Proceedings of Machine Learning Research 93:17–29, 2019 International Conference on Grammatical Inference

Inferring DFA without Negative Examples

Florent Avellaneda FLORENT.AVELLANEDA@CRIM.CA
Alexandre Petrenko ALEXANDRE.PETRENKO@CRIM.CA
Computer Research Institute of Montreal
405 Ogilvy Avenue, Suite 101
Montreal (Quebec), H3N 1M3, Canada

Editors: Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek

Abstract

The inference of a Deterministic Finite Automaton (DFA) without negative examples is one of the most natural inference problems. On the other hand, it is well known that DFA cannot be identified in the limit from positive examples only.

We propose two modifications of this problem to make it solvable, i.e., identifiable in the limit, while remaining rather close to the original problem. First, we propose to use the inclusion of languages to reason about complexity and infer the simplest solution. Second, we set the maximum number of states for the inferred DFA. These changes bring new means to control the solution space. While the language inclusion allows us to choose a simplest solution among possible solutions, the maximum number of states determines the degree of approximation. We propose an efficient inference method based on the incremental use of a SAT solver and demonstrate on a practical example the relevance of our approach.

Keywords: Grammatical Inference, Learning from Text, Inference, DFA, Inferring without Negative Examples, SAT Solving, Identifiable in the Limit.

1. Introduction

The problem of inferring a DFA from positive examples only is a long standing problem studied in numerous works. Also referred to "learning from text" in literature, this problem is considered by many to be the essence of language learning. As de la Higuera (2010) conveys, it is in a sense the *initial* problem, the one with least constraints.

At the same time, Gold (1967, Theorem I.8) showed that any class of languages over an alphabet Σ that contains every finite language together with at least one infinite language over Σ cannot be correctly inferred from positive examples. Since this applies even to the class of finite-state languages over Σ, several approaches were proposed trying to make the problem easier. The classic approach is to consider negative examples (Gold, 1978) or calling an Oracle (Angluin, 1987). Another approach is to focus on particular language classes. For example, Angluin (1979) introduces the class of *pattern languages* where a pattern is defined to be a concatenation of constants and variables, and the language of a pattern is the set of strings obtained by substituting constant strings for the variables.

There exists also an approach adopting a probabilistic view by learning distributions over strings (Clark and Thollard, 2004; Carrasco and Oncina, 1999; Abe and Warmuth, 1992). Thus, by assuming that the examples follow the distribution probabilities, the problem can

be solved. However, since the probabilistic approach has a high complexity, most of the work use heuristics addressing the problem.

Although probabilistic approaches are very popular and elegant, we take a radically different approach in this paper.

Our approach also aims at easing the problem by modifying the initial setup. We make two modifications to the classical DFA inference approach from positive and negative examples to adapt it to the inference without negative examples. This allows us to solve the problem of identifiability in the limit, as defined by Gold (1967). A class of languages is identifiable in the limit by an algorithm if for any language $\mathcal{L}$ of this class, after a certain number of examples, the algorithm always infers the same language $\mathcal{L}$.

When we want to infer a DFA we look for the minimal automaton consistent with the given examples. This approach is very natural and follows the principle of parsimony according to which we usually choose the simplest solution. However, trying to minimize the number of states does not make much sense in the absence of negative examples. Indeed, a single state automaton accepting all strings in Σ^*, that we call here the chaos machine, is a trivial and universal solution. We propose to use the inclusion of languages to reason about complexity and choose the simplest solution. Thus, we consider that one DFA is simpler than another if its language is strictly included in the language of the latter. The first modification fixes the problem of universal simplest solution but yet a trivial solution exists: it suffices to infer an acyclic DFA accepting only the positive examples. This is the simplest solution because only observed words are represented. However, we want to learn the totality of a language and not just the observed subset of this language and to use a smaller set of states. To this end, we add another modification: we set the maximum number of states for the inferred DFA.

These changes bring new means to control the solution space. While the language inclusion allows us to choose a simplest solution among conjectures with the equal number of states, the maximal number of them determines the degree of approximation. Thus, the more states are allowed, the more precise the model will be. Considering that the DFA that accepts exactly the positive examples is a base solution, decreasing the maximal number of states increases the language of the inferred DFA. In the extreme case, when $n = 1$, the chaos machine will be the result.

Although the problem is NP-complete, we are encouraged by a recent efficient inference technique (Avellaneda and Petrenko, 2018) that is based on the incremental resolution of a Boolean formula by a SAT solver. The use of such a method provides reasonable execution time as Heule and Verwer (2010) have already shown.

The paper is organized as follows. Section 2 contains definitions. Section 3 details the method that uses a SAT solver. Section 4 elaborates an algorithm for checking the uniqueness of a solution. Section 5 introduces characteristic positive examples as a set of strings such that only one DFA can be inferred with our method and proposes an algorithm to construct it for a given DFA. In Section 6, we use our method to infer a model of a communication protocol. Finally, we conclude in Section 7.

