Exact Searching for the Smallest Deterministic Automaton

Wojciech Wieczorek[1][0000−0003−3191−9151], Łukasz Strąk[2][0000−0002−1074−2847], Arkadiusz Nowakowski[2][0000−0001−8304−5746], and Olgierd Unold[3][0000−0003−4722−176X]

[1] University of Bielsko-Biala, Willowa 2, 43-309 Bielsko-Biala, Poland
wwieczorek@ath.bielsko.pl
[2] University of Silesia in Katowice, Bankowa 14, 40-007 Katowice, Poland
{lukasz.strak,arkadiusz.nowakowski}@us.edu.pl
[3] Wroclaw University of Science and Technology, Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland olgierd.unold@pwr.edu.pl

Abstract. We propose an approach to minimum-state deterministic finite automaton (DFA) inductive synthesis that is based on using satisfiability modulo theories (SMT) solvers. To that end, we explain how DFAs and their response to input samples can be encoded as logic formulas with integer variables, equations, and uninterpreted functions. An SMT solver is then tasked with finding an assignment for such a formula, from which we can extract the automaton of a required size. We provide an implementation of this approach, which we use to conduct experiments on a series of benchmarks. The results showed that our method outperforms in terms of CPU time other SAT and SMT approaches and other exact algorithms on prepared benchmarks.

Keywords: Grammatical inference · Automata identification · Satisfiability modulo theories · Exact search.

1 Introduction

In his notable paper [7] Gold proved that the following problem is NP-complete. INSTANCE: Finite alphabet Σ, two finite subsets $S_+, S_- \subseteq \Sigma^*$, integer $K > 0$. QUESTION: Is there a K-state deterministic finite automaton (DFA) A that recognizes a language $L \subseteq \Sigma^*$ such that $S_+ \subseteq L$ and $S_- \subseteq \Sigma^* - L$?
This problem is important both from theoretical and practical points of view. In the theory of grammatical inference [11], we can pose interesting questions. For instance: What if instead of a finite state automaton, a regular expression [1] or a context-free grammar is required [19, 13]? What if we are given all words up to a certain length [23]? What if we are given infinitely many words [8]? The problem of finding the smallest deterministic automaton (the optimization version of the above given instance) is also crucial in practice, since searching for a small acceptor compatible with examples and counter-examples is generally a good idea in grammatical inference applications [25].

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

The purpose of the present proposal is threefold. The first objective is to devise an algorithm for the smallest deterministic automaton problem. It entails preparing satisfiability modulo theories (SMT) logical formula before starting the searching process. The second objective is to implement this encoding by means of an available SMT solver to get our approach working. The third objective is to investigate to what extent the power of SMT solvers makes it possible to tackle the regular inference problem for large-size instances and to compare our approach with existing ones. Particularly, we will refer to the following classical and new exact DFA identification methods: Bica [17], Exbar [15], Zakirzyanov et al's SAT encoding [26], and Smetsers et al's SMT encoding [21]. In view of the possibility of future comparisons with other methods, the Python implementation of our method is given via GitHub.[4]

This paper is organized into five sections. In section 2, we present necessary definitions and facts originated from automata, formal languages, and constraint programming. Section 3 describes our inference algorithm based on solving an SMT formula. Section 4 shows experimental results of our approach. Concluding comments are made in Section 5.

2 Preliminaries

We assume the reader to be familiar with basic regular language and automata theory, e.g., from Hopcroft et al. textbook [12], so that we introduce only some notations and notions used later in the paper.

2.1 Words and Languages

An *alphabet* is a finite, non-empty set of symbols. We use the symbol Σ for an alphabet. A *word* is a finite sequence of symbols chosen from an alphabet. For a word w, we denote by $|w|$ the length of w. The *empty word* ε is the word with zero occurrences of symbols. Let x and y be words. Then xy denotes the *catenation* of x and y, that is, the word formed by making a copy of x and following it by a copy of y. As usual, Σ^* denotes the set of words over Σ. A word w is called a *prefix* of a word u if there is a word x such that $u = wx$. It is a *proper* prefix if $x \neq \varepsilon$. A set of words all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a *language*.

2.2 Deterministic Finite Automata

A *deterministic finite automaton* (DFA) is a five-tuple $A = (\Sigma, Q, s, F, \delta)$ where Σ is an alphabet, Q is a finite set of states, $s \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and δ is a relation from $Q \times \Sigma$ to Q such that $((q, a), r_1) \in \delta$ and $((q, a), r_2) \in \delta$ implies $r_1 = r_2$ for every pair $(q, a) \in Q \times \Sigma$.

