Learning Algorithms

Henrik Björklund[1], *Johanna Björklund*[1], *Wim Martens*[2]

[1] Umeå University
Department of Computing Science
90187 Umeå, Sweden

[2] Institut für Informatik
Universität Bayreuth
95440 Bayreuth, Germany

email: henrikb@cs.umu.se, johanna@cs.umu.se, wim.martens@uni-bayreuth.de

2010 Mathematics Subject Classification: 68Q32, 68Q45, 68T05

Key words: Finite automata, regular languages, algorithmic learning.

Contents

1 Introduction

What is a learning algorithm? How can a computer automatically learn things and somehow become smarter as it receives more information? According to Valiant, "a program for perfoming a task has been acquired by *learning* if it has been acquired by any means other than explicit programming" [42]. This view on learning is very broad, as is the field of learning algorithms. Learning algorithms, or *learners*, can be said to specialize in generalizing from *instances* to *concepts*. At the heart of learning lies the ability to derive from a number of instances the common concept that they examplify. The concept can, in principle, be almost anything. If it is a logical formula, the instances may be structures that satisfy it. If it is a language, the intances may be words that belong to it.

Grammatical inference is the subfield of algorithmic learning where the concept to be learned is a formal language. Since languages can be infinte, we are interested in learners that output finite representations of the languages such as *automata*. Even grammatical inference is, however, a field much too vast to be covered here, and we therefore focus on what we believe to be the most fundamental aspect of it, namely the learning of *regular* languages, represented by finite automata.

After some preliminary definitions, we present the most well-known classical results on the learning of deterministic finite automata in Section 3. These results have been taken up by many researchers and extended to the learning of, e.g., nondeterministic finite automata, and regular tree automata. We discuss these extensions in Sections 4–6. The concept of *probably approximately correct* (PAC) learning is important in general learning theory, but arguably less so in grammatical inference. We do, however, discuss its implications for learning finite automata in Sectiton 7. Finally, in Section 8, we present a few examples of more applied settings in which grammatical inference has been successfully used, such as natural language processing, XML databases, and formal verification.

2 Preliminaries

We denote the set of Booleans by $\mathbb{B} = \{0, 1\}$ where 1 represents true and 0 represents false. We use $\mathbb{N}$ for the natural numbers and $\mathbb{N}_+$ for the strictly positive natural numbers. For $k \in \mathbb{N}$, we denote the set $\{1, \ldots, k\}$ by $[k]$, with $[0] = \emptyset$. Given a partial function $f\colon D \to D'$, we denote by $\mathit{Dom}(f)$ the *domain of* f, that is, the subset of D on which f is defined. We write $\mathit{Rng}(f)$ for the *range of* f, that is, the set $\{d' \mid (d' \in D') \wedge (\exists d \in D$ such that $f(d) = d')\}$.

In the following, A always denotes a finite alphabet. A set S of words over alphabet A is *prefix-closed* if $w \cdot x \in S$ implies that $w \in S$, for all $w, x \in A^*$. The set is *suffix-closed* if $w \cdot x \in S$ implies that $x \in S$.

We often abbreviate *(nondeterministic) finite automaton* and *deterministic finite automaton* by *NFA* and *DFA*, respectively. Given an NFA $\mathcal{A} = (Q, I, E, T)$ and a set $S \subseteq Q$, we write $E(S)$ for the set of *successors* of states in S with respect to E, that is, $E(S) = \{p \mid \exists q \in S \wedge \exists a \in A : (q, a, p) \in E\}$. We write $E(S, a)$ for the set $\{p \mid \exists q \in S$ with $(p, a, q) \in E\}$. For a word w we denote by $E^*(w)$ the set of states that can be reached by reading w. That is, $E^*(\varepsilon) = I$ and $E^*(wa) = E(E^*(w), a)$. The language of $\mathcal{A}$ is denoted $L(\mathcal{A})$. By $\mathcal{A}(w)$ we denote the Boolean value that is true if and only if $w \in L(\mathcal{A})$.

Given a set $S = \{w_1, \ldots, w_n\}$ of words, the *prefix tree acceptor*, or *PTA* of S is the DFA $\mathcal{A} = (Q, I, E, T)$ such that $L(\mathcal{A}) = S$ and Q is the set of all prefixes of words from S, $I = \varepsilon$, $T = S$, and $E = (w, a, wa)$ for each state wa in Q.

2.1 A warm-up to learning

At the heart of many learning algorithms for regular languages lies the Myhill-Nerode theorem. We therefore remind the reader of its statement.

Definition 2.1. Let X be a language over alphabet A. Two words w_1, w_2 in A^* are *equivalent with respect to* X, written $w_1 \equiv_X w_2$, if, for every word $x \in A^*$, $w_1 \cdot x \in X$ if and only if $w_2 \cdot x \in X$. For a word $w \in A^*$, we write $[w]_{\equiv_X}$ for the equivalence class of w in the equivalence relation $\equiv_X$, that is, for the set of words $\{y \in A^* \mid y \equiv_X w\}$. We say that w is a *representative* of class $[w]_{\equiv_X}$.

Theorem 2.1 (Myhill-Nerode). *A language X is regular if and only if $\equiv_X$ has finite index, that is, has a finite number of equivalence classes. Furthermore, if X is regular, then each state of the minimal DFA for X corresponds to an equivalence class of $\equiv_X$ and vice versa. In particular, the index of $\equiv_X$ is exactly the number of states of the minimal DFA for X.*

We provide a general example as a warm-up for the reader to the general philosophy behind many algorithms for learning regular languages.

Example 2.1. Assume that we are given the following information about an unknown word language Z over $A = \{a, b\}$.

- The equivalence relation $\equiv_Z$ has four classes, S_1, S_2, S_3, and S_4.
- The words ε, a, and baa belong to S_1.
- The words b and bab belong to S_2.
- The words ba and bba belong to S_3.
- The words bb and bbb belong to S_4.
- $(S_3 \cup S_4) = Z$.

Due to Theorem 2.1, we know that we are looking for a regular word language. As we shall see, this information is enough to recreate the minimal DFA $\mathcal{A}_Z = (Q, I, E, T)$ for Z. Consider the set $S = \{\varepsilon, b, ba, bb\}$ of the shortest representatives we know for each equivalence class of $\equiv_Z$. Notice that, for every $w \in S$ and every $c \in A$, we know to which equivalence class $w \cdot c$ belongs. Furthermore, since $w \equiv_Z x$ implies that $w \cdot c \equiv_Z x \cdot c$

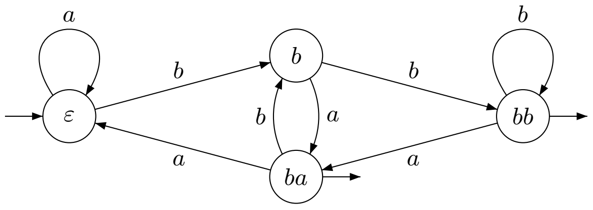


Figure 1. The automaton $\mathcal{A}_Z$ from Example 2.1. Each state is labeled by its shortest representative.

holds for all $w, x \in A^*$ and all $c \in A$, we know everything we need to know about the relationships of the equivalence classes to each other. If we let $Q = \{S_1, S_2, S_3, S_4\}$ as in the Myhill-Nerode theorem, we know, for example, that since $ba \in S_3$ and $bab \in S_2$, there should be a b-labeled edge from S_3 to S_2. Furthermore, since $\varepsilon \in S_1$ we have $I = \{S_1\}$ and since $(S_3 \cup S_4) = Z$ we have $T = \{S_3, S_4\}$. The full automaton $\mathcal{A}_Z$ is depicted in Figure 1. The language Z can also be characterized through the regular expression $(a + b)^*b(a + b)$.

2.2 Observation tables for word languages

Many learning algorithms use so-called *observation tables* to represent the information they have so far gathered about an unknown language X. In this section we recall the observation tables for words from Angluin [4] and we show how to generalize them for trees in Section 6.

An *observation table for words* is a tuple $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$, where

- S_{pre} is a nonempty, finite, prefix-closed set of words;
- S_{suff} is a nonempty, finite, suffix-closed set of words; and
- *Obs* is a finite function from $((S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}})$ to $\{0, 1\}$.

Here, the function *Obs* formalizes what is known about the unknown language X, i.e., $\mathit{Obs}(w) = 1$ if and only if $w \in X$.

An observation table can be organized in rows and columns. The rows are indexed by elements from $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A)$ and the colums are indexed by elements from S_{suff}. The entry for row x and column y is then equal to $\mathit{Obs}(x \cdot y)$. If $w \in (S_{\text{pre}} \cup S_{\text{pre}} \cdot A)$ then we denote by row_w the finite function from S_{suff} to $\{0, 1\}$ with $\mathit{row}_w(x) = \mathit{Obs}(w \cdot x)$.

An observation table is *closed* when for each word w in $S_{\text{pre}} \cdot A$ there exists a word x in S_{pre} such that $\mathit{row}_w = \mathit{row}_x$. It is *consistent* if whenever w and x are elements of S_{pre} with $\mathit{row}_w = \mathit{row}_x$, then, for all $a \in A$, we have that $\mathit{row}_{wa} = \mathit{row}_{xa}$.

It is useful to think of the elements of S_{pre} as potential equivalence classes of X, or, equivalently, as potential states of a DFA for X. If we have $w \in S_{\text{pre}}$ and $w \cdot a \in S_{\text{pre}}$, we know that there should be an a-labeled edge from the state represented by w to the state represented by $w \cdot a$. But what if $w \cdot a$ is not an element of S_{pre}? The elements of $(S_{\text{pre}} \cdot A) \setminus S_{\text{pre}}$ can then be thought of as representing the extra transitions needed in the following sense. If $w \in S_{\text{pre}}$, $w \cdot a \in (S_{\text{pre}} \cdot A) \setminus S_{\text{pre}}$, and $\mathit{row}_{wa} = \mathit{row}_x$, for some $x \in S_{\text{pre}}$, there should be an a-labeled edge from the state represented by w to the state represented by x.

With these correspondences in mind, we are better prepared to understand the significance of *closedness* and *consistency*. If the observation table is not closed, then there are transitions leading to unknown states. If, on the other hand, the observation table is not consistent, then the corresponding automaton is not deterministic.

Formally, the *finite automaton* $\mathcal{A}_{Obs} = (Q, I, E, T)$ *associated to a closed, consistent observation table* (S_{pre}, S_{suff}, Obs) is defined as follows:

- $Q = \{row_w \mid w \in S_{\text{pre}}\}$,
- $I = \{row_\varepsilon\}$,
- $E = \{(row_w, a, row_{wa}) \mid w \in S_{\text{pre}}$ and $a \in A\}$, and
- $T = \{row_w \mid w \in S_{\text{pre}}$ and $Obs(w) = 1\}$.