2. Definitions

A DFA is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, consisting of a finite set of states Q, a finite set of symbols Σ called the alphabet, a (partial) transition function $\delta : Q \times \Sigma \to Q$, an initial state $q_0 \in Q$ and a set of accepting states $F \subseteq Q$. We denote by $q \xrightarrow{a} q'$ a transition from state q to state q' with symbol $a \in \Sigma$ and by $|\mathcal{A}|$ the number of states in Q.

The partial transition function δ can be extended, giving the following recursive definition of $\delta : Q \times \Sigma^* \to Q$. $\delta(q, \epsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$ where ϵ is the empty string, $w \in \Sigma^*, a \in \Sigma$ and $q \in Q$. We denote by $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$ the set of all strings accepted by the DFA $\mathcal{A}$.

A *sample* is a tuple of two finite sets of strings $S = (S_+, S_-)$. The set S_+ (positive examples) represents accepted strings and the set S_- (negative examples) represents rejected strings. We say that a DFA $\mathcal{A}$ is *consistent* with a sample $S = (S_+, S_-)$ if $S_+ \subseteq L(\mathcal{A})$ and $S_- \cap L(\mathcal{A}) = \emptyset$.

If S_- is absent we call a DFA consistent with S_+ a *conjecture* for S_+. Limiting the maximum number of states in conjectures, we have the following definition

Definition 1 *A DFA $\mathcal{A}$ is an* n-conjecture *for S_+ if $S_+ \subseteq L(\mathcal{A})$ and $|\mathcal{A}| \leq n$.*

Note that since the number of n-conjectures is bounded, the inference problem without negative examples becomes decidable.

Definition 2 *A DFA $\mathcal{A}$ is* minimal *if for each $\mathcal{A}'$ such that $|\mathcal{A}'| < |\mathcal{A}|$ we have $L(\mathcal{A}) \neq L(\mathcal{A}')$.*

Definition 3 *A minimal DFA $\mathcal{A}$ is a* simplest n*-conjecture for S_+ if for each n-conjecture $\mathcal{A}'$ for S_+ we have $L(\mathcal{A}') \not\subset L(\mathcal{A})$.*

The idea behind this definition is to use the languages recognized by DFAs to decide if one automaton is simpler than the other. Moreover, a conjecture to be simplest need not to have equivalent states, thus a simplest n-conjecture is also a minimal DFA. The language recognized by a DFA is prefix closed if and only if $Q = F$.

In this paper, we will consider only prefix closed languages, but our results can be adapted for not prefix closed languages. We let $\mathcal{A}_{Chaos}$ denote the chaos machine, i.e., DFA with a single state accepting all strings over Σ.

3. Inference with SAT Solving

In this section, we show that a simplest n-conjecture could be inferred from positive examples S_+ using SAT solving. Inferring it, we also infer negative examples as counterexamples that were used to refute the intermediate conjectures.

This section is organized as follows. The Section 3.1 presents our algorithm. This algorithm refers to SAT formulas described in Section 3.2. Finally, Section 3.3 presents an optimization of our SAT formulation by breaking symmetry.

3.1. Inference algorithm

We elaborate an algorithm which iteratively adds constraints to a SAT formula (Algorithm 1). The algorithm works as follows. We start by considering the conjecture $\mathcal{A}_{Chaos}$ as the current solution $\mathcal{A}$. Indeed, this conjecture is consistent with S_+. Then we try to find a solution with a smaller language iteratively. To do that we search for a DFA $\mathcal{A}'$ with at most n states satisfying a growing set of constraints (initially we do not have any constraints).

- If $\mathcal{A}'$ is not consistent with S_+, i.e., there exists a string w in S_+ that is not accepted by $\mathcal{A}'$, then we add the constraint that w has to be accepted and try to find another conjecture $\mathcal{A}'$.
- If $\mathcal{A}'$ is consistent with S_+, but not language-included in the current solution $\mathcal{A}$, i.e., there exists a string w in $L(\mathcal{A}')$ that is not accepted by $\mathcal{A}$, then we add the constraint that w must not be accepted, add w in S_- which is initially empty, and search for a new conjecture $\mathcal{A}'$.
- If $\mathcal{A}'$ is consistent with S_+ and strictly language-included in $\mathcal{A}$, so there exists $w \in L(\mathcal{A}) \setminus L(\mathcal{A}')$, then we consider $\mathcal{A}'$ as the updated current solution, add the constraint in order not to find $\mathcal{A}'$ again, and the constraint that w must not be accepted, include w into S_-, and try to find a new conjecture $\mathcal{A}'$.
- If $\mathcal{A}'$ is consistent with S_+ and has the same language as $\mathcal{A}$ then we add the constraint excluding $\mathcal{A}'$ in order not to find it again and try to find a new conjecture $\mathcal{A}'$.

The process continues as long as the constraints are satisfiable. When no more solution can be found by the SAT solver, we obtain a simplest n-conjecture by minimizing, if needed, the number of states of $\mathcal{A}$. Indeed, minimization is required because by definition a simplest n-conjecture is minimal. The DFA before minimization has at most n states, but it is not necessarily minimal. A simplest n-conjecture is not always unique and the obtained set of negative examples S_- used to refute the intermediate conjectures represents assumptions made inferring $\mathcal{A}$.

