



Black Box and White Box Identification of Formal Languages Using Test Sets

MARC BEZEM

Department of Informatics, University of Bergen, Norway
7800, N-5020, Bergen, Norway
E-mail: marc.bezem@ii.uib.no

CHRISTIAN SLOPER

Department of Informatics, University of Bergen, Norway
7800, N-5020, Bergen, Norway
E-mail: christian.sloper@ii.uib.no

TORE LANGHOLM

Department of Linguistics, University of Oslo, Norway
1102, Blindern, 0317 Oslo, Norway
E-mail: tore.langholm@ii.uib.no

Abstract

Moore's seminal paper [9] can be taken as the starting point of Algorithmic Learning Theory. Moore studied the problem of unraveling the inner structure of a (minimum state) deterministic finite automaton (DFA) from its input-output behaviour. In this note we pursue Moore's line of research, studying conditions under which it is possible to compute a grammar and/or an automaton for a given language L from a language class C . It doesn't come as a surprise that such conditions must be quite strong. We improve on the algorithms in some cases where computing the grammar/automaton is possible. We correct some mistakes in the literature and come up with some new results, positive and negative, for (subclasses of) context-free languages.

1 Introduction

Among the various models for grammatical inference in Algorithmic Learning Theory there are three models that have proved to be influential.

The first and oldest is the model by Moore [9] ('Gedanken-experiments'), for which Gold [6] coined the term 'black box identification'. Given some language L and an oracle correctly answering membership queries, the challenge is to identify an automaton or grammar for L . Moore showed, among other things, that it is impossible

to reconstruct an accepting DFA for a regular language without any additional information. If, however, an upper bound for the number of states is known, then an accepting DFA can be construed.

The second model for grammatical inference is Gold's [6] model for learning in the limit. Here a stream of automata is returned in response to a stream of positive and negative samples of a language L . If every word over the alphabet occurs in the sample stream at least once, then from some point on the returned automata all accept L . The crucial difference with the previous model is that regular languages can in this way algorithmically be learned without any additional assumptions, but that the algorithm doesn't know when, so to say, the limit has been reached. Though important, this model plays no role in this paper.

The third influential model which is relevant to discuss here, is Angluin's [3] model of the Minimally Adequate Teacher (MAT). Here a Learner (algorithm) may ask, besides membership queries, also so-called equivalence queries. An equivalence query is a proposed solution in the form of a grammar or automaton, and the response from the Teacher (oracle) is a confirmation in case the proposed solution is a correct one. Otherwise the Teacher returns a word in the symmetric difference of (the languages of) the proposed and the correct solution. The MAT model requires additional information on the DFA, but then the complexity of the algorithm goes down to polynomial, as compared to exponential for Moore [2].

In this paper we shall mainly be concerned with exploring Moore's model in other language classes, notably of context-free languages. Besides the black box model in combination with partial disclosure of details of the target automaton/grammar, we have one result on what can be called a white box model. In the field of software engineering the terms black box and white box (or: transparent box) refer to test design methods [4]. Black box testing only considers the input-output behaviour, whereas white box testing also takes the program code into account. In analogy to the use of these terms in software engineering we use the term *black box learning* when one can ask membership queries and *white box learning* when one has full access to a decision procedure for membership, typically a terminating Turing machine. White box learning does *not* mean having access to the target automaton/grammar, for obvious reasons, although some details may be disclosed.

The machine model underlying the black box model is the so-called *oracle Turing machine* [14, Def. 6.16]. We will not exhibit oracle Turing machines in this note, but we express by $\alpha?$ that the algorithm α uses an oracle. If this is an oracle for membership of a given language L , then we write α_L . The result of executing α_L with input i will be denoted by $\alpha_L(i)$, or $\alpha_L()$ if no input is used. The strings that are actually tested on membership of L during the execution of α_L form the so-called *test set* of α_L . As machine model for the white box model we can take the ordinary Turing machine, which takes the code of another Turing machine deciding a language as (extra) input (cf. the universal Turing machine).

From now on Σ is a fixed but arbitrary, finite but non-empty alphabet. If we say 'word' (or: 'string'), then we mean 'word (string) over Σ '. If we say 'language', then we mean a set of words over Σ . Most of the questions we study in this note have the following basic form.

Given $precondition(L, i)$ on languages L and inputs i , as well as $postcondition(L, i, o)$ on L , i and outputs o , does there exist an algorithm $\alpha?$ such that for all languages L and inputs i ,
 if $precondition(L, i)$, then $postcondition(L, i, \alpha_L(i))$?