Members of δ are called *transitions*. A transition $((q, a), r) \in \delta$ with $q, r \in Q$ and $a \in \Sigma$, is usually written as $\delta(q, a) = r$. Relation δ specifies the moves: the

[4] `https://github.com/wieczorekw/wieczorekw.github.io/tree/master/SMT4DFA`

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

meaning of $\delta(q,a) = r$ is that automaton A in the current state q reads a and moves to next state r. If for given q and a there is no such r that $((q,a),r) \in \delta$, simply saying $\delta(q,a)$ is undefined, the automaton stops and we can assume it enters the rejecting state. Moving into a state that is not final is also regarded as rejecting but it may be only an intermediate state.

It is convenient to define $\bar{\delta}$ as a relation from $Q \times \Sigma^*$ to Q by the following recursion: $((q,ya),r) \in \bar{\delta}$ if $((q,y),p) \in \bar{\delta}$ and $((p,a),r) \in \delta$, where $a \in \Sigma$, $y \in \Sigma^*$, and requiring $((t,\varepsilon),t) \in \bar{\delta}$ for every state $t \in Q$. The *language accepted* by automaton A is then

$$L(A) = \{x \in \Sigma^* \mid \text{there is } q \in F \text{ such that } ((s,x),q) \in \bar{\delta}\}.$$

Two automata are *equivalent* if they accept the same language.

A *sample* S will be an ordered pair $S = (S_+, S_-)$ where S_+, S_- are finite languages with an empty intersection (have no common word). S_+ will be called the *positive part of* S (*examples*), and S_- the *negative part of* S (*counter-examples*).

Let S be a sample over an alphabet Σ, P be the set of all prefixes of $S_+ \cup S_-$, and let $A = (\Sigma, Q, s, F, \delta)$ be a DFA. Let $f\colon P \to Q$ be bijective. For simplicity of notation, we will write q_p instead of $f(p) = q$ for $q \in Q$ and $p \in P$. An automaton A is an *augmented prefix tree acceptor* (APTA) if:

- $s = q_\varepsilon$,
- $F = \{q_p \mid p \in S_+\}$,
- $\delta = \{((q_p,a),q_r) \mid p,r \in P \text{ such that } r = pa\}$,
- $\{F,R,N\}$ is a partition of Q, where F is the set of final states, $R = \{q_p \mid p \in S_-\}$ is the set of rejecting states, and $N = Q - (F \cup R)$ is the set of neutral states.

Clearly, $L(A) = S_+$ and transitions (as arcs) along with states (as vertices) form a tree with the q_ε root, whose edges are labelled by symbols taken from an alphabet Σ (see Figure 1 as an example).

From now on, if the states of a DFA $A = (\Sigma, Q, s, F, \delta)$ have been partitioned into final, F, rejecting, R, and neutral, N, states, then we will say that $x \in \Sigma^*$ is: (a) *recognized by accepting* (or simply *accepted*) if there is $q \in F$ such that $((s,x),q) \in \bar{\delta}$, (b) *recognized by rejecting* if there is $q \in R$ such that $((s,x),q) \in \bar{\delta}$, and (c) *rejected* if it is not accepted. So, when we pass an automaton according with consecutive symbols (the letters of a word) and stop after reading the last symbol at a neutral state, then the word is rejected but is not recognized by rejecting.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

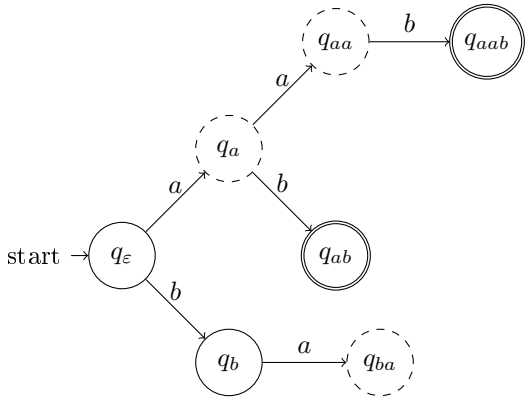

Fig. 1. An APTA accepting ab, aab and 'rejecting' a, aa, ba. The set of its states, Q, is partitioned into $F = \{q_{ab}, q_{aab}\}$, $R = \{q_a, q_{aa}, q_{ba}\}$, $N = \{q_\varepsilon, q_b\}$.