Theorem 4 *Given positive examples S_+ and integer n, Algorithm 1 returns a simplest n-conjecture.*

Proof Suppose that the DFA $\mathcal{A}$ returned by $Infer$ is not a simplest n-conjecture. We know that the algorithm returns an n-conjecture because an invariant is that at any time $\mathcal{A}$ is consistent with S_+. We also know that the algorithm returns a minimal DFA because a call to a minimization function is performed at the end of the function. So if $\mathcal{A}$ is not a simplest n-conjecture, then there exists an n-conjecture $\mathcal{A}'$ for S_+ such as $L(\mathcal{A}') \subset L(\mathcal{A})$. There are three types of constraints in C: string has to be accepted, string has not to be accepted and a conjecture has to be excluded. When there is no more solution satisfying the constraints C it means that no DFA is left that accepts the set of strings S_+ and does not accept any string of S_-. If $L(\mathcal{A}') \subset L(\mathcal{A})$ then $\mathcal{A}'$ accepts all the strings of S_+ and does not accept any string of S_- and therefore $\mathcal{A}'$ is excluded. However, a conjecture is excluded

only when a smaller or equal (by the language inclusion) n-conjecture is found. So $L(\mathcal{A}')$ is not included in $L(\mathcal{A})$, a contradiction.

The termination is assured because in each execution of the while loop, at least one DFA is removed from the solutions satisfying the constraints C. Thus, the loop will eventually be exited because the number of DFAs with at most n states is bounded. ■

Algorithm 1: Inferring a simplest n-conjecture

Input: Positive examples S_+ and an integer n
Output: A simplest n-conjecture for S_+ and negative examples S_-

Function *Infer*(S_+, n)**:**
  Initialize C to $\emptyset$, S_- to $\emptyset$ and $\mathcal{A}$ to $\mathcal{A}_{Chaos}$
  while C *is satisfiable* **do**
    Let $\mathcal{A}'$ be a DFA of a solution of C.
    if $S_+ \nsubseteq L(\mathcal{A}')$ **then**
      Let w be a shortest[1] string in $S_+ \setminus L(\mathcal{A}')$.
      $C \leftarrow C \wedge C_w$, where C_w is clauses encoding the requirement that w must be in the conjecture (Table 1).
    else
      if $L(\mathcal{A}') \subseteq L(\mathcal{A})$ **then**
        $C \leftarrow C \wedge C_{\mathcal{A}}$, where $C_{\mathcal{A}}$ is a clause to further exclude the current solution (Clause (5)).
        if $L(\mathcal{A}') \subset L(\mathcal{A})$ **then**
          Let w be a shortest string in $L(\mathcal{A}) \setminus L(\mathcal{A}')$.
          $C \leftarrow C \wedge C_w$, where C_w is clauses encoding the requirement that w must not be in the conjecture (Table 2).
          $S_- \leftarrow S_- \cup \{w\}$
          $\mathcal{A} \leftarrow \mathcal{A}'$
        else
          Let w be a shortest string in $L(\mathcal{A}') \setminus L(\mathcal{A})$.
          $C \leftarrow C \wedge C_w$, where C_w is clauses encoding the requirement that w must not be in the conjecture (Table 2).
          $S_- \leftarrow S_- \cup \{w\}$
  return $min(\mathcal{A}), S_-$ // min is the minimization of a DFA

Definition 5 *We say that $S = (S_+, S_-)$ is a* characteristic sample *for a minimal DFA $\mathcal{A}$ if $\mathcal{A}$ is consistent with S and if for each $\mathcal{A}'$ consistent with S such that $|\mathcal{A}'| \leq |\mathcal{A}|$ we have that $\mathcal{A}'$ is isomorphic to $\mathcal{A}$.*

The idea of the characteristic sample definition is very close to that of Oncina and Garcia's (Oncina and García, 1992). They define conditions such that if a sample is characteristic for a DFA then their algorithm is guaranteed to return a canonical representation of this DFA. Our definition refers not to any particular algorithm, but to a minimal DFA consistent with the sample S.

The execution time of this algorithm is determined by two factors, a SAT instance solving complexity and the number of instances created by the algorithm, as it works incrementally. In the worst case, the number of iterations increases exponentially with n.