We argue that $\mathcal{A}_{Obs}$ is well-defined. We have that I is well-defined since S_{pre} is a nonempty prefix-closed set and therefore contains ε. To prove that T is well-defined, let w_1 and w_2 be two words in S_{pre} such that $row_{w_1} = row_{w_2}$. Since S_{suff} is nonempty and suffix-closed, S_{suff} always contains ε. Therefore, we have that $Obs(w_1) = Obs(w_1 \cdot \varepsilon) = row_{w_1}(\varepsilon) = row_{w_2}(\varepsilon) = Obs(w_2 \cdot \varepsilon) = Obs(w_2)$ and therefore T is well-defined. Finally, we show that E is well-defined. To this end, assume that w_1 and w_2 are two words in S_{pre} such that $row_{w_1} = row_{w_2}$. Since the observation table $(S_{\text{pre}}, S_{\text{suff}}, Obs)$ is consistent, we have that, for each $a \in A$, $row_{w_1 a} = row_{w_2 a}$. Furthermore, since $(S_{\text{pre}}, S_{\text{suff}}, Obs)$ is closed, $row_{w_1 a}$ and $row_{w_2 a}$ are equal to row_w for some word w in S_{pre}. This proves that $\mathcal{A}_{Obs}$ is well-defined.

The automaton $\mathcal{A}_{Obs} = (Q, I, E, T)$ associated to table $(S_{\text{pre}}, S_{\text{suff}}, Obs)$ is also *consistent with Obs* in the following sense. For every $w_1 \in (S_{\text{pre}} \cup S_{\text{pre}} \cdot A)$ and $w_2 \in S_{\text{suff}}$, we have that $E^*(w_1 \cdot w_2)$ is in T if and only if $Obs(w_1 \cdot w_2) = 1$. Furthermore, we note that any other DFA consistent with *Obs* but inequivalent to $\mathcal{A}_{Obs}$ must have more states than $\mathcal{A}_{Obs}$ (Theorem 1 in [4]).

3 Classical results

3.1 Learning in the limit

Gold's learning paradigm *learning in the limit* formalizes the view of human language acquisition as a process, in which the internal representation of the unknown language is continuously refined by new evidence, and converges towards an accurate model as time advances. In Gold's learning paradigm, an algorithm (henceforth, the *learner*) is to infer a formal representation an unknown language X. It is known that X is a subset of a certain universe $\mathcal{U}$, belongs to a class of languages $\mathcal{C}$, and that the learner needs to infer a formal representation from a certain hypothesis space $\mathcal{R}$. Suppose, for instance, that we are interested in learning regular word languages over the alphabet A. Then $\mathcal{C}$ would be the class of regular word languages, $\mathcal{U}$ would be set of words in A^*, and $\mathcal{R}$ could be the set of deterministic finite state automata, or the set of regular expressions over A.

In the presentation of Gold's paradigm, it will be convenient to identify a language X over the universe $\mathcal{U}$ of words with its *characteristic mapping*. That is, we also view X as a mapping $X : \mathcal{U} \to \mathbb{B}$ such that, for each word $w \in \mathcal{U}$, $X(w) = 1$ if $w \in X$ and $X(w) = 0$ otherwise. Similarly, we can view a *sample* of X on the subdomain $D \subseteq \mathcal{U}$ as the restriction $X|_D$ of X to D. Thus seen, $X|_D$ represents the annotated set of examples $\{(w, X(w)) \mid w \in D\}$. The *positive examples* contained in $X|_D$ are $X|_D^{-1}(1)$, and the *negative examples* are $X|_D^{-1}(0)$.

Additional information about X is provided to the learner at discreet time steps, according to a *presentation*, i.e., a function $g : \mathbb{N}_+ \to \mathcal{U}$. At time i, the learner is told whether the element $g(i)$ of $\mathcal{U}$ is in X or not. For every new piece of information, the learner must guess the identity of X by outputting an element in $\mathcal{R}$. This means that at time $i \in \mathbb{N}_+$, the learner may support its conjecture on membership information for the words in $\{g(1), \ldots, g(i)\}$. It is commonly assumed that the learner does not care about the order in which the examples are presented. From here on, we denote by $g[i]$ the first i examples provided by g, that is, the set $Rng(g|_{[i]})$.

Gold is primarily interested in two classes of presentations: *texts* and *informants*. A presentation g is a text for X if $Rng(g) = X$, and an informant for X if $Rng(g) = \mathcal{U}$. Thus, a *text* is restricted to *positive information*, which reflects the early linguistic hypothesis that children learn to speak by listening to others. When aided by an *informant*, the learner has access to *both positive and negative information*. This more expressive type of presentation can be justified by the fact that children also receive negative examples, e.g., when they try to speak, but fail to be understood.

Definition 3.1. Let $fin(\mathcal{U})$ be the set of all finite subsets of $\mathcal{U}$. A class of languages $\mathcal{C}$ is *learnable in the limit from an informant (from text)* if there is a computable function

$$\mathfrak{L} : \{X|_D \mid X \in \mathcal{C} \text{ and } D \in fin(\mathcal{U})\} \to \mathcal{R}$$

such that the following condition holds: for every $X \in \mathcal{C}$ and every informant $g : \mathbb{N}_+ \to \mathcal{U}$ (every text $g : \mathbb{N}_+ \to X$), there is an index $i \in \mathbb{N}_+$ and a representation $\mathcal{A} \in \mathcal{R}$ of X, such that $\mathfrak{L}(X|_{g[j]}) = \mathcal{A}$, whenever $j \geqslant i$.

Definition 3.1 does not require that the learner's initial conjectures are consistent with the given information, only that the conjectures eventually converge to the correct answer. A learner that only produces consistent conjectures is said to be *feasible*. In the remainder of this section, we briefly discuss learning from text versus learning from an informant.

3.1.1 Learning from text Early results regarding learning from text were discouraging. Angluin showed that no languae class with *infinite elasticity* is learnable in the limit from text [3]. The elasticity of a class is the length of the longest chain of inclusions $X_1 \subsetneq X_2 \subsetneq X_3 \subsetneq \cdots$, where $X_i \in \mathcal{C}$ for every $i \in \mathbb{N}$. The intuition behind Angluin's argument is the following. Towards a contradiction, assume that there is a learner that can infer $\mathcal{C}$ from a text presentation. If the learner is given a presentation g that begins with examples taken from X_1, then there is an index $i_1 \in \mathbb{N}$ such that after seeing the first i_1 examples, the learner will make the conjecture X_1. This is necessary because $X_1 \in \mathcal{C}$ and we assumed that the learner can identify every member of $\mathcal{C}$ in the limit. But assume that after index $i_1 + 1$, g contains a sequence of examples from X_2. Again, since $X_2 \in \mathcal{C}$, there is an index i_2 such that after seeing i_2 examples, the learner will conjecture L_2. Now starting at index $i_2 + 1$, g contains a sequence of examples from X_3, and so forth. Such an unfavorable presentation will always exist, and it will force the learner to renew its conjecture infinitely many times, making convergence impossible. This contradicts our assumption. It follows that no class of languages containing the finite languages and a single infinite language is learnable in the limit from text. There are, however a few positive results, e.g., it is known that languages of fixed cardinality can be (trivially) inferred from text. Further positive results are briefly discussed in Section 8.

3.1.2 Learning from an informant Whereas most language classes cannot be learned from text, even the primitive recursive languages can be learned from an informant using a technique called *identification by enumeration* [24]. This technique is applicable whenever the hypothesis space consists of a recursively enumerable family of representations $\mathcal{R}$ and the problem of deciding whether a particular representation is consistent with a restriction $X|_D$ of some language X to some finite domain D is decidable. The identification rule at time i reads as follows: enumerate the representations in $\mathcal{R}$ in order of size (using the lexicographical order to resolve ties) and take as conjecture the first (an hence the smallest) representation that is consistent with $X|_{g[i]}$. Since the minimal representation $\mathcal{A}$ of X will eventually be reached, the learner only has to modify its hypothesis a finite number of times before finding $\mathcal{A}$. Once the learner has made the conjecture $\mathcal{A}$, the addition of new examples will not cause an inconsistency, so the learner will never be compelled to discard the correct answer.

3.1.3 Characteristic sets A mapping $char_{\mathfrak{L}} : \mathcal{C} \to \mathit{fin}(\mathcal{U})$ is a *characteristic mapping* for the class $\mathcal{C}$ and computable function $\mathfrak{L} : \{X|_D \mid X \in \mathcal{C} \text{ and } D \in \mathit{fin}(\mathcal{U})\} \to \mathcal{R}$ if the following condition holds: for every language $X \in \mathcal{C}$ there is a representation $\mathcal{A} \in \mathcal{R}$ of X such that $\mathfrak{L}(X|_D) = \mathcal{A}$ whenever $char_{\mathfrak{L}}(X) \subseteq D$. If $char_{\mathfrak{L}}$ is a characteristic mapping for $\mathcal{C}$, then $char_{\mathfrak{L}}(X)$ is a *characteristic set* for $X \in \mathcal{C}$.

A class $\mathcal{C}$ has a characteristic mapping with respect to a learner $\mathfrak{L}$, if and only if $\mathfrak{L}$ infers $\mathcal{C}$ in the limit from an informant [13]. For every $X \in \mathcal{C}$ and presentation g, there is an index $i \in \mathbb{N}$ such that $char_{\mathfrak{L}}(X) \subseteq X|_{g[i]}$, so from time i onwards, $\mathfrak{L}$ will correctly identify X. Vice versa, if there is a language $X \in \mathcal{C}$ for which no characteristic set exists, then one can contruct a presentation of X on which $\mathfrak{L}$ will never converge.

3.1.4 Polynomial time inference A class $\mathcal{C}$ is *learnable with polynomial time and data* if there is an algorithm $\mathfrak{L}$ that infers $\mathcal{C}$ in the limit, a characteristic mapping $char_{\mathfrak{L}}$, and polynomials p and q such that for every language $X \in \mathcal{C}$ and every presentation g the following conditions hold:

(1) at each time i, the learner $\mathfrak{L}$ uses $p(|g_i|)$ computation steps to output a conjecture consistent with $X|_{g_{[i]}}$, where $|g_i| = \sum_{j\in\{1,\ldots,i\}} |g(j)|$, and
(2) $|char_{\mathfrak{L}}(X)| = q(|\mathcal{A}|)$, where $\mathcal{A}$ is the smallest representation of X in $\mathcal{R}$ [25].

Under this complexity model, identification by enumeration does not yield polynomial-time inference of regular languages when the representation space is deterministic finite automata (if $P \neq NP$). Recall that identification by enumeration takes as conjecture the *smallest* representation that is consistent with the current information. Our observation now follows from the NP-completeness of the *minimal consistent representation* (MCR) problem for DFAs [25], a decision problem which can be stated as follows: *Given $k \in \mathbb{N}$ and a finite sample $X|_D$ of some unknown language X, is k the minimal integer such* *that there is a DFA of size k consistent with $X|_D$?* The problem cannot even be efficiently approximated, since deciding whether a given DFA is only polynomially larger any state-minimal DFA is also NP-hard [37]. Intuitively, the difficulty is that to synthesize $\mathcal{A}$, missing data must be guessed, and the hypothesis space is exponential in the size of D. We note that any polynomial-time procedure for computing the next hypothesis in the identification by enumeration algorithm would also provide an efficient solution to the MCR problem, so no such procedure can exist.

An alternative attempt at in-the-limit learning could be to maintain a prefix tree acceptor that reflects the information in $X|_D$. However, also this apporach misses because although the time needed to update the conjecture is but linear in the size of the input, the conjectured automaton changes with every new positive example recieved, so the algorithm will typically not converge.