Many exact and inexact DFA learning algorithms work starting from building an APTA and then folding it up into a smaller hypothesis by merging various pairs of compatible nodes. The merging operation takes two states, q_1 and q_2, and replaces them with a single state, q_3, in such a way that every state incoming to q_1 or q_2 now incomes to q_3 and every state outcoming from q_1 or q_2 now outcomes from q_3. It should be noted that the effect of the merge is that a DFA will possibly lose the determinism property through this. Therefore, not all merges will be admitted.

2.3 Satisfiability Modulo Theories

In computer science and mathematical logic, the satisfiability modulo theories (SMT) problem is a decision problem for logical formulas with respect to combinations of background theories—possibly other than Boolean algebra—expressed in classical first-order logic with equality [14]. Examples of theories typically used in computer science are the theory of real numbers, the theory of integers, the theory of uninterpreted functions, and the theories of various data structures such as lists, arrays, bit vectors and so on. Therefore, SMT can be thought of as a form of the constraint satisfaction problem.

Translating a problem into an SMT formula is often the first choice when we are dealing with applications in fields that require determining the satisfiability of formulas in more expressive logics than SAT. The concrete syntax of the SMT standard is very complex. The description of SMT-LIB language [2] is a good reference as a basic background. There are a lot of research publications on the construction of SMT solvers. The reader is referred to an introductory textbook on the topic [14]. Decision procedures for logical formulas with respect to equations and uninterpreted functions use the results of Tarjan [22] and Dawney et al. [5]. Linear arithmetic, on the other hand, may be solved with recipes given by

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

Dutertre and Moura [6]. In our implementation, Z3 library [16] was used, which is released under MIT License and available through a web page.[5]

3 Proposed Encoding for the Induction of Deterministic Automata

Our translation reduces DFA identification into an SMT instance. Suppose we are given a sample S with nonempty S_+ and nonempty S_- over an alphabet Σ, and a nonnegative integer K. We want to find a $(K+1)$-state DFA $A = (\Sigma, \{q_0, q_1, \ldots, q_K\}, s, F, \delta)$ such that every $w \in S_+$ is recognized by accepting and every $w \in S_-$ is recognized by rejecting.

3.1 Direct Encoding

Let P be the set of all prefixes of $S_+ \cup S_-$. The integer variables will be $x_p \in \mathbb{Z}$. Assume further that $\{f_a\}_{a\in\Sigma}$ is a family of uninterpreted functions $\mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$ indexed by Σ and $g\colon \mathbb{Z} \to \mathbb{B}$ is also an uninterpreted function. The interpretation of these variables and functions is as follows. The value of x_p is the index of a state which is reached after passing from the initial state, s, through transitions determined by consecutive symbols of a prefix p in a resultant automaton A. The value of $f_a(i,j)$ is $\top$ if $((q_i, a), q_j) \in \delta$, $f_a(i,j) = \bot$ otherwise. Finally, we let $g(i) = \top$ if $q_i \in F$ and $\bot$ if not.

Let us now see how to describe the constraints of the relationship between an automaton A and a sample S in terms of SMT.

1. Naturally, the number of states is already fixed up
$$0 \le x_p \le K, \quad \text{for } p \in P.$$
2. Every example has to be recognized by accepting
$$g(x_p) = \top, \quad \text{for } p \in S_+.$$
3. Every counter-example has to be recognized by rejecting
$$g(x_p) = \bot, \quad \text{for } p \in S_-.$$
4. Finally, for every pair of prefixes (p, pa) ($a \in \Sigma$, $p, pa \in P$), there has to be exactly one transition from q_{x_p} to $q_{x_{pa}}$ on a symbol a. We can guarantee this by requiring
$$f_a(x_p, x_{pa}) = \top,$$
$$f_a(x_p, i) \implies x_{pa} = i, \quad \text{for } 0 \le i \le K.$$

The conjunction of above-mentioned clauses makes the desired SMT formula.

