangle depicts an equivalence class.

states in `replace_or_register` function. The merge will perform in a backward manner, starting from the last state s until the state p . So, now that we know that also s will not have more successors (because the samples are ordered) we can consider it as *complete* and therefore merge with r . As a consequence, as shown in Fig. 3, all incoming transitions of s are redirected to its representative r and s is deleted. So, $Register = \{r, p, q\}$. After this step, `add_suffix` will create the new states t and v . Following Algorithm 1, we will eventually obtain the last 3DFA in Fig. 3 as the final result. Note that the biggest intermediate 3DFA constructed by Algorithm 1 is usually much smaller than the full APTA.

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

The proof is basically an induction on the number of input samples and merely extends the intuition described above; we thus omit it here.

DFA construction. The proposed construction of Min-3DFAs is more general than the state-of-the-art incremental one [14] in which it is only checked whether p and q are both accepting or rejecting states when defining the equivalence relation $\equiv$. The other parts of the construction can be modified accordingly.

5 Finding Separating Min-DFAs Using SAT Solvers

This section explains how to extract the separating Min-DFAs from dDFAs⁴ built from the 3DFAs through our incremental construction in Sect. 4.2. The encoding approach is used in the Minimiser for both workflows in Figs. 1 and 2 and is agnostic to the SAT solver used. Since minimising DFAs with don't-care words is known to be NP-complete [30], it is unlikely to have polynomial-time *exact* algorithm for the second step unless $\mathbf{P} = \mathbf{NP}$.

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

⁴ 3DFAs can be seen as a special type of dDFAs.

$u \in \Sigma^*$, 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. $0 < n \leq |Q|$, 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 [26, 27, 34, 36]. For simplicity, we let $\{0, \dots, n-1\}$ 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 $i \xrightarrow{a} j$ 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 resulting DFA is indeed deterministic and complete, we need following constraints:

D1 Determinism:

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

D2 Completeness: For every state i and letter $a \in \Sigma$ in $\mathcal{D}$, $\bigvee_{0 \leq j < n} e_{i,a,j}$ 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, Theorem 2 follows.

Theorem 2. *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 $\mathcal{D}_n$ with n states that is separating for $\mathcal{N}$.*

Let n be the minimal integer such that $\phi_n^{\mathcal{N}}$ is satisfiable. Then $\mathcal{D}_n$ is a separating Min-DFA for the sample set S .

The formula $\phi_n^{\mathcal{N}}$ contains $(n^3 \cdot |\Sigma| + n^2 \cdot |Q| \cdot |\Sigma|)$ 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, DFAMiner uses the technique in [34] to check only a representative DFA for all isomorphic DFAs.

6 Applications

Apart from those mentioned in the introduction, in this section we describe two new applications.

6.1 Active Learning of Separating Min-DFAs

Our tool can be applied to the active learning of minimal DFAs using only equivalence queries (EQs). In fact, minimal DFAs cannot be exactly learned using polynomial number of EQs in the size of the target Min-DFA [2].

While learning, we maintain a growing set of pairs $S_i = (S_i^+, S_i^-)$ to store the positive and negative words. That is, for all the indexes i , $S_{i+1}^+ \supseteq S_i^+$, $S_{i+1}^- \supseteq S_i^-$, and $S_{i+1} \neq S_i$. For each $i > 0$, DFAMiner finds a Min-DFA $\mathcal{D}_i$ for S_i and ask an EQ that the teacher returns a *yes* if $\mathcal{D}_i$ accepts all positive and rejects all negative words in the target language L , or provides one (*positive* or *negative*) counterexample (CEX) u otherwise. We can start with $S_0 = (\emptyset, \emptyset)$ and propose a DFA $\mathcal{D}_0$ accepting nothing, and then obtain S_{i+1} from S_i by adding the CEX u to either S_i^+ or S_i^- . If $\mathcal{D}_i(u) = +$, then u is a negative word since u is misclassified by $\mathcal{D}_i$ and should be added into S_{i+1}^- ; otherwise, we add u into S_{i+1}^+ . (The other set will remain the same.) Since $\mathcal{D}_i$ is consistent with S_i but not with all S_j with $j > i > 0$, all $\mathcal{D}_i$ are smaller or equal in size to the target DFA $\mathcal{D}$. For some $k > 0$, S_k will uniquely characterise L , $\mathcal{D}_k$ will accept L , where k can be exponential in the number of states in $\mathcal{D}_k$ in the worst case.