Here the input i allows us to express in the precondition additional information on the language L and to pass this information to the postcondition. Examples of concrete questions are:

- Q1 Does there exist an algorithm $\alpha?$ such that for all languages L , if L is regular, then $\alpha_L()$ is a DFA accepting L ? (No input i plays a role here.)
- Q2 Does there exist an algorithm $\alpha?$ such that for all languages L , if L is accepted by some DFA of at most i states, then $\alpha_L(i)$ is a DFA accepting L ?
- Q3 Does there exist an algorithm $\alpha?$ such that, for all languages L , if L is generated by some CFG of length at most i , then $\alpha_L(i)$ is such a CFG?

The answers to Q1 and Q2 are no and yes, respectively, and they can be found already in Moore [9, Theorem 2 and 9]. We sketch the essence of Moore's argument with respect to Q1. Assume by contradiction that $\alpha?$ as in Q1 exists. Let L be the empty regular language. Assume the algorithm α_L consults the oracle for L on words w_1, \dots, w_n . Let w be a word which is different from all w_i ($1 \leq i \leq n$) and let L' be the regular language $\{w\}$. As the answers of the oracles for L and L' are the same for all words w_1, \dots, w_n , we must have $\alpha_L() = \alpha_{L'}()$, which is obviously wrong. Therefore such $\alpha?$ cannot exist.

Note that we do not know in advance how many states a DFA accepting L' must have. This provides a clue to the positive answer to Q2. We will answer Q3 negatively in Section 4, but this requires a stronger argument than Moore's.

In the above questions, preconditions like 'there exists an accepting DFA for L having at most i states' disclose information about the solution and are referred to as a 'promise'. The question itself is referred to by the phrase 'promise problem'. The promise problem is of interest in the field of parameterized complexity (i can be taken as a parameter), and has been linked to the search for forbidden minors in graph classes. In their book on parameterized complexity [5], Downey and Fellows introduce a technique named Method of Testsets for solving the promise problem.

In the next section we shall state and answer a white box version of Q1, while in Section 3 we give an algorithm for Q2 which improves on previous algorithms. We show how the problem generalizes to context-free languages in Section 4, but that a sufficient counterpart to the promise on states is not easily found, and indeed in a quite reasonable sense *cannot* be found. In Section 5 we present some positive results on subclasses of the context-free languages.

We shall occasionally use some recursion theory. We use ϕ_x to denote the x th partial recursive function in some standard enumeration. Termination (non-) on input y will be denoted by $\phi_x(y)\downarrow$ (\uparrow) and the *domain* of ϕ_x , the set of all y such that $\phi_x(y)\downarrow$, by $\text{dom}(\phi_x)$. A set $S \subseteq \mathbb{N}$ is decidable if there exists a *total* recursive function ϕ such

that, for all $x \in \mathbb{N}$, $\phi(x) = 0$ if and only if $x \in S$. A set $S \subseteq \mathbb{N}$ is *semi*-decidable if there exists a *partial* recursive function ϕ such that, for all $x \in \mathbb{N}$, $\phi(x) = 0$ if and only if $x \in S$. Thus the crucial difference between decidable and semi-decidable is the possible non-termination of $\phi(x)$ in case $x \notin S$. The most famous semi-decidable set which is not decidable is $\{x \in \mathbb{N} \mid \phi_x(x) \downarrow\}$, and this fact is usually referred to as the unsolvability of the Halting Problem. We will use the following version of Kleene's *s-m-n*-Theorem: given a partial recursive function ϕ of two arguments, there exists a total recursive function s such that, for all $x, y \in \mathbb{N}$, $\phi_{s(x)}(y) \downarrow$ if and only if $\phi(x, y) \downarrow$ and if $\phi(x, y) \downarrow$, then $\phi_{s(x)}(y) = \phi(x, y)$. This fact is also referred to as the Parameter Theorem, and it says that we can fix an argument of a partial recursive function and obtain a new partial recursive function whose index can be computed uniformly in the fixed argument (the parameter). For more information on recursion theory we refer to [11].

2 Regular languages

In [5] a white box version of Q1 is considered and the following result is stated.

Observation 6.27 *In general, there is no algorithm to deduce an automaton for a regular language from a machine description of the language.*

Let us review the proof. By Kleene's *s-m-n*-Theorem there exists a total recursive function s such that $\phi_{s(x)}(y) = 1$ if $\phi_x(x) \downarrow$ and $\phi_{s(x)}(y) \uparrow$ otherwise. Let $V_x = \text{dom}(\phi_{s(x)})$. Then, V_x is either \emptyset or \mathbb{N} and $V_x = \mathbb{N}$ iff $\phi_x(x) \downarrow$. Since emptiness of a regular language, given a DFA, can be decided, but the Halting Problem cannot, such DFAs cannot be computed uniformly for all V_x .