Gold [25] was first to present an algorithm that identifies the class of regular languages (over an alphabet A) in the limit with polynomial time and data. To compute a conjecture based on the information contained in $X|_D$, Gold's algorithm searches for a subset S of D from which a DFA $\mathcal{A}$ can be synthesized without having to guess missing data. If $\mathcal{A}$ agrees with X on all of D, then it becomes the learner's next conjecture. If no such $\mathcal{A}$ can be obtained, then the learner synthesizes a consistent PTA instead, which it uses as a dummy conjecture. In essence, the learner waits for a characteristic set for the target language, and disregards data which it cannot use easily.

Gold's algorithm is outlined in Algorithm 1. In the *try-catch* block spanning lines 1 through 5, an attempt is made to construct a DFA from the given data $X|_D$ by invoking the *timid state characterization algorithm*, but the attempt is unsuccessful if $X|_D$ is not compact enough. When this happens, the algorithm resorts to building a prefix tree acceptor that is consistent with $X|_D$, something which is always feasible. On line 6, $\mathcal{A}(w)$ denotes the Boolean that is true if and only if w is accepted by $\mathcal{A}$. The procedure PREFIXTREEACCEPTOR is outlined in Algorithm 5.

The timid state characterization algorithm (TSCA) is the core of Gold's algorithm (see Algorithm 2). If TCSA is informed about X on any superset of a characteristic set for X with respect to Gold's algorithm, then it returns the minimal DFA recognizing X in polynomial time. It does so by constructing an observation table for $X|_D$. However, if TSCA is *not* provided with a superset of a characteristic set, then it often fails to produce a DFA altogether. This is either because the observation table has missing information or because it contains inconsistencies that cannot be resolved without expanding the domain of $X|_D$. The latter problem arises in the subprocedure SYNTHESIZE (not listed explictly) which computes the DFA associated to a closed and consistent observation table (see Section 2.2) or, if the table is not consistent, flags that the information is insufficient and aborts.

Gold's algorithm thus performs rather poorly as long as it does not receive the right data, but since the the data set continues to grow, there will be some time $i \in \mathbb{N}_+$ when the algorithm has received the shortest prefix of g that contains a characteristic set for X with respect to the learner. Although i is guaranteed to be finite, it may be unboundedly large, and the majority of the algorithm's conjectures before time i are prefix tree acceptor.

Algorithm 1 Gold's algorithm to identify a DFA in the limit from an informant [25].

Require: The restriction $X|_D : D \to \mathbb{B}$ of $X : A^* \to \mathbb{B}$ to some finite domain $D \subset A^*$.
Ensure: The DFA $\mathcal{A}$ is consistent with X on D.

```
    try
2:      $\mathcal{A} \leftarrow$ TimidStateCharacterization$(X|_D)$
    catch *Insufficient information in* $X|_D$
4:      return PrefixTreeAcceptor$(X|_D)$
    end try
6:  if $\exists w \in D : \mathcal{A}(w) \neq X|_D(w)$ then
        return PrefixTreeAcceptor$(X|_D)$
8:  return $\mathcal{A}$
```

Algorithm 2 TimidStateCharacterization [25].

Require: The restriction $X|_D : D \to \mathbb{B}$ of $X : A^* \to \mathbb{B}$ to some finite domain $D \subset A^*$ that contains a characteristic set for X.
Ensure: The DFA $\mathcal{A}$ is consistent with X on D and is the minimal DFA for $L(\mathcal{A})$.

```
    $S \leftarrow \{\varepsilon\}$
2:  $T \leftarrow suffixes(D)$
    $Obs \leftarrow$ FillTable$(X|_D, S, T)$
4:  while $\exists w \cdot a \in S \cdot A : row_{wa} \notin \{row_x \mid x \in S\}$ do
        $S \leftarrow S \cup \{w \cdot a\}$
6:      $Obs \leftarrow$ FillTable$(X|_D, S, T)$
    return $\mathcal{A} \leftarrow$ Synthesize$(S, T, Obs)$
```

Algorithm 3 FillTable

Require: The restriction $X|_D : A^* \to \mathbb{B}$ of X to some finite domain D, and a pair of sets $S, T \subseteq A^*$.
Ensure: *Obs* is a (possibly inconsistent and incomplete) observation table.

```
    for each $w \in S$ do
2:      for each $x \in T$ do
            if $w \cdot x \in D$ then
4:              $Obs(w, x) \leftarrow X|_D(w \cdot x)$
            else
6:              throw *Insufficient information*
    return $Obs$
```

3.2 Learning from a Minimally Adequate Teacher

Learning from a *minimally adequate teacher (MAT-learning)* was introduced in a seminal paper by Angluin [4]. The MAT-learning scenario assumes a *learner* and a *teacher*. The learner wants to learn an unknown regular language X over alphabet A known by the teacher. In order to learn X, the learner asks questions, which the teacher must answer. In particular, the learner can ask two types of questions, namely

(1) *membership queries*, each consisting of a word w, and
(2) *conjectures*, each consisting of a description of a language Y.

The teacher will respond to membership queries by answering either *yes* (if $w \in X$) or *no* (otherwise) and to conjectures by either *yes* (if $X = Y$) or by a word x in the symmetric difference of X and Y (otherwise). The word x may be chosen arbitrarily and we refer to it as a *counterexample*. We say that a teacher that behaves as described is a *minimally adequate teacher (MAT)*.

Angluin's main result is that the learner can learn any regular language from a teacher in time polynomial in $|A|$, $|\mathcal{A}|$, and $|x|$, where $|A|$ is the number of symbols in A, $|\mathcal{A}|$ denotes the number of states of the minimal DFA for X, and $|x|$ denotes the maximum length of any counterexample word presented by the teacher.

If the teacher is able to answer both membership queries and conjectures correctly, it may seem that she should actually already have a representation of the language to be learned. If this is the case, we may ask ourselves why she doesn't simply give the learner this representation, rather than playing the cat-and-mouse game of queries and answers. One could therefore think that the MAT learning paradigm is certainly more of a mathematical abstraction than a realistic setting. Still, it has played a very important role in the field of grammatical inference. The explanation for this has two parts. The first is that grammatical inference, as we have seen in Section 3.1, is inherently hard. To be able to achieve positive theoretical results, we must either restrict the class of languages to be learned severely, or give the learner access to a powerful information source, such as a MAT. The second is that there are practical settings where we can assume something very close to a MAT. One such example comes from formal verification. One approach to so-called *black box checking* [35, 26], uses a MAT algorithm where the equivalence queries to the teacher are replaced by *conformance testing* algorithms; see Section 8.

3.2.1 The MAT learning algorithm The crux of Angluin's learning algorithm is that the learner maintains an observation table and iteratively makes this table closed and consistent by querying the teacher. The MAT learning algorithm is presented as Algorithm 4. Initially, the learner sets $S_{\text{pre}} = S_{\text{suff}} = \{\varepsilon\}$ and asks the teacher membership queries for ε and for each $a \in A$. The initial observation table is then constructed to reflect the learned information.

The main loop of the algorithm is concerned with repeatedly making the observation table closed and consistent, and presenting the teacher with a conjecture. If the observation table is not consistent, then the learner finds words w_1, w_2 in S_{pre}, an a in A, and a w in S_{suff} that witness this fact. The witnessing suffix aw is then added to S_{suff} and the table is completed to incorporate this new entry. Similarly, if the observation table is not closed, the learner finds a word w_1 in S_{pre} and an a in A witnessing this fact. The witnessing word w_1a is then added to S_{pre} and the table is completed to incorporate this

Algorithm 4 The MAT learning algorithm by Angluin [4].

```
    Initialize $S_{\text{pre}}$ and $S_{\text{suff}}$ to $\{\varepsilon\}$
 2: Ask membership queries for $\varepsilon$ and each $a \in A$
    Construct the initial observation table $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$
 4: repeat
        while $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not closed or not consistent do
 6:         if $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not closed then
                find $w_1 \in S_{\text{pre}}$ and $a \in A$ such that
 8:                 $\mathit{row}_{w_1 a}$ is different from $\mathit{row}_w$ for all $w \in S_{\text{pre}}$
                add $w_1 \cdot a$ to $S_{\text{pre}}$
10:             extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
            if $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not consistent then
12:             find $w_1$ and $w_2$ in $S_{\text{pre}}$, $a \in A$, and $w \in S_{\text{suff}}$ such that
                    $\mathit{row}_{w_1} = \mathit{row}_{w_2}$ and $\mathit{Obs}(w_1 \cdot a \cdot w) \neq \mathit{Obs}(w_2 \cdot a \cdot w)$
14:             add $a \cdot w$ to $S_{\text{suff}}$
                extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
16:     Once $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is closed and consistent,
                    let $\mathcal{A}_{\mathit{Obs}}$ be its associated automaton and make the conjecture $\mathcal{A}_{\mathit{Obs}}$
18:     if the teacher replies with a counterexample $w$ then
            add $w$ and all its prefixes to $S_{\text{pre}}$
20:         extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
    until the teacher replies yes to the conjecture $\mathcal{A}_{\mathit{Obs}}$
22: return $\mathcal{A}_{\mathit{Obs}}$
```

new entry.

If the observation table is closed and consistent, a conjecture $\mathcal{A}_{\mathit{Obs}}$ is generated. If the conjecture is correct, the learner is done and can output $\mathcal{A}_{\mathit{Obs}}$. Otherwise, the counterexample w from the teacher is added, along with its prefixes, to S_{pre}. The table is then completed to incorporate this new information.

Angluin proves that the total running time of Algorithm 4 is $O(km^2n^2 + kmn^3)$, where $k = |A|$, n is the number of states of the minimal DFA for X, and m is the maximum length of any counterexample presented by the teacher [4]. It follows that, if the teacher always presents counterexamples of minimal length, then they will be at most $O(n)$ in length and therefore the learning algorithm will run in time $O(kn^4)$.

Example 3.1. We consider a run of Algorithm 4 with the language Z of the regular expression

$$r = (a + b)^* b(a + b)$$

as the target. This language consists of all words over $A = \{a, b\}$ that have length at least two and such that the second to last letter is b.

The first thing the algorithm does is to set $S_{\text{pre}} = S_{\text{suff}} = \{\varepsilon\}$ and to use membership queries to find the *Obs*-values for ε, a, and b. This results in the following table.

	ε
ε	0
a	0
b	0

In the table, the columns represent elements of S_{suff} and the rows elements in $S_{\text{pre}} \cup (S_{\text{pre}} \cdot A)$ with the elements of S_{pre} above the double horizontal line and the elements of $(S_{\text{pre}} \cdot A) \setminus S_{\text{pre}}$ below it. The value of a cell in row w and column x is $\mathit{Obs}(w \cdot x)$.