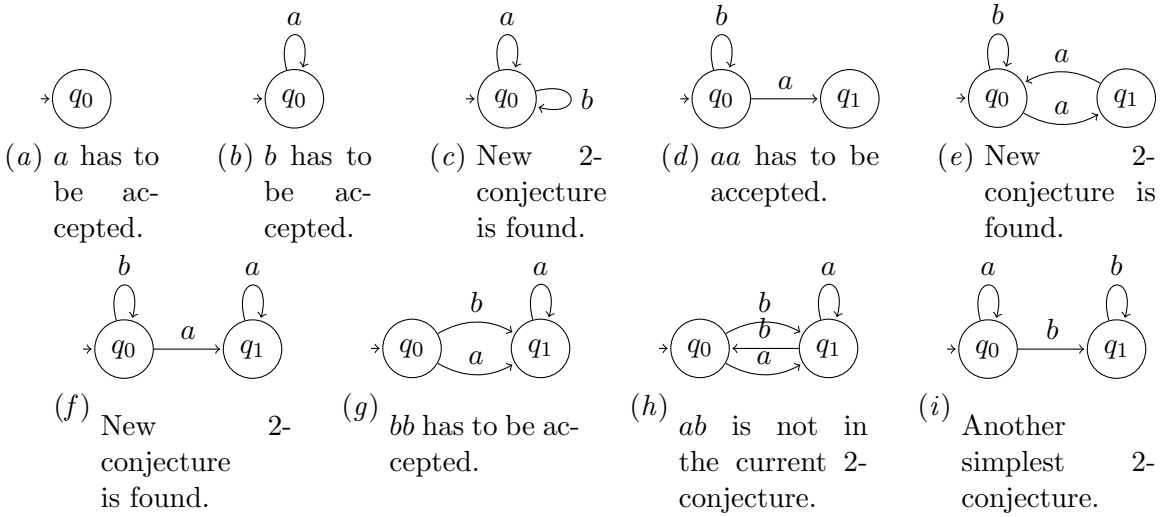
1. by lexicographical order

Theorem 6 *Let $(\mathcal{A}, S_-)$ be the result of Algorithm 1 for S_+ and n. Then (S_+, S_-) is a characteristic sample for $\mathcal{A}$.*

Proof Suppose that the theorem does not hold. By Definition 5, there exists a DFA $\mathcal{A}'$ such that $|\mathcal{A}'| \leq |\mathcal{A}$, consistent with the sample $S = (S_+, S_-)$ that is not isomorphic to $\mathcal{A}$. When there is no more solution satisfying the constraints C it means that there is no DFA left that accepts the set of string S_+, does not accept any string of S_- and has not yet been excluded. Since $\mathcal{A}'$ contains the strings of S_+ and does not accept any string of S_-, $\mathcal{A}'$ is excluded. A conjecture is excluded only when a smaller or equal (by the language inclusion) n-conjecture is found. However, the excluded conjectures that do not have the same language as $\mathcal{A}$ are not consistent with S, because, when a smaller conjecture is found, a string not included in the language of the last conjecture is added to S_-. So $L(\mathcal{A}) = L(\mathcal{A}')$. Since $\mathcal{A}$ is minimal, if $L(\mathcal{A}) = L(\mathcal{A}')$ then $|\mathcal{A}'| = |\mathcal{A}|$ and $\mathcal{A}'$ is isomorphic to $\mathcal{A}$. ■

Example 1 *Let us consider $S_+ = \{\epsilon, a, aa, aaa, b, bb, bbb\}$ and execute $Infer(S_+, 2)$. We present all intermediate DFAs generated by a SAT solver and all the constraints that are added incrementally. Initially, C has no constraints, the solver finds a trivial solution that corresponds to a DFA recognizing only the empty language (Fig. 1(a)). The constraint "a must be in the conjecture" is therefore added to C. The next DFA found (Fig. 1(b)) is not consistent with S_+. Because b is the shortest string in $S_+ \setminus L(\mathcal{A}')$, the constraint "b must be in the conjecture" is added to C. After that, we obtain the DFA $\mathcal{A}'$ in Fig. 1(c). For this DFA it holds that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. The DFA is thus considered as the current solution and constraints are added to not find this same solution any more. The next DFA found (Fig. 1(d)) is not consistent with S_+, so the constraint "a must be in the conjecture" is added to C. The next DFA $\mathcal{A}'$ (Fig. 1(e)) is consistent with S_+. For this DFA it holds that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. This DFA is thus considered as the current solution and constraints are added to not find this same solution any more. The next found DFA (Fig. 1(f)) is also consistent with S_+. Its language is also included in $L(\mathcal{A})$. This DFA is thus considered as the current solution and constraints are added to not find this same solution any more. The next DFA in (Fig. 1(g)) is not consistent with S_+, the constraint "bb must be in the conjecture" is added to C. The next DFA in (Fig. 1(h)) is consistent with S_+ but its language is not included in $L(\mathcal{A})$ i.e., ab is accepted by this DFA, but it is not in $L(\mathcal{A})$. The string ab is added in S_- and the constraint "ab must not be in the conjecture" is added to C. After adding this constraint, the formula C is not satisfiable. There is no more DFA satisfying all the constraints we have. The solution is the last current solution, that is the DFA in (Fig. 1(f)) with $S_- = \{ab\}$. So, (S_+, S_-) is a characteristic sample for the DFA in (Fig. 1(f)) because only this DFA with 2 states is consistent with (S_+, S_-). Note that in this example there are two simplest 2-conjectures, the DFAs in (Fig. 1(f)) and (Fig. 1(i)). Negative examples constructed by the algorithm distinguish the simplest conjectures among each others. In our example, the negative example $S_- = \{ab\}$ distinguishes the two conjectures because ab is not accepted by the DFA in (Fig. 1(f)) but is accepted by the DFA in (Fig. 1(i)).*

We show in Section 3.2 how to encode the requirements that a string has to be accepted or not and how to exclude a solution. In Section 3.3 we add breaking symmetry constraints to improve the efficiency of our formulation.