Observe that only the languages \emptyset and \mathbb{N} play a role in the proof, and that they can be accepted by DFAs with just one single state. Now if the 'machine descriptions' of these, referred to in the above observation, are *decision procedures*, then they provide suitable oracles, and hence by the positive answer to Q2 (for the case $i = 1$) appropriate DFAs can indeed be produced.

How can this paradox be solved? The crucial point is that there exists only a *semi*-decision procedure for membership of $V_x = \text{dom}(\phi_{s(x)})$, and not a decision procedure. Hence by a 'machine description' in the above observation we will have to understand a semi-decision procedure.¹ Below we shall consider a different white box version of Q1 where 'machine description' is reinterpreted to mean a decision procedure.

Definition 1 *Let $L \subseteq \{1\}^*$ be a language and let $l \in \mathbb{N}$. We say that l is a machine description of L if ϕ_l is total recursive and, for all $y \in \mathbb{N}$, $\phi_l(y) = 0$ if and only if $1^y \in L$.*

The above definition can easily be adapted to languages L over Σ , using some standard enumeration of Σ^* . Also, we identify DFAs, being finite objects, with natural numbers via some standard encoding.

¹This understanding is in fact contradicted elsewhere in [5], where 'machine descriptions' are equated with decision procedures, but the proof given of Observation 6.27 only suffices for the weaker version where 'machine descriptions' can be semi-decision procedures.

Observation 6.27 can now be reformulated as follows.

Theorem 2 *There exists no partial recursive function α such that for all regular languages L , if l is a machine description of L , then $\alpha(l)$ encodes a DFA accepting L .*

Proof. We prove this for $\Sigma = \{1\}$, but the argument can easily be generalised. Define for every $x \in \mathbb{N}$:

$$L_x = \{1^y \mid \phi_x(x) \downarrow \text{ in exactly } y \text{ steps}\}$$

Note that L_x is empty if $\phi_x(x)$ diverges, and is a singleton language otherwise. Define a total recursive function f of two arguments by

$$f(x, y) = \begin{cases} 0 & \text{if } \phi_x(x) \downarrow \text{ in exactly } y \text{ steps} \\ 1 & \text{otherwise} \end{cases}$$

By Kleene's s - m - n -Theorem there exists a total recursive function s such that $\phi_{s(x)}(y) = f(x, y)$ for all x, y . Note that $\phi_{s(x)}$ is total recursive since f is. For every x , $s(x)$ is a machine description of L_x . Now we can finish the proof by contradiction, reducing the Halting Problem to a decidable problem. Assume α is partial recursive and, for every x , $\alpha(s(x))$ encodes a DFA accepting L_x . Then we would have $\phi_x(x) \downarrow$ iff $L_x \neq \emptyset$ iff $\alpha(s(x))$ has a reachable final state. The latter problem can easily be decided, which conflicts with the undecidability of the Halting Problem. \square

In the above reduction of the Halting Problem, $s(x)$ and L_x are notably different from $s(x)$ and V_x in the proof of Observation 6.27 from [5]. In particular there is no bound on the numbers of states of the DFAs accepting the singleton languages L_x .

Theorem 2 (white box) is actually stronger than just a negative answer to Q1 (black box) from the introduction, since α may use *all information* from the machine description l of L . The algorithm $\alpha?$ in Q1 can only do membership tests. If $\alpha?$ would exist, then $\alpha_{\phi_l}()$ would be a partial recursive function (of l) contradicting Theorem 2. We think that the black box formulation best reflects the nature of the Method of Testsets.

Moore [9, Theorem 9] answers Q2 from the introduction affirmatively, by an argument based on the so-called minimum state DFA. Here we elaborate the proof as in [5, Theorem 6.28], which is based on the Myhill-Nerode Theorem [10]. This proof uses the well-known right-invariant equivalence relation $x \sim_L y$ iff for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$. Here and below concatenation of strings is denoted by juxtaposition. By Myhill-Nerode L is regular if and only if \sim_L has finite index, and in that case an accepting DFA can be constructed as follows. The states are equivalence classes modulo \sim_L , the initial state being the class $[e]$ of the empty string and the final states being those that have a representative in L . The next-state function is given by $\delta([x], a) = [xa]$. This definition actually yields the minimum state DFA. If we have the extra information that there exists a DFA accepting L having at most k states, then the DFA constructed above has at most k states and all these states can be reached in less than k steps, so we can restrict attention to representatives of length $< k$. For deciding equivalence we can use the fact [7, Exercise 4.4.3]

that there exist short strings witnessing that two representatives are not equivalent. More precisely, if $x \not\sim_L y$, then there exists $z \in \Sigma^*$ such that $xz \in L \iff yz \notin L$ with $|z| \leq k - 2$, where k is the number of states. Thus the accepting minimum state DFA can be computed.