[5] `https://github.com/Z3Prover/z3`

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

Theorem 1. *The above encoding of (1)-(4) is proper, i.e., if there exists a DFA with at most $K+1$ states that matches a given sample S, then a DFA A determined by variables x and functions f, g accepts all examples and rejects all counter-examples.*

Outline of the proof. First note that if there exists a DFA with at most $K+1$ states that matches a given sample S, then the SMT formula has to be satisfiable. Let A be a DFA determined by x, f, and g. Take any $u \in S_+$, $u = a_0 a_1 \cdots a_{m-1}$, $m \geq 0$. Because of (4) there is exactly one path q_ε, q_{a_0}, $q_{a_0 a_1}$, ..., q_u in A that is traversed on reading the sequence of symbols $a_0, a_1, \cdots, a_{m-1}$. On account of (2) the last state $q_u \in F$. So u is recognized by accepting. Now, take any $w \in S_-$, $w = a_0 a_1 \cdots a_{m-1}$, $m \geq 0$. Similarly, based on (4) and (3), we can conclude that w is recognized by rejecting. Naturally, $s = q_\varepsilon$. Because every x_p is not bigger than K, the automaton A has at most $K+1$ states.

3.2 Symmetry Breaking

So as to improve the speed of search process we follow the symmetry-breaking advice [4]. In case there is no k-state DFA for a given sample S, the corresponding (unsatisfiable) SMT instance will solve the problem many times: once for each permutation of the state indices. Therefore, we build an APTA A for S, and then construct the graph G whose vertices are the states of A and there are edges between vertices that can be merged (i.e., x_p and x_r can get the same state index). There are two types of constraints which must be respected during graph building:

- Consistency constraints (on pairs of states): $q_p \in F$ cannot be merged with $q_r \in R$.
- Determinization constraints (on couple of pairs of states): if $\delta(q_{p_1}, a) = q_{r_1}$ and $\delta(q_{p_2}, a) = q_{r_2}$, then merging q_{p_1} with q_{p_2} implies that q_{r_1} and q_{r_2} must also be merged in order to produce a deterministic automaton.

Notice that in any valid solution to an SMT instance, all vertices in an independent set[6] in G must get a different index. So, we fix vertices in a large independent set I to numbers (states indices) in a preprocessing step. For finding an independent set a greedy algorithm analyzed in [9] is used. Obviously, if there is an independent set of size n in G ($|I| = n$), then there is no k-state DFA for S with $k < n$.

3.3 Iterative SMT Solving

The translation of DFA identification into SMT (direct encoding with symmetry breaking predicates) uses a fixed set of states. To prove that the minimal size of a DFA equals K, we have to show that the translation with K states is satisfiable

[6] The independent set problem and the well-known clique problem are complementary: a clique in G is an independent set in the complement graph of G and vice versa.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

and that the translation with $K-1$ states is unsatisfiable. Algorithm 1 is used to determine the minimal size.

Algorithm 1 Determine minimum-state DFA for a given sample

```
function MinDFA(S+, S−, Σ)
    K ← |I| as shown in Section 3.2
    loop
        construct an SMT formula as shown in Sections 3.1 and 3.2
        solve the formula (we use Z3)
        if the formula is satisfiable then
            decode a DFA using variables x and function g
            return the DFA
        else
            K ← K + 1
        end if
    end loop
end function
```

4 Experimental Results

In this section, we describe some experiments comparing the performance of our approach implemented[7] in Python (SMT) with Pena and Oliveira's C implementation[8] of backtrack search (BICA), our effective implementation[9] of Lang's algorithm in C++ (EXBAR), Zakirzyanov et al's translation-to-SAT approach implemented[10] in Java (SAT), and Smetsers et al's translation-to-SMT approach implemented[11] in Python (Z3GI), when positive and negative words are given. For these experiments, we used a set of 70 samples based on randomly generated regular expressions.

4.1 Brief Description of Other Approaches

The Bica [17] algorithm incrementally builds a hypothesis DFA by examining the nodes in the APTA in breadth-first order. For each of these checks, the algorithm decides whether it is possible to identify that node with one node in the hypothesis, or whether the hypothesis DFA needs to be changed. In order to do this efficiently, it applies advanced search techniques which general idea is based on conflict diagnosis. First, the full set of restrictions is created and entered into a constraint database. This database of constraints is then used to keep a set of constraints that define the solution. When a conflict is reached,

[7] `https://github.com/wieczorekw/wieczorekw.github.io/tree/master/SMT4DFA`
[8] We have obtained Linux executable file from the authors.
[9] `https://github.com/lazarow/exbar`
[10] `https://github.com/ctlab/DFA-Inductor`
[11] `https://gitlab.science.ru.nl/rick/z3gi/tree/lata`

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

the algorithm diagnoses which earlier decision is the cause of the conflict, and backtracks all the way to the point where that bad decision was taken.