The table is clearly consistent, since S_{suff} is a singleton. It is also closed, since every row that appears below the double line also appears above it. This means that the algorithm will reach line 16 and construct an automaton from the table. Since S_{pre} is a singleton and the table contains only zeroes, the automaton will only have one state and this state will not be accepting. Thus it will accept the empty language. When the algorithm makes this conjecture, the teacher is obliged to provide a counterexample. Since the conjecture was an automaton that accepts nothing, the counterexample will have to be a word in Z. We will assume that the teacher responds with $ba \in Z$. The algorithm now adds ba, b, and ε to S_{pre} (the latter was of course already included) and uses membership queries to compute a new table:

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

The new table is obviously closed, since every row below the double line is also present above the line. However, it is not consistent. For example, $\mathit{row}_\varepsilon = \mathit{row}_b$, but $\mathit{Obs}(\varepsilon \cdot a \cdot \varepsilon) = \mathit{Obs}(a) = 0$ while $\mathit{Obs}(b \cdot a \cdot \varepsilon) = \mathit{Obs}(ba) = 1$. This means that the algorithm will execute the code on lines 11 to 15, for example with the instantiations $w_1 = \varepsilon$, $w_2 = b$, $a = a$, and $w = \varepsilon$. In this case, $a \cdot \varepsilon = a$ will be added to S_{suff} and the following table will be computed:

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

This third table is consistent, since no two rows above the double line are the same. But, this time, it is not closed since row_{bb} does not appear above the double line. This means that the code on lines 6 to 10 will be executed with $w_1 = b$ and $a = b$. Finally, bb will be added to S_{pre} with the following table as a result:

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

This table is consistent, since no two rows above the double line have the same values, and closed, since all possible rows are represented above the double line. Thus the algorithm will reach line 16 again and construct an automaton $\mathcal{A} = (Q, I, E, T)$ with

- $Q = \{\varepsilon, b, ba, bb\}$,
- $I = \{\varepsilon\}$,
- $T = \{ba, bb\}$, and
- $E = \{(\varepsilon, a, \varepsilon), (\varepsilon, b, b), (b, a, ba), (b, b, bb), (ba, a, \varepsilon), (ba, b, a), (bb, a, ba), (bb, b, bb)\}$.

This automaton is shown in Figure 1 and is the minimal DFA for Z. Thus the teacher will approve the conjecture and the algorithm will terminate.

4 Learning from given data

Recall that Gold's algorithm (which was covered in Section 3.1) is informed about a regular language X through an infinite sequence g of positive and negative examples and converges in the limit to the minimal DFA for X. The convergence, however, may take an unbounded amount of time, and the majority of the algorithm's conjectures before that will be prefix tree acceptors (PTAs) which, in a practical setting, isn't very convenient. PTAs do not provide any *compression* nor *generalization* of the data. This means that they do not represent the data in a significantly more succinct manner than the data itself and that they always exactly describe the positive examples seen so far; and nothing more. In brief, PTAs are therefore not much more useful than the positive data itself.

Since Gold's first paper on learning in the limit, much effort has been invested in the search for polynomial time learning algorithms that produce small automata and which generalize the given examples even when conditions are less than perfect, i.e., when the examples do not contain a characteristic set. In this work, the emphasis has shifted from in-the-limit behavior to the problem of *learning from given data*. This problem, which is also known as *the consistency problem*, reads as follows. *Given a restriction X_D of a target language X to a finite domain D, find a representation $\mathcal{A}$ in the hypothesis space $\mathcal{R}$ that agrees with X on all of D, i.e., $\mathcal{A}(w) = X(w)$, for every $w \in D$.* Ideally, we want such representations to be small, and we want to find them efficiently.

An algorithm that learns from given data may also identify X in the limit, but this is not always true. For example, the naive algorithm that outputs a PTA for the input sample

Algorithm 5 PREFIXTREEACCEPTOR

Require: A restriction $X|_D$ of X to the finite domain $D \subseteq A^*$.
Ensure: $\mathcal{A}$ is a DFA consistent with X on D.

```
   $Q \leftarrow \mathit{prefixes}(X|_D^{-1}(1))$
2: $I \leftarrow \varepsilon$
   $E \leftarrow \{(x, a, x \cdot a) \mid x \cdot a \in \mathit{prefixes}(X|_D^{-1}(1))\}$
4: $T \leftarrow X|_D^{-1}(1)$
   return $\mathcal{A} = (Q, I, E, T)$
```

solves the consistency problem, but it does not have the in-the-limit behavior. However, if $\mathfrak{L}$ is a learning algorithm that solves the consistency problem and every language $X \in \mathcal{C}$ has a characteristic set with respect to $\mathfrak{L}$, then $\mathfrak{L}$ identifies $\mathcal{C}$ in the limit.

In our presentation of algorithms that learn from given data, we assume that the input is a restriction of the target language X to a finite domain D. As in Section 3.1, we say that the function $X|_D$ is a *sample* of X. The algorithm's objective is to produce an automaton that is consistent with $X|_D$, that is, a representation $\mathcal{A}$ such that $X|_D^{-1}(1) \subseteq L(\mathcal{A})$ and $X|_D^{-1}(0) \cap L(\mathcal{A}) = \emptyset$.

State merging algorithms. A common approach to learning from given data is to start by constructing a PTA for the positive sample $X|_D^{-1}(1)$ (see Algorithm 5) and then merge as many states as possible in this PTA. Since merging states may result in an automaton that recognises a larger language, each merge has to be preceded by a consistency check, to make sure that none of the negative examples from $X|_D^{-1}(0)$ are included in the language of the automaton. The method, which is due to Trakhtenbrot and Barzdin [41], has given rise to a family of state merging algorithms that mainly differ in the strategies they use to merge states and the order in which the merges are done. This order may influence the eventual outcome, because performing one state merge may preclude another one.

When the hypothesis space is deterministic finite automata, merges are typically done as in Algorithm 6. The algorithm takes as input a DFA $\mathcal{A} = (Q, I, E, T)$ and a pair of states q_1, q_2 in Q, and returns a DFA $\mathcal{A}'$ in which q_1 and q_2 have been merged into the new state q. Every edge that once led to q_1 or q_2 now leads to q, and every edge that left q_1 or q_2 is now leaving q. If at least one of q_1 or q_2 is an initial state, then so is q, and the same holds for the property of being an accepting state. It is easy to see that the resulting automaton recognizes a superset of $L(\mathcal{A})$, but is not necessarily deterministic. This is the case when both q_1 and q_2 have outgoing edges that are labeled by the same symbol, but which lead to different states p_1 and p_2. To remove the nondeterminism, the merge algorithm calls itself recursively to merge p_1 and p_2. Since every application of the algorithm produces a smaller automaton, the algorithm will eventually terminate and return a DFA.

The RPNI algorithm. Perhaps the most well known state merging algorithm is the regular positve-negative inference (RPNI) algorithm [33]. The basic idea of the algorithm is to control the order in which merges of states are performed. In particular, the algorithm avoids merging two states that are both involved in loops. This is done as follows. First,

Algorithm 6 MERGE

```
Require: A DFA or NFA $\mathcal{A} = (Q, I, E, T)$ and a pair of states $q_1, q_2 \in Q$
Ensure: In the DFA $\mathcal{A}'$, the states $q$ and $q'$ have been merged into one.
    $Q' \leftarrow \{\{q\} \mid q \in Q \setminus \{q_1, q_2\}\} \cup \{\{q_1, q_2\}\}$
 2: $I' \leftarrow \{q' \in Q' \mid q' \cap I \neq \emptyset\}$
    $E' \leftarrow \{(q', a, p') \mid \exists q \in q', p \in p' : (p, a, q) \in E\}$
 4: $T' \leftarrow \{q' \in Q' \mid q' \cap T \neq \emptyset\}$
    $\mathcal{A}' = (Q', I', E', T')$
 6: while $\exists q', p'_1, p'_2 \in Q', a \in A : (q', a, p'_1) \in E' \wedge (q', a, p'_2) \in E'$ do
        $\mathcal{A}' \leftarrow$ MERGE$(\mathcal{A}', p'_1, p'_2)$
 8: return $\mathcal{A}'$
```

Algorithm 7 RPNI

```
Require: A restriction $X|_D$ of $X$ to the finite domain $D \subseteq A^*$.
Ensure: $\mathcal{A}$ is a DFA consistent with $X$ on $D$.
    $\mathcal{A} = (Q, I, E, T) \leftarrow$ PREFIXTREEACCEPTOR$(X|_D^{-1}(1))$
 2: RED $\leftarrow \varepsilon$
    BLUE $\leftarrow E($RED$)$
 4: while BLUE $\neq \emptyset$ do
        choose $q_b \in$ BLUE
 6:     BLUE $\leftarrow$ BLUE $\setminus \{q_b\}$
        if $\exists q_r \in$ RED : MERGE$(\mathcal{A}, q_r, q_b)$ is consistent with $X$ on $D$ then
 8:         $\mathcal{A} \leftarrow$ MERGE$(\mathcal{A}, q_r, q_b)$
        else
10:         RED $\leftarrow$ RED $\cup \{q_b\}$
        BLUE $\leftarrow E($RED$) \setminus$ RED
12: return $\mathcal{A} = (Q, I, E, T)$
```

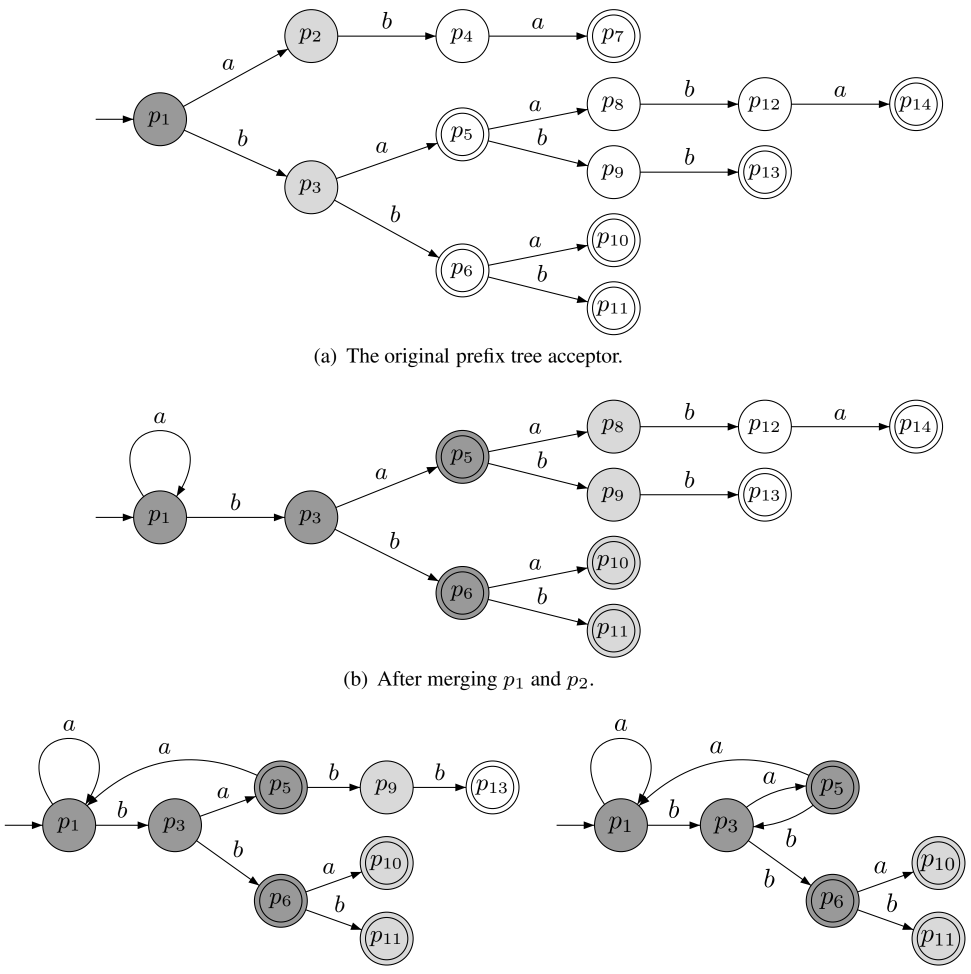
the root of the PTA is colored red and all successors of the root are colored blue (all other states are white). The algorithm then always tries to merge a blue state with a red state. When a merge is performed, the new combined state is red and all its white successors are colored blue. If a blue state cannot be merged with any red state, it is colored red and its white successors become blue. The basic RPNI scheme is shown in Algorithm 7.