As the above method only uses an oracle for L and not the internals of the machine description of the language, it does not only prove a white box version but also answers the black box version of Q2 affirmatively, which is a slightly stronger result. Furthermore, the affirmative answer to Q2 shows that it is more informative to know a bound on the number of states than to have full access to the code of a decision procedure for the language when no such bound has been disclosed (cf. Theorem 2).

In the next section we present an algorithm based on the above method which is slightly better than Angluin's [2, Corollary 6], which in turn is better than Downey and Fellows' [5, proof of Theorem 6.28].

3 On-the-fly algorithm

As indicated in the previous section there are several algorithms for Q2. In [2, Corollary 6] it is shown that the algorithm ID solves Q2 with $O(k|\Sigma|^k)$ queries to the oracle, in [3] it is shown that Q2 can even be solved with a polynomial number of queries if the oracle can also answer questions on equivalence. We give an algorithm for Q2 that improves on the time complexity of the ID algorithm.

Our algorithm is based on determining and storing information about equivalence classes using a tree structure. By the sharp solution to [7, Exercise 4.4.3, p.164] it is sufficient to test suffixes of length $\leq k - 2$ to determine whether two strings are equivalent. Since the answers for any two strings in the same equivalence class are the same (obviously), the sequence of answers uniquely determines each equivalence class. We refer to this sequence of answers as the equivalence class' *signature*.

Definition 3 *Given Σ and k , let z_1, z_2, \dots, z_l be some fixed but arbitrary enumeration of all strings of length $\leq k - 2$ over Σ . Here $l = 1 + |\Sigma| + \dots + |\Sigma|^{k-2} = \frac{|\Sigma|^{k-1} - 1}{|\Sigma| - 1} = O(|\Sigma|^{k-2})$.*

Definition 4 *Let L be a language which is accepted by a DFA with at most k states and let $L(x) = \text{True}$ if $x \in L$ and $L(x) = \text{False}$ otherwise. Recall the equivalence relation \sim_L . The signature $\sigma_{[y]}$ of an equivalence class $[y]$ is the sequence of booleans $\langle L(xz_1), L(xz_2), \dots, L(xz_l) \rangle$ any representative $x \in [y]$ will produce. A signature tree is a binary tree of height l representing the signatures of all equivalence classes. A branch on level i labelled *True* (*False*) means that suffixing x with z_i does (doesn't) result in a string in L . The signature of any $x \in \Sigma^*$ can be viewed as a path in the signature tree and stored by labelling the appropriate leaf in the signature tree with $[x]$.*

The idea of the algorithm is the following. We will construct the signature tree iteratively, at each stage we determine parts of the transition-function δ until it is

totally defined. If there are k states, there are no more than k equivalence classes, thus the maximum size of the tree is limited.

1. Compute the signature for $[\epsilon]$, create the corresponding leaf in the signature tree and label it $[\epsilon]$.
2. If there exists a leaf labelled $[x]$ in the signature tree such that, for some $a \in \Sigma$, there is no value defined for $\delta([x], a)$, then compute the signature of xa . If the signature of xa leads to a previously created leaf $[y]$, then define $\delta([x], a) = [y]$. Otherwise, create a new leaf corresponding to the signature, label it $[xa]$ and define $\delta([x], a) = [xa]$.
3. Repeat 2 until δ is totally defined.

This determines the states and transitions of the automaton. Moreover, $[\epsilon]$ is the initial state, while the accepting states are those corresponding to signatures starting with *True*. (Provided the suffix ϵ is tested first in the signature tree, i.e., $z_1 = \epsilon$.) This concludes the construction of a DFA accepting L .

The above algorithm has been implemented and is described in detail in Sloper [15]. Our algorithm uses k paths in the signature tree, each $O(|\Sigma|^{k-2})$ long, using a total of $O(k|\Sigma|^{k-2})$ space. During the construction of the automaton the signatures of $k|\Sigma| + 1$ strings are computed, which adds up to $O(k|\Sigma|^{k-1})$ queries, thus improving by a factor of $|\Sigma|$ on the time complexity of the ID algorithm in [2].

We will now elaborate an example given an oracle L for the regular language c^*a over $\Sigma = \{a, b, c\}$ and the promise that there exists a DFA with 3 states accepting this language.