The Exbar [15] algorithm also starts with building the APTA. In the beginning, all nodes in the APTA are blue (candidates for merging), except for the red (nodes already in a hypothesis) root. Then, the algorithm considers the blue nodes that are adjacent to the red nodes and for a selected candidate decides whether to accept it (changing its color into red) or merge with another previously accepted node. The order in which the blue nodes are picked and merged matters, hence Exbar first chooses nodes that can be disposed of in as few ways as possible. The best kind of blue node is the node that cannot be merged with any red node. The next best kind is a node that has only one possible merge and so forth. Additionally, the algorithm tries all possible merges in order to avoid latent conflicts. Exbar searches a deterministic automaton that has at most N states (the red nodes limit) and is consistent with all positive and negative examples. If DFA is not found, then the maximum number of the red nodes is increased.

The headmost idea of SAT encoding comes from [10], where the authors based on transformation from DFA identification into graph coloring proposed in [3]. In another work [26] BFS-based symmetry breaking predicates were proposed, instead of original max-clique predicates, which improved the translation-to-SAT technique what was demonstrated with the experiments on randomly generated input data. The core idea is as follows. Consider a graph H, the complement of a graph G described in Section 3.2. Finding minimum-size DFA is equivalent to a graph coloring (i.e., such an assignment of labels traditionally called 'colors' to the vertices of a graph H that no two adjacent vertices share the same color) with a minimum number of colors. The graph coloring constraints, in turn, can be efficiently encoded into SAT [24].

Suppose that $A = (\Sigma, Q = \{0, 1, \ldots, K-1\}, s = 0, F, \delta)$ is a target automaton and P is the set of all prefixes of $S_+ \cup S_-$. An SMT encoding proposed in [21] uses four functions: $\delta\colon Q \times \Sigma \to Q$, $m\colon P \to Q$, $\lambda^A\colon Q \to \mathbb{B}$, $\lambda^T\colon S_+ \cup S_- \to \mathbb{B}$, and the following five constraints:

$$
\begin{gathered}
m(\varepsilon) = 0, \\
x \in S_+ \iff \lambda^T(x) = \top, \\
\forall xa \in P\colon x \in \Sigma^*,\, a \in \Sigma \quad \delta(m(x), a) = m(xa), \\
\forall x \in S_+ \cup S_- \quad \lambda^A(m(x)) = \lambda^T(x), \\
\forall q \in Q \quad \forall a \in \Sigma \quad \bigvee_{r \in Q} \delta(q, a) = r.
\end{gathered}
$$

The encoding has been also implemented using Z3Py, the Python front-end of an efficient SMT solver Z3. The main difference between this and our proposals lies in the way they determine state indices and ensure determinism. Smetsers et al. [21] used for these purposes functions m and δ, we used integer variables x and the family of functions f along with the collection of implications. Besides, our δ (decoded from family f) is a relation, while their δ is a surjection, which always defines a completely specified automaton.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

4.2 Benchmarks

As far as we know all common benchmarks are too hard to be solved by exact algorithms without some heuristic non-exact steps. Thus, our own algorithm was used for generating problem instances. This algorithm builds a set of words with the following parameters: size N of a regular expression to be generated, alphabet size A, the number $|S|$ of words actually generated and their minimum, $d_{\min}$, and maximum, $d_{\max}$, lengths. The algorithm is arranged as follows. First, using Algorithm 2 construct a random regular expression E.

Algorithm 2 Generate random expression

```
function GEN(N, star_parent ← False by default)
    if N ≤ 1 then
        return randomly chosen symbol from Σ
    else
        if star_parent then
            operator ← choose randomly: concatenation or alternation
        else
            operator ← choose randomly: concat., alt. or repetition
        end if
        if operator is repetition then
            return Gen(N − 1, True)*
        else if operator is alternation then
            r ← choose a random integer from range [1, max(1, N − 2)]
            return (Gen(r) | Gen(N − r − 1))
        else
            r ← choose a random integer from range [1, max(1, N − 2)]
            return (Gen(r) Gen(N − r − 1))
        end if
    end if
end function
```

Next, obtain corresponding minimum-state DFA M. Then, as long as a sample S is not symmetrically structurally complete[12] with respect to M repeat the following steps: (a) using the Xeger library for generating random strings from a regular expression, get two words u and w; (b) truncate as few symbols from the end of w as possible in order to achieve a counter-example $\bar{w}$, if it has succeeded, add u to S_+ and $\bar{w}$ to S_-. Finally, accept $S = (S_+, S_-)$ as a valid sample if it is not too small, too large or highly imbalanced.