(a) a has to be accepted. (b) b has to be accepted. (c) New 2-conjecture is found. (d) aa has to be accepted. (e) New 2-conjecture is found.

(f) New 2-conjecture is found. (g) bb has to be accepted. (h) ab is not in the current 2-conjecture. (i) Another simplest 2-conjecture.

Figure 1: Intermediate DFAs and the constraints.

3.2. Accepted strings, rejected strings and excluded solutions

In this section, we formulate constraints to ensure that a DFA represented by a solution of the resulting SAT formula accepts a given string, ditto for a string which should not be accepted. Given a string $w = a_1a_2...a_m$, we let $\mathcal{A}_w = (Q_w, \Sigma_w, \delta_w, q_0, Q_w)$ denote the minimal (linear) DFA accepting w and all prefixes of w. We first consider that the string must be accepted.

The idea of encoding the constraint is to partition the states of Q_w into at most n blocks. By merging all the states in each block, we obtain a DFA with no more than n states. The constraints we give to the SAT solver are such that a DFA corresponding to a solution is consistent with $\mathcal{A}_w$.

Each state $q \in Q$ is represented by n Boolean variables $v_{q,0}, v_{q,1}, ..., v_{q,n-1}$. If the Boolean variable $v_{q,i}$ is true, this means that the state q is in the block i. We start with constraints which encode the fact that each state of Q_w should be in exactly one block. They consist in two formulas. The first requires that any state should be in at least one block. For each state $q \in Q_w$, we have the clause:

$$\bigvee_{0 \leq i < n} v_{q,i} \tag{1}$$

The second formula requires that each state should be in at most one block. For each state $q \in Q_w$ and for all i, j such that $0 \leq i < j < n$, we have the clauses:

$$\neg v_{q,i} \vee \neg v_{q,j} \tag{2}$$

Then, we enforce the determinism of solutions by requiring that for two states of the same block, their successors for any symbol have also to be in the same block. We encode this property by the following formula. For all $q_1 \xrightarrow{a} q_2$, $q'_1 \xrightarrow{a} q'_2$ and i, j such that $0 \leq i < j < n$, we have a Boolean formula (which can be translated trivially into clauses):

$$(v_{q_1,i} \wedge v_{q'_1,i}) \Rightarrow (v_{q'_2,j} \Rightarrow v_{q_2,j}) \tag{3}$$

Now we consider the case when a string must be rejected. Encoding this, we use again (2) and (3) but replace (1) by a formula which asserts that the last state of Q_w has not to be a state in a solution. For each i such that $0 \leq i < n$, we have the clauses:

$$\neg v_{Q_w[|w|],i} \tag{4}$$

If the last state of Q_w is not assigned to any block in a solution, then w is not accepted by the DFA of the solution.

Finally, to exclude a solution, we just have to add the clause:

$$\bigvee_{v_{q,i}=True} \neg v_{q,i} \tag{5}$$

Note that all the states of the resulting DFAs are accepting states. If we do not assume that the languages we want to infer are prefix- closed, then we must add Boolean variables to indicate whether each state is accepting or not.

3.3. Breaking symmetry

It is possible that different assignments for a given SAT formula are equivalent, representing the same solution. In this case, we say that we have symmetry. A good practice is to break this symmetry (Aloul et al., 2002, 2006; Brown et al., 1988) by adding constraints such that different assignments satisfying the formula represent different solutions.

A formulation can result in a significant amount of symmetry if any permutation of the blocks is allowed. To eliminate this symmetry, we use a symmetry breaking method which forbids block permutations by using a total order on the set of states.

Let $<$ be a total order over the set of states $Q = \bigcup_w Q_w$ for all strings to be accepted and rejected. Based on a chosen order we add the following clauses excluding permutations. For each $q \in Q$ and each i such that $0 \leq i < n-1$, we have a Boolean formula (which can be translated trivially into clauses):

$$(\bigwedge_{q'<q} \neg v_{q',i}) \Rightarrow \neg v_{q,i+1} \tag{6}$$

Intuitively, these clauses force to use blocks not already assigned when a state requires a new block.

4. Uniqueness

The definition of the simplicity of n-conjectures does not guarantee the uniqueness. Indeed, there can be several simplest n-conjectures for given positive examples S_+. We are interested in this section in verifying whether a simplest n-conjecture is unique.

We say that a string w is a *distinguishing string* for a positive sample S_+ and an integer n if there exist two n-conjectures $\mathcal{A}$ and $\mathcal{A}'$ for S_+ such that $w \notin L(\mathcal{A})$ and $w \in L(\mathcal{A}')$.

Theorem 7 *If there exists a single simplest n-conjecture for S_+ then Algorithm 2 determines its uniqueness, otherwise it returns a distinguishing string.*

Table 1: Encoding the requirement that w must be in the n-conjecture.