As noted before we need only consider suffix-strings of length at most $k - 2 = 1$, i.e. the strings ϵ, a, b, c , when determining equivalence. First we determine the signature $\sigma_{[\epsilon]}$ for the equivalence class $[\epsilon]$, $\sigma_{[\epsilon]} = \langle L(\epsilon), L(a), L(b), L(c) \rangle = \langle \text{False}, \text{True}, \text{False}, \text{False} \rangle$, the signature tree is seen in Figure 1(a).

We proceed to define the transition function δ . We note that $\delta([\epsilon], x)$ is not yet defined for any $x \in \Sigma$. We determine signatures of $[a]$, $[b]$, and $[c]$. $\sigma_{[a]}$ and $\sigma_{[b]}$ create new leaves while $\sigma_{[c]} = \sigma_{[\epsilon]}$. The transition function δ can now be updated $\delta([\epsilon], a) = [a]$, $\delta([\epsilon], b) = [b]$, and $\delta([\epsilon], c) = [\epsilon]$.

Since two new equivalence classes $[a]$ and $[b]$ were discovered, we have found all $k = 3$ states, but the transition function is not yet totally defined. The full signature tree can be seen in Figure 1(b). For completing the transition function a further optimization can be used: already at the dotted nodes in Figure 1(b) it is decided in which equivalence class one will end up. We need to define $\delta([a], x)$ and $\delta([b], x)$ for all $x \in \Sigma$. Consider the classes $[aa]$, $[ab]$, $[ac]$, $[ba]$, $[bb]$ and $[bc]$. One discovers quickly (by testing suffixes ϵ and a) that $\sigma_{[aa]} = \sigma_{[ab]} = \sigma_{[ac]} = \sigma_{[ba]} = \sigma_{[bb]} = \sigma_{[bc]} = \sigma_{[b]}$, thus $\delta([a], a) = \delta([a], b) = \delta([a], c) = \delta([b], a) = \delta([b], b) = \delta([b], c) = [b]$.

The transition function δ is now totally defined, which finishes the construction of the DFA, see Figure 2. Note that since the equivalence classes are defined by

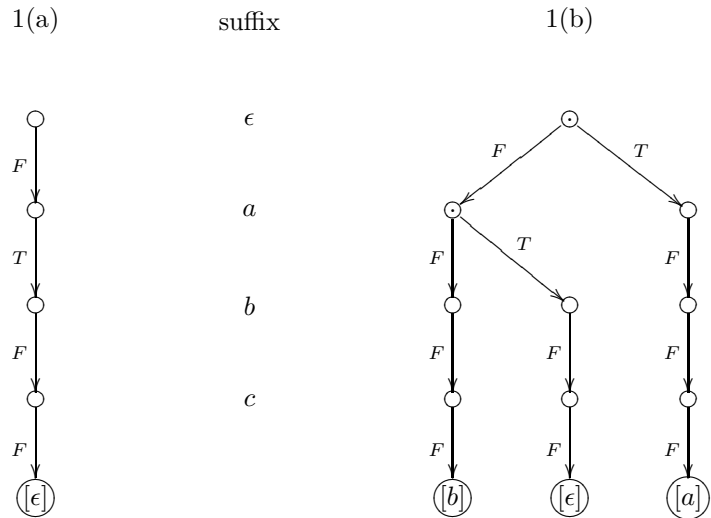


Figure 1: Signature trees at different stages of construction. In 1(a) only the signature of $[\epsilon]$ is known, while in 1(b) the signatures of all three classes are known.

the *canonical right congruence* \sim_L induced by L it follows from the Myhill-Nerode Theorem that the constructed automaton is a minimum state automaton. Thus, every state is reachable and there is at most one ‘sink-state’ (corresponding to the signature consisting of only *False*-answers).

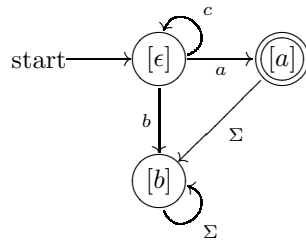


Figure 2: The automaton for c^*a over $\Sigma = \{a, b, c\}$.