In this manner we produced 70 sets with: $N \in [30, 80]$, $A \in \{2, 5, 6, 7, 8\}$, $|S| \in [60, 3000]$, $d_{\min} = 0$, and $d_{\max} = 850$. For the purpose of diversification samples' structure, 10 of them (those with $A = 6$, and $N = 26, 27, \ldots, 35$) were generated with Σ in Algorithm 2 extended by one or two randomly generated

[12] Refer to Chapter 6 of [11] for the formal definition of this concept.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

words of length between 10 and 20. The file names with samples[13] have the form 'aAwords$N$.txt'.

4.3 Performance Comparison

In all experiments, we used Intel Xeon W-2135 CPU, 3.7 GHz processor, under Ubuntu 20.04 LTS operating system with 32 GB RAM. The time limit (TL) was set to 3600 s. The results are listed in Table 1.

Table 1: Execution times of exact solving DFA identification in seconds. The sign '-' means that the execution was impossible.

Problem	SAT	EXBAR	BICA	Z3GI	SMT
a2words30	7.01	0.01	0.41	1.71	2.69
a2words36	19.91	2.98	2858.36	201.53	29.68
a2words40	22.99	3.70	-	**TL**	74.07
a2words42	0.37	0.01	0.10	0.30	0.44
a2words44	0.48	0.02	0.19	0.83	1.08
a2words45	5.07	0.61	1.15	**TL**	32.56
a2words47	0.50	0.05	0.20	2017.24	1.19
a2words48	3.84	0.96	3.28	232.11	8.27
a2words49	72.39	1.84	-	61	60.76
a2words50	0.82	0.06	0.19	21.91	1.58
a6words26	29.47	**TL**	92.08	33.28	12.15
a6words27	33.88	**TL**	**TL**	9.63	9.67
a6words28	2.58	12.31	13.58	2.36	2.46
a6words29	104.38	589.84	-	9.65	28.85
a6words30	3.20	0.19	0.40	0.70	1.54
a6words31	2.33	0.15	8.56	1.59	3.34
a6words32	10.22	0.4	0.46	2.57	5.62
a6words33	5.30	**TL**	2.69	5.36	7.37
a6words34	87.92	2114.58	213.85	18.17	16.54
a6words35	82.18	0.04	0.60	1.89	8.21
a8words60	118.90	**TL**	23.08	7.73	15.69
a8words61	1588.92	11.67	-	83.31	134.43
a8words62	42.34	1.92	**TL**	25.11	77.03
a8words63	86.44	**TL**	-	16.50	41.92
a8words64	35.68	1.09	0.35	0.91	7.85
a8words65	**TL**	**TL**	-	**TL**	**TL**
a8words66	121.37	1.35	2.96	3.03	12.70
a8words67	43.19	34.72	**TL**	9.75	34.38
a8words68	**TL**	105.04	-	172.39	243.05
a8words69	427.59	23.43	36.30	0.78	34.61
a5words40	8.32	0.93	4.37	1.24	3.78
a5words41	509.23	4.46	-	**TL**	111.65
a5words42	4.71	0.08	1.06	0.38	2.73
a5words43	**TL**	133.03	-	427.64	320.10

[13] `https://github.com/lazarow/exbar/tree/master/samples`

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

Table 1: Execution times of exact solving DFA identification in seconds.