6.2 Learning Separating Automata for Parity Games

A few years ago an algorithm to solve parity games in quasi-polynomial time [8] has been proposed. It has then been shown that the underlying approach essentially builds a separating automaton of quasi-polynomial size to distinguish runs with only winning cycles (according to the parity condition) from the losing ones [13]. Such a separating automaton distinguishes the two disjointed languages composed of the set of infinite words that correspond to paths on the graph where the highest colour occurring is *even* (hence, winning), or *odd* (losing). Where each colour occurs only once, a cycle occurs when a colour has repeated at least twice. For instance, the word $(001212)^\omega$ contains only even cycles including 00,121 and 212, while the word $(1312331)^\omega$ contains only odd cycles such as 131, 3123, and 33. Given a parity game $\mathcal{G}$, and a separating automaton $\mathcal{S}$ that accepts only even cycles and rejects odd cycles, solving the parity game $\mathcal{G}$ can be reduced to solving the safety game $\mathcal{G} \otimes \mathcal{S}$ [6]. Although the product game is much bigger than $\mathcal{G}$, safety games are easier to solve than parity games. Moreover, the constructed separating automaton $\mathcal{S}$ is quasi-polynomial in the number of colours, which gives an upper bound for solving parity games.

These separating safety automata work on infinite words, but we will employ our tool to learn them by using finite-length samples. This is because as long as the length of the finite sample words is long enough, the learned DFAs will converge to the correct safety automata. The hardest case for the separation

approach [6] occurs when the colours are unique (occur only once, hence, the colour itself can be used as a node identifier, making detection of cycles easier). We have implemented this case as follows: we fix an alphabet with $\mathbf{c}$ different colours, a length $\ell > \mathbf{c}$, (to ensure that each word contains at least one cycle), and $\mathbf{c}$ as highest colour. In the learned DFA, we must accept a word if all cycles are winning (e.g. 001212) and reject it if all cycles are losing (e.g. 13123312). Words with winning and losing cycles (e.g. 21232) are don’t-care words.

The resulting automata are always safety automata 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 [6].

Separating automata built with the current state-of-the-art construction [8] 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 2 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 positive and negative samples generated. Although most words have both winning 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. 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. These supplementary data are provided in [16]. 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.

Table 2. Samples required to learn the minimal separating automata for solving parity games.

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

7 Evaluation

To further demonstrate the improvements of DFAMiner⁵ over the state of the art, we conducted comprehensive experiments on standard benchmarks [34, 36]. We compared with DFA-Inductor [36] and DFA-Identify⁶ [24], 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 [20] and the representative DFAs in the second step [34]. Like DFA-Inductor, DFAMiner is also implemented in Python with PySAT [22]. We delegate all SAT queries to the SAT solver CaDiCal 1.5.3 [4] in all tools.

DFAMiner accepts samples formalised in the Abbadingo⁷ format.

The experiments were carried on an Intel i7-4790 3.60 GHz processor. In Table 3, 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 1,200s. The alphabet for the samples has two symbols while the size of the generated DFA

Table 3. Comparison for the minimisation of DFAs from random samples of DFAMiner with DFA inductor.

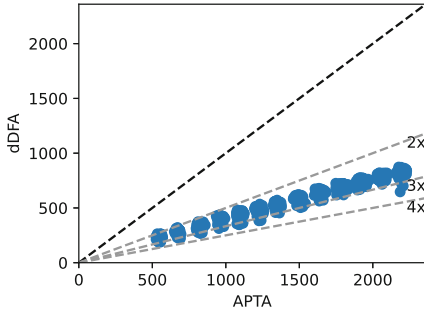
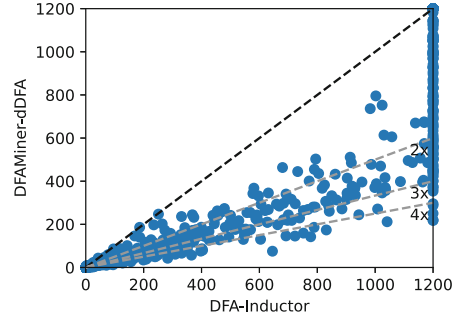
N	DFA-Inductor		DFA-Identify		dDFA-MIN		3DFA-MIN	
	avg	%	avg	%	avg	%	avg	%
4	0.12	100	0.09	100	0.03	100	0.02	100
5	0.29	100	1.38	100	0.06	100	0.05	100
6	0.67	100	2.33	100	0.30	100	0.18	100
7	1.81	100	4.12	100	0.80	100	0.73	100
8	3.57	100	9.70	100	1.29	100	1.25	100
9	10.84	100	20.76	100	3.83	100	3.78	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

is N . We compare four approaches to inferring Min-DFAs: DFA-Inductor, DFA-Identify, and DFAMiner with both 3DFA (3DFA-MIN) and dDFA (dDFA-MIN). 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 min 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 dDFA approach is slightly faster than the 3DFA one.

⁵ <https://github.com/liyong31/DFAMiner>.

⁶ <https://github.com/mvcisback/dfa-identify>.