4 Context-free languages, negative result

Extending the method of test sets to context-free languages can be expected to require a more powerful promise than in the regular case. Observe first that $L' = \{w\}$ in Moore’s argument with respect to Q1, see Section 1, can be generated by a CFG with just one rule $S \rightarrow w$. Alternatively, L' can be accepted by a PDA with just

one state if transitions can push arbitrarily many symbols on the stack. Following [7, Exercise 6.2.28], a PDA is called *restricted* if on any transition it can increase the height of the stack by at most one symbol. For every PDA there exists a restricted PDA accepting the same language and language equivalence for restricted PDAs is undecidable like for PDAs. Even restricted PDAs can accept $L' = \{w\}$ with just one state if the stack alphabet contains at least $|w|$ symbols (actually, instead of a stack, a read/write register would suffice). As an ad-hoc notion we define the *size* of a PDA to be the sum of the number of states and the number of stack symbols. Now for any k there are only finitely many non-isomorphic restricted PDAs of size at most k . Still (in contrast to the positive answer to Q2) we have the following negative result.

Theorem 5 *There is no algorithm $\alpha_?$ such that, for all languages L , if L is accepted by a restricted PDA of size at most k , then $\alpha_L(k)$ is a PDA accepting L .*

Proof. Assume by contradiction that $\alpha_?$ exists as above. Observe that, under this assumption, $\alpha_{L(P)}(k)$ is unique for all restricted PDAs P accepting the same language and having size at most k . Moreover, since membership of $L(P)$ is decidable uniformly in the PDA P , we can view $\alpha_{L(P)}(k)$ as an algorithm with input P and k , not using an oracle. This allows us to solve the language equivalence problem for restricted PDAs, which is undecidable, thereby obtaining a contradiction. For this last step, consider restricted PDAs P_1 and P_2 of sizes m_1 and m_2 and defining context-free languages $L(P_1)$ and $L(P_2)$, respectively. Let $k = \max(m_1, m_2)$. Then we have

$$L(P_1) = L(P_2) \iff \alpha_{L(P_1)}(k) = \alpha_{L(P_2)}(k)$$

The latter is decidable, since it involves only syntactical inspection of the finite objects $\alpha_{L(P_i)}(k)$. \square

There are many variations of Theorem 5. Promises like: L is defined by a CFG of length at most k (such as in Q3, Section 1), and similar conditions on Chomsky normal forms or on linear CFGs, can all be settled by similar arguments. In fact, by taking $k = \max(f(P_1), f(P_2))$ in the proof of Theorem 5 one can prove the following theorem. Here and below, the neutral phrase *device for language definition* has been chosen to refer to both machines accepting and grammars generating languages. Given such a device M , the language defined will be denoted by $L(M)$.

Theorem 6 *Let \mathcal{M} be a class of finite devices for language definition such that membership of $L(M)$ is decidable uniformly in $M \in \mathcal{M}$, but language equivalence in \mathcal{M} is undecidable. Let f be an arbitrary computable function from \mathcal{M} to the natural numbers. Then there is no algorithm $\alpha_?$ such that, for all languages L , if L is defined by some $M \in \mathcal{M}$ with $f(M) \leq k$, then $\alpha_L(k) \in \mathcal{M}$ defines L .*

5 Positive results on two classes of context-free languages

Recently, Sénizergues [13] has proved the decidability of language equivalence for deterministic PDAs, which has been an open problem for over 30 years. This highly

technical result allows us to obtain some positive results on the method of test sets in a rather easy way. First we state and prove a general result.

Theorem 7 *Let \mathcal{M} be a class of finite devices for language definition such that*

1. *membership of $L(M)$ is decidable uniformly in $M \in \mathcal{M}$, and*
2. *language equivalence of devices in \mathcal{M} is (semi-)decidable.²*

Then there exists an algorithm α_L such that, for all languages L , if there are finitely many devices, say $M_1, \dots, M_m \in \mathcal{M}$, such that at least one of them accepts L , then $\alpha_L(M_1, \dots, M_m) = M_i$ and $L = L(M_i)$ for some $1 \leq i \leq m$.

Proof. Recall the alphabet Σ and let z_1, z_2, \dots be an enumeration of Σ^* . Run in parallel:

1. tests $z_n \in L(M_i)$ for every $n > 0$ and $1 \leq i \leq m$;
2. semi-decision procedures testing $L(M_i) = L(M_j)$ for all $1 \leq i \neq j \leq m$.

After finitely many steps in the parallel computation we find for every $1 \leq i \neq j \leq m$ either a word $z_{n(i,j)}$ which allows us to distinguish between L_i and L_j , or a proof of $L(M_i) = L(M_j)$. Consulting the oracle for L on all those $z_{n(i,j)}$ and comparing the ‘signature’ of L with those of the $L(M_i)$ enables us to put $\alpha_L(M_1, \dots, M_m) = M_i$ with $L(M_i) = L$. \square

Condition (1) of this theorem is fulfilled for PDAs. The simplified proof of the decidability of DPDA equivalence by Stirling [16] gives a tableaux proof procedure meeting precisely condition (2) of Theorem 7. Let us fix a finite stack alphabet Γ . Transitions of a restricted DPDA are of the form $\delta(q, a, X) = (p, \gamma)$ with $X \in \Gamma$, $\gamma \in \Gamma^*$ and $|\gamma| < 3$. As a consequence, there are only finitely many restricted DPDAs with at most k states and we can apply Theorem 7 in order to prove the following theorem. Recall that the size is the number of states plus the number of stack symbols.