Example 4.1. Consider running the RPNI algorithm with input $Z|_D$, where

- $Z|_D^{-1}(1) = \{ba, bb, aba, bba, bbb, babb, baaba\}$ and
- $Z|_D^{-1}(0) = \{\varepsilon, a, b, aa, bab\}$.

Constructing a prefix tree acceptor from $Z|_D^{-1}(1)$, we get the automaton shown in Figure 2(a). State p_1 is the only red state and its successors p_2 and p_3 are blue.

States p_1 and p_2 can be merged without violating $Z|_D$. To avoid nondeterminism, the successors of p_2 are "folded" into the successors of p_1. This means that p_4 is merged with p_3 and p_7 is merged with p_5. The resulting automaton is depicted in Figure 2(b), where we have also introduced a number of new red states, which is justified as follows. If we were to merge p_3 with p_1, the resulting automaton would accept a and b, both of which



(a) The original prefix tree acceptor.

(b) After merging p_1 and p_2.

(c) After merging p_8 with p_1.

(d) After merging p_9 with p_3.

Figure 2. The automata from Example 4.1. The accepting state are shown as double circles. Red states are colored dark grey while blue states are light gray.

belong to $Z|_D^{-1}(0)$. Once p_3 becomes red, p_5 and p_6 become blue. None of them can be merged with p_1, since they are accepting, while p_1 is initial, and ε belongs to $Z|_D^{-1}(0)$. Also, none of them can be merged with p_3, since the resulting automaton would accept $b \in Z|_D^{-1}(0)$. Thus we can color both of them red and their successors blue. We can now merge state p_8 with p_1. The resulting automaton is shown in Figure 2(c). Next, we merge p_9 with p_3, resulting in the automaton in Figure 2(d).

Finally, state p_{10} can be merged with p_5 and state p_{11} can be merged with p_6. This actually gives us the minimal DFA $\mathcal{A}_Z$ for the language Z from Example 2.1, depicted in Figure 1.

The running time of the RPNI algorithm is polynomial in the size of D, that is, in the sum of lengths of the words for which $X|_D$ is defined. This can be seen from the fact that every iteration of the main while loop colors at least one white state blue and no state that has once become blue can ever become white.

The algorithm also identifies the regular languages in the limit. In fact, for every implementation of RPNI, there is a mapping $\mathit{char}_{RPNI} : \mathit{Reg} \to \mathit{pow}(\mathcal{U})$ from the regular languages to the powerset of the universe such that, for every language $X \in \mathit{Reg}$,

- the size of $\mathit{char}_{RPNI}(X)$ is polynomial in the size of the minimal DFA for X and
- if $\mathit{char}_{RPNI}(X) \subseteq D$, then the algorithm, when run on input $X|_D$, returns the minimal DFA for X.

For a complete proof of this statement, we refer to de la Higuera [14].

Notice that above, we talk about a characteristic sample *for every implementation* of the RPNI algorithm. This is because Algorithm 7 does not completely specify in which order merges should be performed. To specify this completely, we must define how the choice of blue state on line (5) is made and in which order red states are considered on line (7). These choices lead to different functions for the characteristic samples and may also significantly influence the performance of the algorithm. It is also in these choices that many variants of the RPNI scheme differ. We discuss one such variant next.

Evidence driven state merge. The evidence driven state merge (EDSM) algorithm was entered by Price into the Abbadingo One DFA Learning Competition [28] and was one of two co-winners. It is a good example of an RPNI-variant that uses a heuristic to find the merges that are in some sense most promising. These merges are then performed first.

In each iteration the EDSM algorithm first checks whether there is some blue state that can be promoted. Only when this is not the case it tries to find a merge. At this stage, the algorithm evaluates *every* possible merge, that is, every pair of one red state and one blue state. Each possible merge is given a score and the merge with the highest score is performed before the algorithm continues with the next iteration. The score for a merge of a red state p_r and a blue state p_b is computed as follows. The automaton $\mathcal{A}' = \textsc{Merge}(\mathcal{A}, p_r, p_b)$ is constructed. If it violates $X|_D$, the score is set to $-\infty$. Otherwise, let S be the subset of the states of $\mathcal{A}'$ such that, for every $p \in S$, there is some word in D that takes $\mathcal{A}'$ from the initial state to p. Set the score for the merge to $|D| - |S|$. Intuitively, the score gets better the more words from D lead to the same state.

5 Learning non-deterministic finite automata

While DFAs and NFAs both define the class of regular languages, there are many situations where one is preferable above the other. The vast majority of research into learning regular languages has focused on DFAs, to a large part because of the direct correspondence between the Myhill-Nerode equivalence classes and the states of a minimal DFA. In many applications, however, the use of NFAs as language representations is more desirable, primarily due to the compactness of representation they offer. There are also language classes for which the DFA-based methods perform poorly, and NFA-based methods can be expected to do better [22].

Since there is not always a unique minimal NFA for a language, most research into NFA learning deals with subclasses, within which languages have representations that are in some sense unique. We start by introducing an important such subclass.

Residual languages. Let $X \subseteq A^*$ be a language and let $w \in A^*$ be a word. Then the set of *all suffixes* that can be added to w to form a word in X is a *residual language* of X. In other words, Y is a residual language of X if there is a word $w \in A^*$ such that $Y = \{x \mid w \cdot x \in X\}$. We call Y the residual language of X with respect to w (also known as the Brzozowski derivative [8] or left quotient [45]). If w is not a prefix of any word in X, then the residual language of X with respect to w is $\emptyset$. For the set of all residual languages of X, we write $\mathit{Res}(X)$.

Recall that two words w_1 and w_2 are equivalent with respect to X in the sense of Myhill and Nerode, written $w_1 \equiv_X w_2$, if and only if for every word x, we have $w_1 \cdot x \in X$ if and only if $w_2 \cdot x \in X$ (Definition 2.1). This means that $w_1 \equiv_X w_2$ if and only if the residual languages of X with respect to w_1 and w_2 are the same. We can thus conclude that every language X has exactly as many residual languages as the equivalence $\equiv_X$ has classes. In particular, a regular language has finitely many residual languages, one corresponding to each state of the minimal DFA for the language. We say that a residual language Y of X is *prime* if it is not the union of some members of $\mathit{Res}(X) \setminus \{Y\}$. Equivalently, the residual language Y of X is *composite*, or *non-prime*, if and only if it is the union of all members of $\mathit{Res}(X)$ that it properly contains.

Example 5.1. Consider the language Z represented by the regular expression

$$(a + b)^* b(a + b).$$

This language has the four equivalence classes $[\varepsilon]_{\equiv_Z}$, $[b]_{\equiv_Z}$, $[ba]_{\equiv_Z}$, and $[bb]_{\equiv_Z}$. In the following, we denote by Z_w the residual language of Z with respect to w. In this notation, these classes correspond to residual languages as follows.

- $[\varepsilon]_{\equiv_Z}$ corresponds to $Z_\varepsilon = Z$.
- $[b]_{\equiv_Z}$ corresponds to $Z_b = Z \cup \{a, b\}$.
- $[ba]_{\equiv_Z}$ corresponds to $Z_{ba} = Z \cup \{\varepsilon\}$.
- $[bb]_{\equiv_Z}$ corresponds to $Z_{bb} = Z \cup \{\varepsilon, a, b\}$.

Notice that we have $Z_{bb} = Z_b \cup Z_{ba}$, while none of the other residual languages can be formed as a non-trivial union of some of the others. Therefore, Z_ε, Z_b, and Z_{ba} are *prime* residual languages, while Z_{bb} is a *non-prime* residual language.

Residual finite state automata. The above terminology was introduced by Denis et al., who also used it to define an important subclass of the NFAs, the *Residual finite state automata* (RFSA) [16, 17]. As noted above, each state in the minimal DFA for a language corresponds to a unique residual language. This is also true for RFSA, but we do not require *every* residual language to be represented by a state.

Definition 5.1. An NFA $\mathcal{A}$ is a residual finite state automaton if, for every state p of $\mathcal{A}$, the language accepted by $\mathcal{A}$, when started from p, is a residual language of the language accepted by $\mathcal{A}$.

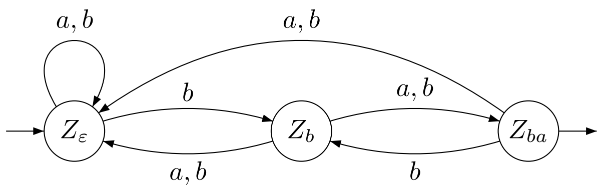


Figure 3. The canonical RFSA for Z from Example 5.2.

In particular, every DFA without unreachable states is an RFSA. Indeed, if p is a state of $\mathcal{A}$ and w is a word that takes $\mathcal{A}$ from the initial state to p, then the language accepted by $\mathcal{A}$ when it is started from p is the residual language of $L(\mathcal{A})$ with respect to w. Denis et al. prove the following theorem [16].

Theorem 5.1. *Every regular language X is accepted by an RFSA with a number of states equal to the number of prime residual languages of X.*

The automaton refered to in Theorem 5.1 can be constructed as follows. Let $\mathit{PrimeRes}(X)$ be the set of prime residual languages of the regular language X over alphabet A. We can construct an RFSA $\mathcal{A}_X$ for X from $\mathit{PrimeRes}(X)$ as follows.

- The set of states of $\mathcal{A}_X$ is $\mathit{PrimeRes}(X)$;
- $p \in \mathit{PrimeRes}(X)$ is an initial state if $p \subseteq X$;
- $p \in \mathit{PrimeRes}(X)$ is an accepting state if $\varepsilon \in p$; and
- (p_1, a, p_2) is an edge of $\mathcal{A}_X$ if p_2 is a subset of the residual language of p_1 with respect to a.

The automaton $\mathcal{A}_X$ is *saturated* in the sense that no additional state can be made initial and no additional edge can be added without changing the language of the automaton. It is also *reduced* in the sense that no state represents a language that is the union of the languages of some other states. In fact, $\mathcal{A}_X$ is the *unique* reduced and saturated RFSA for X and is called the *canonical* RFSA for X [16, 17].