⁷ <https://abbadingo.cs.nuim.ie/>.

**Fig. 4.** Scatter plot on automata size**Fig. 5.** Scatter plot on runtime (secs)

Figures 4 and 5 report 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 values, we show the individual data for each sample. Both DFA-Inductor and DFA-Identify build the same APTA (they differ for the encoding step), and as shown in Fig. 4, its size is three times larger than the dDFA built by DFAMiner, no matter how big the final DFA is. Figure 5, instead, shows that, when using a dDFA, DFAMiner always performs better than DFA-Inductor, on average three times faster with peaks of more than four times faster. The comparison between dDFA and DFA-Identify is similar.

The experimental results have confirmed that our construction of sample representations significantly advances the state-of-the-art, making it a valuable contribution to the Min-DFA inference problem. We note that DFA-Inductor 2 [36] is faster than DFA-Inductor due to a better encoding of the representative DFAs. Nonetheless, DFAMiner still performs significantly better than DFA-Inductor 2 regarding the overall number of solved cases and running time. For a fair comparison, we choose DFA-Inductor as the baseline, as DFAMiner only differs from it in the construction of APTAs. Additional comparisons on runtime and automata size with DFA-Inductor 2 can be found in [16].

8 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 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 [36]. Another easy extension of our construction is to learn a set of decomposed DFAs [24], 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.

Acknowledgments. We thank the anonymous reviewers for their valuable feedback. This work has been supported in part by the NSFC grant 62102407 and the EPSRC through grants EP/X021513/1, EP/X017796/1, and EP/X03688X/1.

Data Availability. The source code and data are available in [15].

References

1. Alquezar, R., Sanfeliu, A.: 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, vol.94, pp. 291–300 (1995)
2. Angluin, D.: Negative results for equivalence queries. *Mach. Learn.* **5**, 121–150 (1990). <https://doi.org/10.1007/BF00116034>
3. Avellaneda, F., Petrenko, A.: Learning minimal DFA: taking inspiration from RPNI to improve SAT approach. In: Ölveczky, P.C., Salaün, G. (eds.) Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings, LNCS, vol. 11724, pp. 243–256. Springer (2019). https://doi.org/10.1007/978-3-030-30446-1_13
4. Biere, A., Fazeekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froykys, 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)
5. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput.* **21**(6), 592–597 (1972). <https://doi.org/10.1109/TC.1972.5009015>
6. Bojańczyk, M., Czerwiński, W.: An Automata Toolbox. unpublished (2018)
7. Bugalho, M.M.F., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. *Pattern Recognit.* **38**(9), 1457–1467 (2005). <https://doi.org/10.1016/J.PATCOG.2004.03.027>
8. Calude, C., Jain, S., Khoussainov, B., Li, W., Stephan, F.: deciding parity games in quasipolynomial time. In: Symposium on Theory of Computing 17, pp. 252–263. Association for Computing Machinery (2017)
9. Chen, Y., Farzan, A., Clarke, E.M., Tsay, Y., Wang, B.: Learning minimal separating DFA’s for compositional verification. In: Kowalewski, S., Philippou, A. (eds.) 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, LNCS, vol. 5505, pp. 31–45. Springer (2009). https://doi.org/10.1007/978-3-642-00768-2_3
10. Clemente, L., Mayr, R.: Efficient reduction of nondeterministic automata with application to language inclusion testing. *Log. Methods Comput. Sci.* **15**(1) (2019). [https://doi.org/10.23638/LMCS-15\(1:2\)2019](https://doi.org/10.23638/LMCS-15(1:2)2019)
11. Coste, F., Nicolas, J.: Regular inference as a graph coloring problem. In: IWGI (1997)