Problem	SAT	EXBAR	BICA	Z3GI	SMT
a5words44	**TL**	0.15	-	2.49	227.65
a5words45	95.95	0.76	1.31	4.16	32.04
a5words46	568.48	0.03	2.32	8.20	27.43
a5words47	**TL**	0.05	-	1.82	130.99
a5words48	7.03	0.20	-	1.16	4.38
a5words49	67.32	**TL**	**TL**	25.16	491.73
a6words50	10.38	0.09	0.77	3.25	3.77
a6words51	328.00	**TL**	-	37.97	76.26
a6words52	1.10	0.02	0.17	0.32	0.82
a6words53	**TL**	5.02	-	8.68	143.01
a6words54	66.08	0.36	9.56	6.47	18.09
a6words55	848.91	**TL**	-	11.41	43.55
a6words56	**TL**	0.93	-	36.25	147.75
a6words57	135.24	**TL**	-	36.60	179.41
a6words58	1374.14	184.24	-	25.16	83.36
a6words59	16.90	0.54	0.91	1.87	5.17
a7words60	135.74	0.04	0.58	1.63	10.71
a7words61	1.48	0.00	0.15	0.7	1.28
a7words62	149.90	13.63	**TL**	26.85	35.88
a7words63	9.00	0.51	1.03	0.89	5.36
a7words64	0.62	0.00	0.10	0.17	0.57
a7words65	7.47	0.05	0.52	0.70	2.75
a7words66	412.56	0.12	3.8	4.17	62.81
a7words67	3147.21	2829.06	-	33.46	488.95
a7words68	158.95	71.04	-	96.68	90.91
a7words69	114.08	**TL**	**TL**	44.94	674.51
a8words70	349.94	193.38	**TL**	57.11	272.24
a8words71	**TL**	**TL**	-	5.67	444.03
a8words72	1238.59	3.69	-	17.43	56.11
a8words73	**TL**	16.52	-	1.96	370.62
a8words74	46.02	26.78	0.37	29.18	7.67
a8words75	200.96	**TL**	87.07	17.35	34.16
a8words76	1206.09	164.72	-	15.36	114.76
a8words77	61.51	**TL**	1649.78	936.65	20.76
a8words78	**TL**	150.43	-	**TL**	260.69
a8words79	2368.12	**TL**	-	**TL**	TL
Mean	751.56	867.25	686.88	378.18	187.40

First, note that two of the analyzed solutions are written in Python (SMT and Z3GI) and use Z3 library that is implemented in C++, the remaining algorithms are implemented in C (BICA), C++ (EXBAR) and Java (SAT). While the comparison of the execution times of algorithms in Python is not objectionable, doubts may arise when comparing Python implementations with other programming languages. First of all, note that Python is a scripting language, while C, C ++, and Java are non-scripting languages. As shown by empirical

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

research [18], in the initialization phase of a C and C ++ program, programs show up to three and four times advantage over Java and five to ten times faster than script languages. In the internal data structure search phase, the advantage of C and C ++ over Java is about two-fold, and the scripting languages are comparable or may even be faster than Java.

While computing the mean values, all TL cells were substituted by 3600. The dash sign (only in BICA column) means that the program we obtained form the authors was not able to execute on a certain file due to the "Too many collisions. Specify a large hash table." error. Therefore, for the comparison between SMT and BICA, we took only those rows which do not contain the dash sign.

In order to determine whether the observed mean difference between SMT and remaining methods is a real CPU time decrease we used a paired samples t test [20, pp. 1560–1565] for SMT vs SAT, SMT vs EXBAR, SMT vs BICA, and SMT vs Z3GI. As we can see from Table 4.3, p value is low in all cases, so we can conclude that our results did not occur by chance and that using our SMT encoding is likely to improve CPU time performance for prepared benchmarks. It must be mentioned, however, that the difference between the two SMT-based approaches (SMT and Z3GI) is not of strong significance. The p-value is 0.07. Usually p-value should be below 0.05 to tell that the rejection of the hypothesis H_0 (the two means are equal) is strong, or the result is highly statistically significant. On the other hand, the execution times would seem to suggest that for large or complex data sets, SMT performs better. In Table 1 there is no entry in which SMT exeeds the time limit and Z3GI does not, but there are four entries (a2words40 with 382 words for which a minimum-size DFA has 13 states, a2words45 with 485 words for which a minimum-size DFA has 14 states, a5words41 with 1624 words for which a minimum-size DFA has 11 states, and a8words78 with 1729 words for which a minimum-size DFA has 4 states) in which Z3GI exeeds the time limit and SMT does not.

Table 2. Obtained p values from the paired samples t test.