Ref	Clauses	Condition	Meaning
(1)	$(v_{q,0} \vee v_{q,1} \vee ... \vee v_{q,n-1})$	$q \in Q_w$	Each state should be in at least one block.
(2)	$(\neg v_{q,i} \vee \neg v_{q,j})$	$0 \leq i < j < n$	Each state should be in at most one block.
(3)	$(\neg v_{q_1,i} \vee \neg v_{q'_1,i} \vee \neg v_{q'_2,j} \vee v_{q_2,j})$	$q_1 \xrightarrow{a} q_2$ $q'_1 \xrightarrow{a} q'_2$	Determinism.
(6)	$(\bigwedge_{q'<q} \neg v_{q',i}) \Rightarrow \neg v_{q,i+1}$	$q \in Q_w$ $0 \leq i < n$	Breaking symmetry.

Table 2: Encoding the requirement that w must not be in the n-conjecture.

Ref	Clauses	Condition	Meaning
(1)	$(v_{q,0} \vee v_{q,1} \vee ... \vee v_{q,n-1})$	$q \in Q_w$	Each state should be in at least one block.
(3)	$(\neg v_{q_1,i} \vee \neg v_{q'_1,i} \vee \neg v_{q'_2,j} \vee v_{q_2,j})$	$q_1 \xrightarrow{a} q_2$ $q'_1 \xrightarrow{a} q'_2$	Determinism.
(4)	$\neg v_{Q_w[\lvert w\rvert],i}$	$0 \leq i < n$	The string w is not accepted.
(6)	$(\bigwedge_{q'<q} \neg v_{q',i}) \Rightarrow \neg v_{q,i+1}$	$q \in Q_w$ $0 \leq i < n$	Breaking symmetry.

Proof If the algorithm returns w, then we have $L(\mathcal{A}) \not\subset L(\mathcal{A}')$ where $\mathcal{A}'$ is a simplest n-conjecture for $S_+ \cup \{w\}$ with $w \notin S_+$. Obviously, $\mathcal{A}'$ is a n-conjecture for S_+. So there exists a simplest n-conjecture $\mathcal{A}''$ for S_+ such that $L(\mathcal{A}'') \subseteq L(\mathcal{A}')$. However, this simplest n-conjecture for S_+ cannot be $\mathcal{A}$ because $L(\mathcal{A}) \not\subseteq L(\mathcal{A}')$.

Assume that there exist several simplest n-conjectures for S_+ and let $\mathcal{A}$ and $\mathcal{A}'$ be two of them. Because (S_+, S_-) is a characteristic sample for $\mathcal{A}$, by Definition 5 we know that $\mathcal{A}'$ is not consistent with (S_+, S_-). Then there exists $w \in S_-$ such that $w \notin L(\mathcal{A})$ and $w \in L(\mathcal{A}')$ and the algorithm will return the distinguishing string w. ■

Example 2 *We call the function CheckUniqueness with the result of Example 1, where $\mathcal{A}$ is in Fig. 1(f), $S_+ = \{aaa, bbb\}$ and $S_- = \{ab\}$. So, the algorithm CheckUniqueness calls*

Algorithm 2: Checking whether $\mathcal{A}$ is the single simplest n-conjecture for S_+

Input: An n-conjecture $\mathcal{A}$ and a characteristic sample (S_+, S_-) for $\mathcal{A}$.
Output: Return *True* if $\mathcal{A}$ is the only simplest n-conjecture for S_+ and return a distinguishing string otherwise.

```
Function CheckUniqueness(A, (S+, S−)):
    foreach w ∈ S− do
        (A', S'−) ← infer(S+ ∪ {w}, |A|)
        if L(A) ⊄ L(A') then return w ;
    return True
```

the function $infer(\{aaa, bbb, ab\}, 2)$ *in line 3. The simplest* 2*-conjecture shown in Fig. 1(*i*) will be inferred and so, the string* $w = ab$ *will be returned as a distinguishing string.*

5. Characteristic Positive Examples

Once we know how to determine a simplest n-conjecture for a given set of strings, a natural question on the inverse problem arises. Given a minimal DFA $\mathcal{A}$ with n states, find S_+ a subset of $L(\mathcal{A})$, such that the unique n-conjecture for S_+ is $\mathcal{A}$.

Definition 8 *Positive examples* S_+ *are* characteristic positive examples *for* $\mathcal{A}$ *if the simplest* $|\mathcal{A}|$*-conjecture for* S_+ *is* $\mathcal{A}$ *and it is unique.*

Notice that characteristic positive examples should not be confused with characteristic samples consisting of positive and negative examples. Removing negative examples from a characteristic sample does not necessarily result into characteristic positive examples.

We show in this section that it is always possible to find characteristic positive examples and give an algorithm to generate them (Algorithm 3). This result is interesting because when $\mathcal{A}$ is a black box, it says that with a set of observation S_+, sufficiently large and representative, we will be able to infer correctly a model for $\mathcal{A}$.