Theorem 8 *There exists an algorithm α_L such that for all languages L , if L is accepted by a restricted DPDA of size at most k , then $\alpha_L(k)$ is such a DPDA accepting L .*

Note that the qualifier ‘restricted’ is essential here. For DPDAs that can push arbitrarily many symbols on the stack the theorem is refuted by Moore’s argument with respect to Q1 from the introduction.

The algorithm suggested by Theorem 7 is rather impracticable. Observe that Theorem 7 also answers Q2 from Section 1 positively, since the machine class of DFAs satisfies both conditions (1) and (2). However, the algorithm from Section 3 is much better than the algorithm suggested by Theorem 7.

²In view of condition 1, requiring decidability or semi-decidability makes no difference here.

Theorem 8 settles positively an analogue of Q2 for deterministic context-free languages, but this class by no means contains all of the simpler context-free languages. Consider for example $\{ww^R \mid w \in \{a,b\}^*\}$, which is not recognized by any DPDA. We can, however, stretch the techniques for *regular* languages to such palindrome languages and various other simple non-regular context-free languages such as $\{a^n b^n \mid n \in \mathbb{N}\}$. These languages are called ‘even linear’ as they have a CFG with all rules of the form $N \rightarrow a$, $N \rightarrow \epsilon$ or $N_1 \rightarrow aN_2b$. The even linear languages have been studied extensively in the literature, see [1, 17, 12, 8]. The class of even linear languages extends the class of regular languages and has several nice closure properties, such as closure under finite boolean operations. Language equivalence is decidable for even linear grammars. Be careful to distinguish even linear languages from linear CFLs; the linear (and deterministic) language $\{a^{2^n} b^n \mid n \in \mathbb{N}\}$ is, for example, not even linear.

Takada [17] showed how to extend techniques for regular languages to even linear languages, using the notion of ‘control sets.’ Stripping away some of the theoretical niceties, the crucial idea is captured by an operation [12, 8] that ‘folds’ rules like $N_1 \rightarrow aN_2b$ into the regular format $N_1 \rightarrow \frac{a}{b}N_2$, introducing a new terminal $\frac{a}{b}$, while preserving rules like $N \rightarrow a$ and $N \rightarrow \epsilon$. The result is a simple right-linear grammar that defines a regular language L_f over alphabet $\Sigma \cup \{\frac{a}{b} \mid a,b \in \Sigma\}$, which is the image of the original even linear language L under a similar ‘folding’ operation on words that takes, for instance, $abcde$ to $\frac{a}{e}\frac{b}{d}c$ and $abcdef$ to $\frac{a}{f}\frac{b}{e}\frac{c}{d}$.

Techniques for regular languages can then be applied to the ‘folded’ even linear language. We conclude with the result that ‘folding’ would give us here.

Theorem 9 *There exists an algorithm $\alpha_?$ such that for all languages L , if L is generated by an even linear CFG having at most k non-terminals, then $\alpha_L(k)$ is such a CFG accepting L .*

We illustrate the general strategy with an example. Given the promise that some linear language L (in fact, unknown to us, $\{a^{n+1}b^n \mid n \geq 0\}$) over alphabet $\{a,b\}$ is produced by an even linear grammar with a single non-terminal, we know that the corresponding ‘folded’ language L_f over alphabet $\{a,b,\frac{a}{a},\frac{a}{b},\frac{b}{a},\frac{b}{b}\}$ is produced by a corresponding simple right-linear grammar with one non-terminal, hence also by an NFA of at most $1 + 1 = 2$ states, and hence by a DFA of at most $2^2 = 4$ states. Now by the positive answer to Q2 there exists an algorithm (written for the alphabet $\{a,b,\frac{a}{a},\frac{a}{b},\frac{b}{a},\frac{b}{b}\}$) which when given an oracle³ for L_f and the input 4 (representing the promise of 4 states) produces an appropriate DFA for L_f . As described in Section 3, this will in fact be a minimum state DFA, meaning in particular that it contains at most one state from which no accepting state is reachable. We refer to such a state as a ‘sink state.’ When the sink state and all transitions leading into it are suppressed, the particular output DFA of the example takes the form seen in Figure 3.