Example 5.2. Consider our example language Z again, defined by the regular expression $(a + b)^*b(a + b)$. Since the language has four equivalence classes, we know that the minimal DFA that recognizes it has four states. Theorem 5.1 tells us that there is an RFSA for Z that has only three states, one each for the prime residual languages Z_ε, Z_b, and Z_{ba}. Using the above construction, we obtain the canonical RFSA for Z, shown in Figure 3. The state for Z_ε is an initial state, since $Z_\varepsilon = Z$ is a subset of Z. The state Z_{ba} is accepting, since $\varepsilon \in Z_{ba}$. For the transitions, consider, for example, the b-labeled edge from Z_{ba} to Z_b. This edge is included because $Z_b = Z \cup \{a, b\}$ is a subset of the residual language of $Z_{ba} = Z \cup \{\varepsilon\}$ with respect to b, which is in fact Z_b.

The uniquness of canonical RFSA for regular languages opens up the possibility of reusing many of the learning techniques used for DFAs.

A MAT learning algorithm for RFSA. The DFA-like properties of RFSA come to the fore in the context of MAT learning. Recently, Bollig et al. [7] developed such an algorithm. With the right definitions, the changes needed to the original algorithm of Angluin (Algorithm 4) are rather minor.

Given an observation table $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$, we need a notion of inclusion between rows. Given $w, x \in S_{\text{pre}} \cup S_{\text{pre}} \cdot A$, we say that the row of w is *covered* by the row of x, written $\mathit{row}_w \sqsubseteq \mathit{row}_x$, if $\mathit{Obs}(w \cdot y) \leqslant \mathit{Obs}(x \cdot y)$ for every $y \in S_{\text{suff}}$.

Let $\mathcal{R}$ be the set of rows of the table, i.e., $\mathcal{R} = \{\mathit{row}_w \mid w \in S_{\text{pre}} \cup S_{\text{pre}} \cdot A\}$. For a subset R of $\mathcal{R}$, let Join_R be the function from S_{suff} to $\{0,1\}$ defined by $\mathit{Join}_R(y) = \max_{r \in R}(r(y))$.

We can now define prime rows as follows. The row row_w of $w \in S_{\text{pre}} \cup S_{\text{pre}} \cdot A$ is *composite* if there is an $R \subseteq \mathcal{R} \setminus \{\mathit{row}_w\}$ such that $\mathit{row}_w = \mathit{Join}_R$. In fact, row_w is composite if and only if it is the join of all rows that it strictly covers. Otherwise, row_w is *prime*. We write Primes for the set of all prime rows and $\mathit{Primes}_{\text{pre}}$ for $\mathit{Primes} \cap \{\mathit{row}_w \mid w \in S_{\text{pre}}\}$.

The notion of closedness of an observation table now translates into a notion of *RFSA-closedness* that requires that every row corresponding to a word in $S_{\text{pre}} \cdot A \setminus S_{\text{pre}}$ must be the join of the rows in $\mathit{Primes}_{\text{pre}}$ that it covers. Formally, $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is RFSA-closed if, for every word $w \in S_{\text{pre}} \cdot A \setminus S_{\text{pre}}$, there is $R \subseteq \mathit{Primes}_{\text{pre}}$ such that $\mathit{row}_w = \mathit{Join}_R$.

Similarly, $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is *RFSA-consistent* if, for all $w_1, w_2 \in S_{\text{pre}}$ and every $a \in A$, if $\mathit{row}_{w_1} \sqsubseteq \mathit{row}_{w_2}$, then $\mathit{row}_{w_1 \cdot a} \sqsubseteq \mathit{row}_{w_2 \cdot a}$.

Finally, we need to modify the way an automaton is constructed from an observation table. Given $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$, we define the corresponding automaton $\mathcal{A}_{\mathit{Obs}} = (Q, I, E, T)$ with

- $Q = \mathit{Primes}_{\text{pre}}$,
- $I = \{p \in Q \mid p \sqsubseteq \mathit{row}_\varepsilon\}$,
- $T = \{\mathit{row}_w \in Q \mid \mathit{Obs}(w) = 1\}$, and
- $E = \{(\mathit{row}_w, a, \mathit{row}_x)) \mid \mathit{row}_x \sqsubseteq \mathit{row}_{w \cdot a}\}$.

The well-definedness of $\mathcal{A}_{\mathit{Obs}}$ is proved similarly as in Section 2.2. The MAT learner for RFSA is presented in Algorithm 8. It requires $O(n^2)$ equivalence queries and $O(mn^3)$ membership queries in the worst case, where n is the size of the minimal *DFA* for the language and m is the length of the longest counterexample received from the teacher [7].

Other learning algorithms for NFAs. During the first decade of the 21st century, a number of other algorithms for learning NFAs were presented. Here, we only briefly mention some of them.

Denis at al. present a learning algorithm for RFSA, called *DeLeTe2*, which is based on prefix tree acceptors and state merging [17]. The algorithm does not necessarily learn the canonical RFSA for the target language, but rather an RFSA whose size lies between that of the canonical RFSA and the minimal DFA.

The *unambigous finite automata* (UFA) form another class of restricted nondeterministic finite automata, with the central property that no word in the language of a UFA has more than one accepting run [40]. Coste and Fredouille give an algorithm for learning a UFA from given data, again by merging states from a prefix tree acceptor [12]. It has the property that, given a specific sample, the same automaton will be constructed, irrespective of the order in which merges are performed.

Algorithm 8 The MAT-Learning algorithm for RFSA by Bollig et al. [7].

```
    Initialize $S_{\text{pre}}$ and $S_{\text{suff}}$ to $\{\varepsilon\}$
 2: Ask membership queries for $\varepsilon$ and each $a \in A$
    Construct the initial observation table $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$
 4: repeat
      while $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not RFSA-closed or not RFSA-consistent do
 6:     if $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not RFSA-closed then
          find $w_1 \in S_{\text{pre}}$ and $a \in A$ such that
 8:         $\mathit{row}_{w_1 \cdot a} \in \mathit{Primes} \setminus \mathit{Primes}_{\text{pre}}$
          add $w_1 \cdot a$ to $S_{\text{pre}}$
10:       extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
        if $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is not RFSA-consistent then
12:       find $w_1$ and $w_2$ in $S_{\text{pre}}$, $a \in A$, and $w \in S_{\text{suff}}$ such that
            $\mathit{Obs}(w_1 \cdot a \cdot w) = 1$ and $\mathit{Obs}(w_2 \cdot a \cdot w) = 0$ and $\mathit{row}_{w_1} \sqsubseteq \mathit{row}_{w_2}$
14:       add $a \cdot w$ to $S_{\text{suff}}$
          extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
16:   Once $(S_{\text{pre}}, S_{\text{suff}}, \mathit{Obs})$ is RFSA-closed and RFSA-consistent,
            let $\mathcal{A}_{\mathit{Obs}}$ be its associated automaton and make the conjecture $\mathcal{A}_{\mathit{Obs}}$
18:   if the teacher replies with a counterexample $w$ then
        add $w$ and all its suffixes to $S_{\text{suff}}$
20:     extend $\mathit{Obs}$ to $(S_{\text{pre}} \cup S_{\text{pre}} \cdot A) \cdot S_{\text{suff}}$ using membership queries
    until the teacher replies yes to the conjecture $\mathcal{A}_{\mathit{Obs}}$
22: return $\mathcal{A}_{\mathit{Obs}}$
```

The consistency problem for unrestricted (i.e., fully nondeterministic) finite automata is treated by Vazquez et al. [44]. They present a family of in-the-limit algorithms for nondeterministic finite automata that use maximal automata and state merging to infer a regular language. This work was later continued with the definition of the OIL algorithm, based on so-called universal automata [22, 21].

6 Learning regular tree languages

In this section we discuss the extension of MAT-learning (Section 3.2) to (ranked) regular tree languages. This topic was pioneered by Sakakibara, although the main focus of his work was on derivation trees of context-free grammars [38]. It is, however, well-known that regular tree languages and derivation trees of context-free grammars are extremely closely connected. This can be seen as follows. The *yield* of a tree is the sequence of labels of the leafs, from left to right. The relation between context-free grammars and regular tree languages is that the set of derivation trees of a context free grammar is always a regular tree language and each language of yields of a regular tree language is a context-free word language. Sakakibara generalized MAT-learning to *skeletal* tree languages, which are tree languages in which all internal nodes are unlabeled [30]. We note that MAT-learning can also be extended to weighted tree series — for a survey on this topic, we suggest [18].

The extension to full regular tree languages, which we present here, is by Drewes and Högberg [19]. We assume that the alphabet A is partitioned into $A^{(0)}, \dots, A^{(R)}$ for some constant natural number R. Here $A^{(i)}$ contains the symbols of *rank* i. We denote trees by the letters t and s. We write (ranked) trees as $a(t_1, \dots, t_n)$, where the root is labeled by $a \in A^{(n)}$ under which the (ranked) trees $t_1, \dots, t_n$ are attached. A *context* c is a tree in which exactly one leaf is labeled with a *hole marker* $\bullet$. The *depth* of a context c is the length of the path from the root of c to the hole marker, not including the hole marker. We denote the depth of context c by $\text{depth}(c)$. For example, a context where the hole marker is a child of the root has depth one. Given a context c and a tree t, we denote by $c[t]$ the tree obtained by replacing the unique $\bullet$-labeled node in c by the tree t. If S is a set of trees, we denote by $A(S)$ the set $\{a(t_1, \dots, t_n) \mid a \in A^{(n)} \text{ and } t_1, \dots, t_n \in S\}$. As before, we denote the unknown regular (tree) language by X.

6.1 Observation tables for trees

An *observation table for trees* $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ is defined analogously as in Section 3.2 with the differences that

- T_{pre} is a *subtree-closed* set of trees, that is, for every tree $a(t_1, \dots, t_n) \in T_{\text{pre}}$ we also have hat $t_1, \dots, t_n$ are in T_{pre};
- C_{suff} is a *generalization-closed* set of contexts, that is, for every context of the form $c[a(t_1, \dots, t_{i-1}, \bullet, t_{i+1}, \dots, t_n)] \in C_{\text{suff}}$ we have that the context c is in C_{suff} as well and $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$ are in T_{pre}.
- *Obs* is a function $\mathit{Obs} : (T_{\text{pre}} \cup A(T_{\text{pre}})) \times C_{\text{suff}} \to \{0,1\}$ such that, for a context c and a tree t, $\mathit{Obs}(t,c) = 1$ if and only if $c[t] \in X$.

By row_t we denote the function $\mathit{row}_t : C_{\text{suff}} \to \{0,1\}$ such that, for each context c, $\mathit{row}_t(c) = 1$ if and only if $c[t] \in X$. In this section, we are interested in learning bottom-up deterministic tree automata. For more details about tree automata, we refer to Chapter **??**. With bottom-up automata in mind, the intuition behind observation tables for trees is similar to observation tables for words, i.e., T_{pre} can be seen as a set of prefixes and C_{suff} as a set of suffixes.

An observation table for trees is *closed* if, for each $a \in A^{(n)}$ and trees $t_1, \dots, t_n \in T_{\text{pre}}$, there is a $t \in T_{\text{pre}}$ such that $\mathit{row}_{a(t_1,\dots,t_n)} = \mathit{row}_t$. It is *consistent* if the following condition is satisfied: for all $a \in A^{(n)}$, all $t_1, \dots, t_n, t'_1, \dots, t'_n \in T_{\text{pre}}$, if $\mathit{row}_{t_i} = \mathit{row}_{t'_i}$ for all $i = [n]$, then $\mathit{row}_{a(t_1,\dots,t_n)} = \mathit{row}_{a(t'_1,\dots,t'_n)}$.