Algorithm 3: Generation of characteristic positive examples

Input: A DFA $\mathcal{A}$
Output: Characteristic positive examples for $\mathcal{A}$
1 **Function** *GenerateCharacteristicPositiveExamples*$(\mathcal{A})$**:**
2 $S_+ \leftarrow \emptyset$
3 **while** S_+ *is not a characteristic positive examples for* $\mathcal{A}$ **do**
4 Let $\mathcal{A}'$ be a simplest $|\mathcal{A}|$-conjecture for S_+ for which there exists $w \in L(\mathcal{A})$ such that $w \notin L(\mathcal{A}')$.
5 $S_+ \leftarrow S_+ \cup \{w\}$
6 **return** S_+

Theorem 9 *For each DFA* $\mathcal{A}$*, Algorithm 3 returns characteristic positive examples.*

Proof The algorithm terminates when S_+ is a characteristic positive examples. We prove that it terminates. In each cycle, because S_+ is not a characteristic positive examples, we can find a simplest $|\mathcal{A}|$-conjecture $\mathcal{A}'$ for S_+ such that there exists $w \in L(\mathcal{A}) \setminus L(\mathcal{A}')$. Then at least one conjecture becomes inconsistent with S_+ by adding w in S_+. Hence, the number of conjectures is strictly decreasing in each cycle. Because the maximum number of states is fixed, the number of conjectures is bounded and therefore the algorithm terminates. ■

Theorem 10 *If* S_+ *is characteristic positive examples for* $\mathcal{A}$*, then each* S'_+ *such that* $S_+ \subseteq S'_+ \subseteq L(\mathcal{A})$ *is also characteristic positive examples for* $\mathcal{A}$*.*

Proof Let us show that if $\mathcal{A}'$ is a simplest $|\mathcal{A}|$-conjecture for S'_+ then $\mathcal{A}' = \mathcal{A}$. Assume that $\mathcal{A}'$ is a simplest $|\mathcal{A}|$-conjecture for S'_+. Then $S'_+ \subseteq L(\mathcal{A}')$ and because $S_+ \subseteq S'_+$, we know that $S_+ \subseteq \mathcal{A}'$.

Because $\mathcal{A}$ is a **simplest** $|\mathcal{A}|$-conjecture for S_+, we know that $L(\mathcal{A}') \not\subset L(\mathcal{A})$. Moreover, because $\mathcal{A}$ is a **unique** simplest $|\mathcal{A}|$-conjecture for S_+, we also know that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. So, $\mathcal{A}' = \mathcal{A}$. ■

Corollary 11 *The prefix-closed languages generated by DFAs with n states are identifiable in the limit from positive examples by searching the simplest n-conjectures.*

Proof Let $\mathcal{A}$ be a minimal DFA with n states. By Theorem 9 we know that there exists characteristic positive examples S_+ and by Theorem 10 we know that for each $S'_+ \subseteq L(\mathcal{A})$ such that $S_+ \subseteq S'_+$, the simplest n-conjecture is $\mathcal{A}$. ■

6. Experiments

To evaluate the proposed approach, we infer the model of a communication protocol from messages over the communication channel between two machines (Endriss et al., 2003). The model is represented in Fig. 2(e).

We use a set of 50 traces generated by random walks and increase the maximum number of states of a DFA to infer. The prototype was implemented in C++ calling the SAT solver Cryptominisat (Soos, 2010). The experiments were carried out on a machine with 8 GB of RAM and an i7-3537U processor. The time required to infer a DFA with $n = 7$ is about 4 seconds and less than one second for smaller numbers of states.

When we reach $n = 7$, we find the minimal automaton having the same language as the DFA modeling the communication protocol. Note that with only 50 traces, our approach was able not only to correctly infer the communication protocol, but also to guarantee the uniqueness of the solution.

It is interesting to notice that when the number of states is smaller than the minimum number of states required to exactly represent the communication protocol, we obtain approximations at various levels of abstractions defined by the number of states. Although the relevance of approximations when using values of n smaller than 7 remains subjective in this example, we can still guarantee that each result is optimal in the sense that it is an n-conjecture.