Using a general strategy for conversion to right-linear grammars, this yields a CFG with start symbol S_1 and productions $S_1 \rightarrow \frac{a}{b}S_1$, $S_1 \rightarrow aS_2$ and $S_2 \rightarrow \epsilon$, and thus

³Such an oracle for L_f is easily obtained from an oracle for L in the following way: if the input word contains some occurrence of a non-composite terminal (i.e., a or b) anywhere but in the final position, then reply negatively; else ‘unfold’ and pass the result on to the oracle for L .

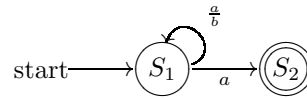


Figure 3: Minimum state DFA for $L_f = \frac{a}{b}^*a$, with ‘sink state’ suppressed.

the grammar⁴ with productions $S_1 \rightarrow \frac{a}{b}S_1$, $S_1 \rightarrow a$ and $S_2 \rightarrow \epsilon$, which unfolds to the even linear grammar with productions $S_1 \rightarrow aS_1b$, $S_1 \rightarrow a$ and $S_2 \rightarrow \epsilon$.

An easy inspection reveals that S_2 and the production involving it are redundant, hence an equivalent even linear grammar with a single non-terminal can be returned.⁵

References

- [1] V. Amar, G. Putzolu, “On a Family of Linear Grammars”, *Information and Control*, bf 7, pp. 283-291, 1964.
- [2] D. Angluin, “A Note on the Number of Queries Needed to Identify Regular Languages”, *Information and Control*, 51, pp. 76-87, 1981.
- [3] D. Angluin, “Learning Regular Sets from Queries and Counterexamples”, *Information and Computation*, 75, pp. 87-106, 1987.
- [4] B. Beizer, *Software Testing Techniques*, 2nd edition, Van Nostrand Reinhold, 1990.
- [5] R. Downey, M. Fellows, *Parameterized Complexity*, Springer-Verlag, 1997.
- [6] M.E. Gold, “Language identification in the limit”, *Information and Control*, 10, pp. 447-474, 1967.
- [7] J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Second Edition, Addison-Wesley, 2001.
- [8] T. Langholm, M. Bezem, “A Descriptive Characterisation of Even Linear Languages”, *Grammars*, 6(3), pp. 169-181, 2003.

⁴The general strategy is to replace any production $A \rightarrow aB$, when a is a non-composite terminal, by $A \rightarrow a$. As described in [8, Section 3], this is safe when the grammar is obtained from a minimum state DFA with sink state suppressed, as this will yield a CFG without useless terminals, hence if $A \rightarrow aB$ is in the grammar (keeping in mind that the grammar produces a folded language) then from B the empty, and no other, terminal string can be derived.

⁵In general, the method outlined may not produce a grammar with a minimal number of non-terminals, since the output grammar is derived from a DFA, which in general may contain exponentially more states than a corresponding NFA. To obtain the minimal grammar, one might compare the output DFA with all NFAs of fewer states, and then proceed with a minimum state NFA. Using such a method, one would of course have to renounce all claims of efficiency.

- [9] E.F. Moore, “Gedanken-experiments on sequential machines”, C.E. Shannon, J. McCarthy, W.R. Ashby (editors), *Automata Studies*, Annals of mathematics studies 34, pp. 129-153, Princeton University Press, 1956.
- [10] A. Nerode, “Linear bounded automata”, *Proceedings AMS*, 9, pp. 541-544, 1958.
- [11] P. Odifreddi, *Classical Recursion Theory*, North-Holland, 1990.
- [12] J.M. Sempere, P. García, “A Characterization of Even Linear Languages and its Application to the Learning Problem”, *Proceedings of the Second International Colloquium*, ICGI-94. LNAI, 862, pp. 38-44, Springer-Verlag, 1994.
- [13] G. Sénizergues, “The equivalence problem for deterministic pushdown automata is decidable”, P. Degano et alii (eds.), *Proceedings ICALP '97*, LNCS 1256, pp. 671-681, Springer-Verlag, 1997.
- [14] M. Sipser, *Introduction to the Theory of Computation*, PWS Publ. Company, 1997.
- [15] C. Sloper, *Parameterized Complexity and the Method of Testsets*, MSc thesis, Bergen University, 2001.
- [16] C.P. Stirling, “Decidability of DPDA equivalence”, *Theoretical Computer Science*, 255, pp. (1-2, 1-31), 2001.
- [17] Y. Takada, “Grammatical Inference for Even Linear Languages based on Control Sets”, *Information Processing Letters*, 28, pp. 193-199, 1988.