With a closed and consistent observation table for trees $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$, we can associate a finite bottom-up tree automaton $\mathcal{A}_{\mathit{Obs}} = (Q, A, \Delta, T)$ that represents the information we currently have about the unknown language X. Here, the transition relation Δ is a subset of $\bigcup_n (Q^n \times A^{(n)} \times Q)$. This tree automaton is defined as follows:

- $Q = \{\mathit{row}_t \mid t \in T_{\text{pre}}\}$;
- for every $n \in \mathbb{N}$, $a \in A^{(n)}$, we have $(\mathit{row}_{t_1}, \dots, \mathit{row}_{t_n}, a, \mathit{row}_{a(t_1,\dots,t_n)}) \in \Delta$;
- $I = \{\mathit{row}_a \mid a \in A\}$; and
- $T = \{\mathit{row}_t \mid t \in T_{\text{pre}} \cap X\}$.

Algorithm 9 MAT-Learning for regular tree languages [19].

```
    Initialize $T_{\text{pre}} := \{a\}$ for some arbitrary $a \in A^{(0)}$
 2: Initialize $C_{\text{suff}}$ to $\{\bullet\}$
    Construct the initial observation table $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$
 4: repeat
        while $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ is not closed or not consistent do
 6:         if $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ is not closed then
                find $t \in A(T_{\text{pre}})$ such that $\mathit{row}_t$ is different from $\mathit{row}_s$ for all $s \in T_{\text{pre}}$
 8:             add $t$ to $T_{\text{pre}}$
                extend $\mathit{Obs}$ to $(T_{\text{pre}} \cup A(T_{\text{pre}})) \times C_{\text{suff}}$ using membership queries
10:         if $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ is not consistent then
                find $c[t_0], c[t_1] \in A(T_{\text{pre}})$ with $t_0, t_1 \in T_{\text{pre}}$ and $\text{depth}(c) = 1$ such that
12:                     $\mathit{row}_{c[t_0]} \neq \mathit{row}_{c[t_1]}$ and $\mathit{row}_{t_0} = \mathit{row}_{t_1}$
                find $s_0, s_1 \in T_{\text{pre}}$ such that
14:                     $\mathit{row}_{s_0} = \mathit{row}_{c[t_0]}$ and $\mathit{row}_{s_1} = \mathit{row}_{c[t_1]}$
                find $c' \in C_{\text{suff}}$ such that $\mathit{row}_{t_0}(c') \neq \mathit{row}_{t_1}(c')$
16:             add $c'[c]$ to $C_{\text{suff}}$
                extend $\mathit{Obs}$ to $(T_{\text{pre}} \cup A(T_{\text{pre}})) \times C_{\text{suff}}$ using membership queries
18:     Once $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ is closed and consistent,
                    let $\mathcal{A}_{\mathit{Obs}}$ be its associated automaton and make the conjecture $\mathcal{A}_{\mathit{Obs}}$
20:     if the teacher replies with a counterexample tree $t$ then
            EXTRACT($T_{\text{pre}}, t$)
22:         extend $\mathit{Obs}$ to $(T_{\text{pre}} \cup A(T_{\text{pre}})) \times C_{\text{suff}}$ using membership queries
    until the teacher replies yes to the conjecture $\mathcal{A}_{\mathit{Obs}}$
24: return $\mathcal{A}_{\mathit{Obs}}$
```

The well-definedness of $\mathcal{A}_{\mathit{Obs}}$ is proved analogously as in Section 3.2. Furthermore, given an observation table, $\mathcal{A}_{\mathit{Obs}}$ can be constructed in time linear in the size of its transition table, i.e., time $|T_{\text{pre}}|^R \cdot |C_{\text{suff}}|$, where R is the maximum rank of A [19]. The following observation is easily proved by induction:

Observation 6.1. Let $(T_{\text{pre}}, C_{\text{suff}}, \mathit{Obs})$ be a closed and consistent observation table for trees. For all trees $t \in (T_{\text{pre}} \cup A(T_{\text{pre}}))$ and all contexts c, we have that $c[t] \in L(\mathcal{A}_{\mathit{Obs}})$ if and only if $\mathit{Obs}(t, c) = 1$. Moreover, $\mathcal{A}_{\mathit{Obs}}$ is the unique minimal bottom-up finite tree automaton with this property (up to isomorphism).

6.2 The MAT-learning algorithm for trees

The MAT-learning algorithm for regular tree languages is structured similarly to the standard MAT learning algorithm for regular word languages. However, as we will see, there are some subtleties that need to be taken into account when dealing with trees.

The algorithm is summarized in Algorithm 9. In the main loop, if the observation table is not closed, we repair the table much like in Algorithm 4. The manner in which non-

Algorithm 10 Procedure EXTRACT(T_{pre}, t) for Algorithm 9.

```
    choose context $c$ and a subtree $t_0$ of $t$ such that $t_0 \in A(T_{\text{pre}}) \setminus T_{\text{pre}}$ and $t = c[t_0]$
2:  **if** there is a $t' \in T_{\text{pre}}$ such that $row_{t'} = row_{t_0}$ and $(t \in X \Leftrightarrow c[t'] \in X)$ **then**
        EXTRACT($T_{\text{pre}}$, $c[t']$)
4:  **else**
        add $t$ to $T_{\text{pre}}$
```

consistent tables are addressed, however, is quite different. We provide some intuition. If the table $(T_{\text{pre}}, C_{\text{suff}}, Obs)$ is not consistent, then there exist trees $a(s_1, \ldots, s_n)$ and $a(s'_1, \ldots, s'_n)$ in $A(T_{\text{pre}})$ such that, for all $i \in [n]$, $row_{s_i} = row_{s'_i}$, but $row_{a(s_1,\ldots,s_n)} \neq row_{a(s'_1,\ldots,s'_n)}$. It follows that there also must be an $i \in [n]$ such that

$$row_{a(s_1,\ldots,s_{i-1},s'_i,\ldots,s'_n)} \neq row_{a(s_1,\ldots,s_i,s'_{i+1},\ldots,s'_n)},$$

since otherwise we would have that

$$row_{a(s_1,\ldots,s_n)} = row_{a(s_1,\ldots,s_{n-1},s'_n)} = \cdots = row_{a(s'_1,\ldots,s'_n)}.$$

This means that we can take $c = a(s_1, \ldots, s_{i-1}, \bullet, s'_{i+1}, \ldots, s'_n)$, $t_0 = s_i$, and $t_1 = s'_i$ and obtain trees $c[t_0], c[t_1] \in A(T_{\text{pre}})$ with $t_0, t_1 \in T_{\text{pre}}$ such that $row_{t_0} = row_{t_1}$ but $row_{c[t_0]} \neq row_{c[t_1]}$. Furthermore, we have that $\text{depth}(c) = 1$. This is exactly what is tested on lines 10–17 of Algorithm 9.

Once the observation table $(T_{\text{pre}}, C_{\text{suff}}, Obs)$ is closed and consistent, we produce a conjecture $\mathcal{A}_{Obs}$. If the teacher replies with a counterexample tree t we could, in principle, simply add t so T_{pre} and extend *Obs*. However, since extending *Obs* would require adding all subtrees of t to T_{pre}, the observation table can possibly become extremely large. Therefore, a more refined approach is presented here in which we extract from t *another* counterexample s for which $T_{\text{pre}} \cup \{s\}$ is subtree-closed. As such, only one tree needs to be added to T_{pre}. The counterexample tree s is constructed through repeated substitutions of subtrees and its construction is detailed in the procedure EXTRACT (Algorithm 10).

The EXTRACT procedure first locates a subtree t_0 of t which is in $A(T_{\text{pre}}) \setminus T_{\text{pre}}$. Since $t \notin T_{\text{pre}}$ (Observation 6.1), such a subtree must exist. Then the procedure searches a tree $t' \in T_{\text{pre}}$ such that $c[t']$ is also a counterexample (line 10). We test whether $c[t']$ is also a counterexample by testing whether $row_{t'} = row_{t_0}$ and whether $t \in X \Leftrightarrow c[t'] \in X$. This test is correct for the following reason. By construction of $\mathcal{A}_{Obs}$ and since $row_{t'} = row_{t_0}$, we have that $\Delta^*(t') = \Delta^*(t_0)$, where $\Delta^*(s)$ denotes the state of $\mathcal{A}_{Obs}$ reached at the root of s after reading s in a bottom-up fashion. Therefore, by the Myhill-Nerode theorem for trees, we have that $c[t'] \in L(\mathcal{A}_{Obs})$ if and only if $c[t_0] \in L(\mathcal{A}_{Obs})$. But, since $t = c[t_0]$ is a counterexample, we have $t \in L(\mathcal{A}_{Obs})$ if and only if $t \notin X$. This implies that $c[t'] \in L(\mathcal{A}_{Obs})$ if and only if $c[t'] \notin X$ and therefore $c[t']$ is a counterexample as well. Finally, if it is not possible to find a replacement tree t' on line 10, we add t to T_{pre} and return. In particular, if the teacher provides a counterexample, EXTRACT makes sure that at most one tree is added to T_{pre}.

6.3 Complexity

We argue that the complexity of the learner is in polynomial time in the size I of the minimal deterministic tree automaton for X and the largest counterexample t_{max} returned by the teacher.

Just as for Angluin's algorithm we always have that the tree automaton $\mathcal{A}_{Obs}$ associated with $(T_{\text{pre}}, C_{\text{suff}}, Obs)$ is the minimal bottom-up deterministic tree automaton consistent with *Obs* (Observation 6.1). By induction on the number of iterations of the repeat-until loop, one can prove that

(1) for all trees $t_1, t_2 \in T_{\text{pre}}$, $t_1 \neq t_2$, we always have that $t_1 \not\equiv_X t_2$; and
(2) we always have that $|C_{\text{suff}}| \leqslant |T_{\text{pre}}|$.

This proves that the number of rows and columns in the observation table is always polynomial in I. In particular, it also implies that we execute the bodies of the if-tests at lines 6 and 10 in Algorithm 9, and the call to EXTRACT at line 21 in Algorithm 9 at most polynomially often. Furthermore, the if-tests on lines 6 and 10 can clearly be executed in polynomial time in the size of the observation table. Finally, one call to EXTRACT also takes time polynomial in $|t_{\text{max}}|$ and I.

Theorem 6.2. *If t_{max} is the largest counterexample returned by the teacher, the running time of the learner is polynomial in I^R and $|t_{max}|$, where R denotes the maximum rank in A and I the number of equivalence classes in X.*

7 PAC learning

In an effort to initiate the study of complexity issues for machine learning tasks, Valiant introduced a formal setting for studying such problems, which was later dubbed *Probably approximately correct* (PAC) learning [42].