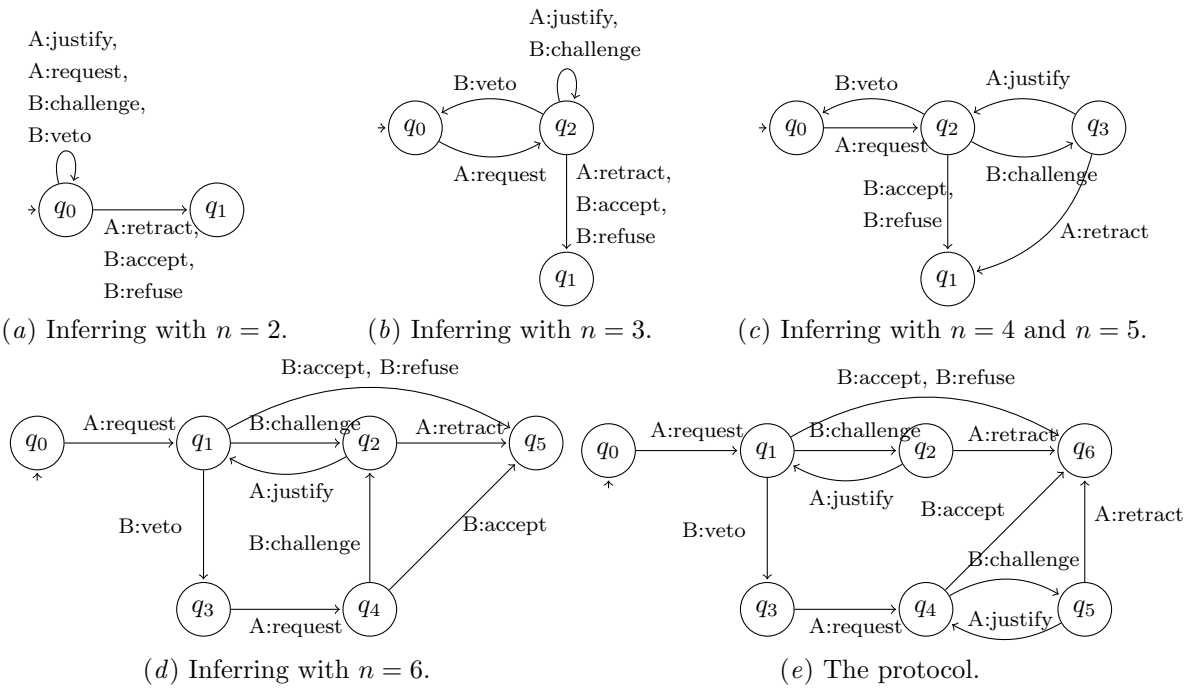
For $n = 2$ the abstraction is significant, but still rather adequate: numerous messages are exchanged before ending up when B accepts or refuses or A retracts.

For $n = 4$ and $n = 5$, the model is more precise. It is easy to identify each state. State q_0 is the initial state, A has to do a *request*. State q_1 is the final state. For the state q_2, B has four possible choices: *accept*, $refuse$, *veto* and *challenge*. For the state q_3, A is challenged and need to make a choice between $justify$ and *retract*. Notice that the number of states of the conjecture for $n = 5$ is four. It simply means that in this case, there is no DFA with five states refining the model.

For $n = 6$, the model almost matches the specification. The main difference is that in the model with fewer states, B can refuse after having challenged A whereas in the original specification, B can only ask for a challenge again or accept.

7. Conclusions

In this paper we considered the problem of inferring a DFA without negative examples. We made two modifications to the inference problem of DFA from positive and negative examples by adapting it to the case when negative examples are absent. The first modification is the use of the language inclusion to reason about complexity and choose the simplest solution. The second modification is fixing the maximum number of states of a DFA to



(*a*) Inferring with $n = 2$. (*b*) Inferring with $n = 3$. (*c*) Inferring with $n = 4$ and $n = 5$.

(*d*) Inferring with $n = 6$. (*e*) The protocol.

Figure 2: Inferring the communication protocol

infer. Then with these two modifications, we proposed an approach to solve the formulated inference problem.

We presented an efficient inference algorithm that uses a SAT solver. We also showed how to check that a solution is unique and to generate characteristic positive examples from a DFA. Finally, we demonstrated on a practical example the relevance of our approach and the use the maximum number of states for obtaining various approximations of a complex language.

We focused in this paper on an exact algorithm solving this inference problem. More research is needed to find heuristics which could allow us to mitigate the complexity issues and thus increase the scalability of the proposed approach. Since we use in this paper a general case of DFAs, it could also be interesting to refine our approach to deal with more specific automata models.

References

Naoki Abe and Manfred K Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine learning*, 9(2-3):205–260, 1992.

Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th annual Design Automation Conference*, pages 731–736. ACM, 2002.

Fadi A Aloul, Karem A Sakallah, and Igor L Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006.

Dana Angluin. Finding patterns common to a set of strings. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 130–141. ACM, 1979.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

Florent Avellaneda and Alexandre Petrenko. FSM inference from long traces. In *International Symposium on Formal Methods*, pages 93–109. Springer, 2018.

Cynthia A Brown, Larry Finkelstein, and Paul Walton Purdom Jr. Backtrack searching in the presence of symmetry. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 99–110. Springer, 1988.

Rafael C Carrasco and Jose Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO-Theoretical Informatics and Applications*, 33(1): 1–19, 1999.

Alexander Clark and Franck Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5(May):473–497, 2004.

Colin de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

Ulle Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Logic-based agent communication protocols. In *Workshop on agent communication languages*, pages 91–107. Springer, 2003.

E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

Marijn JH Heule and Sicco Verwer. Exact DFA identification using SAT solvers. In *International Colloquium on Grammatical Inference*, pages 66–79. Springer, 2010.

José Oncina and Pedro García. Identifying regular languages in polynomial time. *Advances in Structural and Syntactic Pattern Recognition*, 5(99-108):15–20, 1992.

Mate Soos. CryptoMiniSat 2.5. 0. *SAT Race competitive event booklet*, 2010.