SMT vs SAT	SMT vs EXBAR	SMT vs BICA	SMT vs Z3GI
1.51e−4	1.11e−4	2.13e−3	7.10e−2

5 Conclusions

We presented an efficient translation from DFA identification into an SMT instance. By performing this new encoding, we can use the advanced SMT solvers more efficiently than using the earlier approach shown in the literature. In experimental results, we show that our approach outperforms the current state-of-the-art satisfiability-based methods and the well-known backtracking algorithms, which was confirmed by an appropriate statistical test.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

It is possible to verify whether the DFAs found are equivalent to the given regexes used in benchmark generation, but it seems useless because the samples are too small to infer a target automaton. On the other hand, the size of a proper sample is beyond the scope of exact algorithms, such as our SMT-based or SAT-based ones. For such big data, we use—in the GI field—heuristic search. In some sense, then, the minimal-size DFA is an overgeneralization, but one should remember that thanks to the parameter K, we can also obtain less general automata. This parameter can be regarded as the degree of data generalization. The smallest K for which our SMT formula is satisfiable, will give the most general automaton. As K increases, we obtain a set of less general automata. What is more, usually the running time for larger K is growing short.

We see here a new area of research. One may ask, for example, whether for any K an obtained DFA is equivalent to an original regex. Perhaps it is a matter of a number of factors including the size of data, the density of data, etc.

References

1. Angluin, D.: An application of the theory of computational complexity to the study of inductive inference. Ph.D. thesis, University of California (1976)
2. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017)
3. Coste, F., Nicolas, J.: Regular inference as a graph coloring problem. In: Workshop on Grammar Inference, Automata Induction, and Language Acquisition, ICML 1997 (1997)
4. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning. pp. 148–159. Morgan Kaufmann Publishers Inc. (1996)
5. Downey, P.J., Sethi, R., Tarjan, R.E.: Variations on the common subexpression problem. J. ACM **27**(4), 758–771 (1980)
6. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification. Lecture Notes in Computer Science, vol. 4144, pp. 81–94. Springer Berlin Heidelberg (2006)
7. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**, 302–320 (1978)
8. Gold, E.M.: Language identification in the limit. Information and Control **10**, 447–474 (1967)
9. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: Approximating independent sets in sparse and bounded-degree graphs. Algorithmica **18**(1), 145–163 (1997)
10. Heule, M., Verwer, S.: Exact DFA identification using SAT solvers. In: Grammatical Inference: Theoretical Results and Applications 10th International Colloquium, ICGI 2010. Lecture Notes in Computer Science, vol. 6339, pp. 66–79. Springer (2010)
11. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, second edn. (2001)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5

13. Imada, K., Nakamura, K.: Learning context free grammars by using SAT solvers. In: Proceedings of the ICMLA 2009. pp. 267–272. IEEE Computer Society (2009)
14. Kroening, D., Strichman, O.: Decision Procedures – An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer (2016)
15. Lang, K.J.: Faster algorithms for finding minimal consistent DFAs. Tech. rep., NEC Research Institute (1999)
16. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer Berlin Heidelberg (2008)
17. Pena, J.M., Oliveira, A.L.: A new algorithm for exact reduction of incompletely specified finite state machines. IEEE Trans. on CAD of Integrated Circuits and Systems **18**(11), 1619–1632 (1999). https://doi.org/10.1109/43.806807
18. Prechelt, L.: An empirical comparison of c, c++, java, perl, python, rexx and tcl. IEEE Computer **33**(10), 23–29 (2000)
19. Sakakibara, Y.: Learning context-free grammars using tabular representations. Pattern Recognition **38**(9), 1372–1383 (2005)
20. Salkind, N.J.: Encyclopedia of Research Design. SAGE Publications, Inc (2010)
21. Smetsers, R., Fiterau-Brostean, P., Vaandrager, F.W.: Model learning as a satisfiability modulo theories problem. In: Language and Automata Theory and Applications – 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings. pp. 182–194 (2018)
22. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975)
23. Trakhtenbrot, B.A., Barzdin, Y.M.: Finite automata: behavior and synthesis. North-Holland Publishing Company (1973)
24. Walsh, T.: SAT v CSP. In: Dechter, R. (ed.) Principles and Practice of Constraint Programming – CP 2000. pp. 441–456. Springer Berlin Heidelberg (2000)
25. Wieczorek, W.: Grammatical Inference: Algorithms, Routines and Applications, Studies in Computational Intelligence, vol. 673. Springer (2017)
26. Zakirzyanov, I., Shalyto, A., Ulyantsev, V.: Finding all minimum-size DFA consistent with given examples: SAT-based approach. In: Cerone, A., Roveri, M. (eds.) Software Engineering and Formal Methods. pp. 117–131. Springer International Publishing, Cham (2018)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_5