12. Coste, F., Nicolas, J.: How considering incompatible state mergings may reduce the DFA induction search tree. In: Honavar, V.G., Slutzki, G. (eds.) *Grammatical Inference*, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1433, pp. 199–210. Springer (1998). <https://doi.org/10.1007/BFB0054076>
13. Czerwinski, W., Daviaud, L., Fijalkow, N., Jurdzinski, M., Lazic, R., Parys, P.: Universal trees grow inside separating automata: quasi-polynomial lower bounds for parity games. In: *Symposium on Discrete Algorithms 19*, pp. 2333–2349. SIAM (2019)
14. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite state automata. *Comput. Linguistics* **26**(1), 3–16 (2000). <https://doi.org/10.1162/089120100561601>
15. Dell’Erba, D., Li, Y., Schewe, S.: Artifact for DFAMiner: Mining Minimal Separating DFAs from Labelled Samples (Jun 2024). <https://doi.org/10.5281/zenodo.12528885>
16. Dell’Erba, D., Li, Y., Schewe, S.: DFAMiner: mining minimal separating DFAs from labelled samples (2024). <https://arxiv.org/abs/2405.18871>
17. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) *Theory and Applications of Satisfiability Testing*, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003. Selected Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003). https://doi.org/10.1007/978-3-540-24605-3_37
18. Gold, E.M.: Complexity of automaton identification from given data. *Inf. Control.* **37**(3), 302–320 (1978). [https://doi.org/10.1016/S0019-9958\(78\)90562-4](https://doi.org/10.1016/S0019-9958(78)90562-4)
19. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: Furbach, U., Shankar, N. (eds.) *Automated Reasoning*, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4130, pp. 483–497. Springer (2006). https://doi.org/10.1007/11814771_40
20. Heule, M., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) *Grammatical Inference: Theoretical Results and Applications*, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6339, pp. 66–79. Springer (2010). https://doi.org/10.1007/978-3-642-15488-1_7
21. de la Higuera, C.: A bibliographical study of grammatical inference. *Pattern Recognit.* **38**(9), 1332–1348 (2005). <https://doi.org/10.1016/J.PATCOG.2005.01.003>
22. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: a python toolkit for prototyping with SAT oracles. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) *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. Lecture Notes in Computer Science, vol. 10929, pp. 428–437. Springer (2018). https://doi.org/10.1007/978-3-319-94144-8_26
23. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) *Grammatical Inference*, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1433, pp. 1–12. Springer (1998). <https://doi.org/10.1007/BFB0054059>
24. Lauffer, N., Yalcinkaya, B., Vazquez-Chanlatte, M., Shah, A., Seshia, S.A.: Learning deterministic finite automata decompositions from examples and demonstrations. In: Griggio, A., Rungta, N. (eds.) *22nd Formal Methods in Computer-Aided*

- Design, FMCAD 2022, Trento, Italy, October 17-21, 2022, pp. 1–6. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_39
25. Myhill, J.: Finite automata and the representation of events. In: Technical Report WADD TR-57-624, pp. 112–137 (1957)
 26. Neider, D.: Computing minimal separating DFAs and regular invariants using SAT and SMT solvers. In: Chakraborty, S., Mukund, M. (eds.) *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings. Lecture Notes in Computer Science*, vol. 7561, pp. 354–369. Springer (2012). https://doi.org/10.1007/978-3-642-33386-6_28
 27. Neider, D., Jansen, N.: Regular model checking using solver technologies and automata learning. In: Brat, G., Rungta, N., Venet, A. (eds.) *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings. Lecture Notes in Computer Science*, vol. 7871, pp. 16–31. Springer (2013). https://doi.org/10.1007/978-3-642-38088-4_2
 28. Nerode, A.: Linear automaton transformations. *Am. Math. Soc.* **4** 541–544 (1958)
 29. Oncina, J., Garcia, P.: Inferring regular languages in polynomial updated time. In: *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*, pp. 49–61. World Scientific (1992)
 30. Pfleger, C.P.: State reduction in incompletely specified finite-state machines. *IEEE Trans. Computers* **22**(12), 1099–1102 (1973). <https://doi.org/10.1109/T-C.1973.223655>
 31. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. *J. ACM* **40**(1), 95–142 (1993). <https://doi.org/10.1145/138027.138042>
 32. Smetsers, R., Fiterau-Brostean, P., Vaandrager, F.W.: Model learning as a satisfiability modulo theories problem. In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) *Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings. Lecture Notes in Computer Science*, vol. 10792, pp. 182–194. Springer (2018). https://doi.org/10.1007/978-3-319-77313-1_14
 33. Trakhtenbrot, B.A., Barzdin, Y.M.: *Finite Automata: Behavior And Synthesis*. Elsevier (1973)
 34. Ulyantsev, V., Zakirzyanov, I., Shalyto, A.: BFS-based symmetry breaking predicates for DFA identification. In: Dediu, A., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings. Lecture Notes in Computer Science*, vol. 8977, pp. 611–622. Springer (2015). https://doi.org/10.1007/978-3-319-15579-1_48
 35. Verwer, S., Hammerschmidt, C.A.: flexfringe: a passive automaton learning package. In: *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*, pp. 638–642. IEEE Computer Society (2017). <https://doi.org/10.1109/ICSME.2017.58>
 36. Zakirzyanov, I., Morgado, A., Ignatiev, A., Ulyantsev, V., Marques-Silva, J.: Efficient symmetry breaking for sat-based minimum DFA inference. In: Martín-Vide, C., Okhotin, A., Shapira, D. (eds.) *Language and Automata Theory and Applications - 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings. Lecture Notes in Computer Science*, vol. 11417, pp. 159–173. Springer (2019). https://doi.org/10.1007/978-3-030-13435-8_12

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