In Valiants terminology, the setting was originally intended for studying the learning of *concepts*. A concept can, for example, be a Boolean function over a set of variables. Positive and negative examples can be given as variable assignments, annotated with information as to whether they belong to the concept or not. In such a setting, exact learning is very hard, and Valiant therefore made his setting probabilistic. The main idea is that the set of examples should be drawn at random from some distribution D that is fixed, but unknown to the learning algorithm. The algorithm is then supposed to, with high probability, come up with a concept that gives an approximation of the concept to be learnt that is close with respect to D. In other words, for an element of the target domain, drawn at random according to D, the probability of a correct classification should be high.

The following formal definition of PAC learning is due to Angluin [5]. Let $\mathcal{U}$ be a finite or countable universe and let X, the concept to be learned, be a subset of $\mathcal{U}$. Let D be a probability distribution over $\mathcal{U}$. Let $\{X_1, X_2, \dots\}$ be a countable set of subsets of $\mathcal{U}$. This set is the hypothesis space. As a knowledge source, the learner has access to an *oracle*. When called, the oracle returns an element $x \in \mathcal{U}$, drawn according to D. It also indicates whether x belongs to X or not.

The PAC learning problem is parameterized by two positive probabilities:

(1) The *accuracy* parameter ϵ and
(2) the *confidence* parameter δ.

The learner is supposed to find, with probability at least $1 - \delta$, an index i such that the probability that X_i disagrees with X on an element $x \in \mathcal{U}$ drawn according to D is at most ϵ.

PAC learning for regular languages. Given an alphabet A and a probability distribution D over A^*, we can define a PAC learning setting for regular languages as follows. The universe is A^* and the concept to be learned is some regular language $X \subseteq A^*$. As hypothesis space, we could take the set of all DFAs over A. A PAC learning algorithm gets example words drawn from A^* according to D and annotated with the information whether they belong to X or not. To probably approximately correctly learn X with parameters ϵ and δ, the DFA $\mathcal{A}$ that the algorithm outputs should, with probability at least $1 - \delta$, be such that a word drawn according to D has probability at most ϵ of being classified incorrectly by $\mathcal{A}$. In other words, with probability at least $1 - \delta$, a random word drawn according to D should lie in the symmetric difference of X and the language of $\mathcal{A}$ with probability at most ϵ.

The above still does not say anything about the complexity of PAC learning. After considering a number of possibilities, Pitt gave the following definition of what it would mean for the regular languages to be PAC learnable with DFA as the representation of choice [36].

Definition 7.1. DFAs are PAC identifiable if and only if there exists a (possibly randomized) algorithm $\mathfrak{L}$ such that for any input parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, for any DFA $\mathcal{A}$ of size n, for any number m, and for any probability distribution D on strings of A^* of length at most m, if $\mathfrak{L}$ obtains words generated according to distribution D and labeled according to membership in $L(\mathcal{A})$, then $\mathfrak{L}$ produces a DFA $\mathcal{B}$ such that, with probability at least $1-\delta$, the probability (with respect to D) of the set $\{w \mid w \in L(\mathcal{A}) \oplus L(\mathcal{B})\}$ is at most ϵ (where $\oplus$ denotes the symmetric difference). The running time of $\mathfrak{L}$ is required to be polynomial in $n, m, \frac{1}{\epsilon}, \frac{1}{\delta}$, and $|A|$.

Notice that the above definition requires D to be a probability distribution over A^m, rather than A^*, that is, over a finite subset of A^*. This is to avoid technical difficulties in dealing with words of arbitrary length, and is justified as follows. For any probability distribution D' over the countable set A^* and for any arbitrarily small $\lambda > 0$, there is a number m such that the probability, when drawing a word from A^* according to D', the probability of the word having length larger than m is smaller than λ. Thus there is a probability distribution D over A^* that assigns zero probability to all words longer than m and approximates D' within λ.

Though PAC is a well known and widely used framework within machine learning, the results for grammatical inference have mostly been negative. In particular, a number of authors have shown the problem to be hard under various complexity-theoretic assumptions; an overview is given by Pitt [36].

Simple PAC. Since grammatical inference in the PAC model has not yielded any significant positive results, researchers have studied reasonable restrictions of the model. In particular, the PAC requirement that learning algorithms should work equally well for *any* distribution, and that the distribution is completely unknown to the algorithm, has been questioned. It can be argued that there are a number of practical settings in which the distribution is more predictable. For instance, it is not unreasonable to assume that the distribution will have a higher probability for words that are in some sense simple. The question is how to define this simplicity in a formal and uniform way. The method of choice has mostly been to use Kolmogorov complexity.

A formal definition, known as PACS, for this setting was introduced by Denis et al. [15]. It assumes the distribution over the words to be the universal distribution of Solomomoff-Levin. The authors also prove some polynomiality results for restricted classes of DFAs. Later, Parekh and Honovar showed that the full class of DFA are PACS learnable in polynomial time [34].

8 Applications and Further Material

To round off this chapter, we give a few examples of more applied settings where grammatical inference has been used successfully. We can by no means claim to give a complete view of such fields, but rather aim at giving a few, hopefully inspiring, examples and pointers.

Natural language processing Several well-known inference algorithms stem from the field of natural language processing. The algorithm Adios (for Automatic DIstillation Of Structure) derives a context-free grammar from a positive sample of unannotated sentences by searching for sequential patterns in the data [20]. A different approach is suggested by Sakakibara and Muramatsu who present a genetic algorithm (GA) for inferring context-free grammars from (partially structured) positive and negative samples. The algorithm organizes a maximal set of nonterminals in a table similar to that used in chart parsing, and then uses a GA-based technique to merge nonterminals, so as to obtain a small output grammar [39].

A polynomial time in-the-limit algorithm that infers *substitutable* context-free languages from positive examples has been presented by Clark and Eyraud [11]. A CFL L is substitutable if it adheres to Harris' principle: if a pair of word sequences constitute the same grammatical category, then they can be interchanged in any sentence without altering the grammatical correctness of the sentence [27]. Harris' principle is also the foundation for alignment-based learning (ABL), an inference framework for unsupervised learning. ABL algorithms typically operate in two phases. In the first phase, sample sentences are aligned to generate a set of hypotheses, each suggesting that a pair of sub-sentences constitute the same grammatical category. In the second phase, the most probable combination of hypotheses is selected through an expectation-maximization search [43, 23].

The algorithm Emile learns a *shallow* context-free language in the limit from set of positive examples. A context-free language is shallow if it is generated by a grammar G such that every production in G is used in to generate at least one sentence of a length that is logarithmic in the size of G. Emile works by dividing and recombining the sample sentences, so as to discover syntactical categories. For efficiency reasons, some presentations of the algorithm includes a membership oracle that can answer whether a given combination of sentence fragments is grammatically correct [1, 2].

XML schemas and web applications. In the past years, the eXtensible Markup Languages (XML) has become a very popular data format. In order to automatically process XML data, it is often beneficial to have an *XML schema* associated to the data (e.g., for automatic error detection in the data). However, it has been observed, that much of the XML data on the web does not have an accompanying schema [6].

In terms of formal languages, one can abstract XML data as *unranked* trees (i.e., trees in which each node can have arbitrarily many children) and XML schemas as tree automata. Therefore, Bex et al. investigated automatically learning XML schemas [6]. In this setting, a big challenge is that there is no negative information available, and therefore one encounters the problems discussed in Section 3.1.1. In this particular case, a solution was the observation that regular expressions in practical XML schemas are often of a very restricted form [31] and that regular expressions of this form *can* be learned by positive information alone.

Another interesting application of learning is by Carme et al. [9]. In particular, they investigate automatically learning web information extraction algorithms based from annotated examples by the user. More concretely, they learn node-selecting tree transducers which are based on stepwise tree automata, which allow a Myhill-Nerode characterization for unranked trees [32].

Verification and testing. We give a few examples where automaton learning techniques have been applied to the verification of software systems. For further applications, see the survey by Leucker [29].

In so-called *black box checking* [35, 26], the goal is to model-check an implemented system, whose internal structure is unknown. One approach to this problem uses a MAT algorithm to obtain a model for the relevant aspects of the system's structure. Here, the equivalence queries to the teacher are replaced by *conformance testing* algorithms, which compare an automaton to a "black box", i.e., to the implemented system.

Another verification area where learning techniques have been used is *compositional verification*. Chen et al. [10] give a learning algorithm that learns the minimal separating DFA for a pair of regular languages L_1 and L_2. The learner thus derives a minimal DFA M such that $L_1 \subseteq L(M)$ and $L_2 \cap L(M) = \emptyset$ by quering an extended MAT oracle. This learning algorithm is then used to infer the contextual assumption needed for the compositional verificaton. For references to other uses of learning in compositional verification, see the introduction of Chen et al. [10].

References

[1] P. Adriaans. Learning shallow context-free languages under simple distributions. Technical Report PP-1999-13, Institute for Logic, Language, and Computation, Amsterdam, 1999. 378

[2] P. W. Adriaans, M. Trautwein, and M. Vervoort. Towards high speed grammar induction on large text corpora. In *SOFSEM*, pages 173–186, 2000. 378

[3] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 5 1980. 356

[4] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75:87–106, 1987. 354, 355, 360, 361

[5] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988. 375

[6] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Trans. Datab. Syst.*, 35(2), 2010. 378

[7] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *Proc. 21st Int. Joint Conf. Artificial Intell.*, pages 1004–1009, 2009. 369, 370, 371

[8] J. Brzozowski. Derivatives of regular expressions. *J. Assoc. Comput. Mach.*, 11:481–494, 1964. 368

[9] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. *Machine Learning*, 66(1):33–67, 2007. 378

[10] Y. Chen, A. Farzan, E. Clarke, Y. Tsay, and B. Wang. Learning minimal separating dfa's for compositional verification. In *Proc. 15th Int. Conf. Tools Algor. for Construct. and Anal. of Syst.*, pages 31–45, 2009. 378

[11] A. Clark and R. Eyraud. Identification in the limit of substitutable context-free languages. In *Proc. Algorithmic Learning Theory*, volume 3734 of *LNAI*, pages 283–296. Springer, 2005. 377

[12] F. Coste and D. Fredouille. Unambiguous automata inference by means of state-merging methods. In *Proc. Eur. Conf. Mach. Learning*, pages 60–71, 2003. 370

[13] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997. 357

[14] C. de la Higuera. *Grammatical inference. Learning automata and grammars*. Cambridge University Press, 2010. 367

[15] F. Denis, C. D'Halluin, and R. Gilleron. PAC learning with simple examples. In *STACS 96, Proc. 13th Symp. Theoretical Aspects of Comp. Sci.*, pages 231–242, 1996. 377

[16] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fund. Inform.*, 51(4):339–368, 2002. 368, 369

[17] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSAs. *Theoret. Comput. Sci.*, 313(2):267–294, 2004. 368, 369, 370

[18] F. Drewes. MAT learners for recognizable tree languages and tree series. *Acta Cybernetica*, 19:249–274, 2009. 372

[19] F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In *Developments in Language Theory*, pages 279–291, 2003. 372, 373

[20] S. Edelman, Z. Solan, D. Horn, and E. Ruppin. Learning syntactic constructions from raw corpora. In *29th Boston university conference on language development*, 2005. 377

[21] P. García, M. Vázques de Parga, G. Álvarez, and J. Ruiz. Universal automata and NFA learning. *Theoret. Comput. Sci.*, 407:192–202, 2008. 371

[22] P. García, M. Vázquez de Parga, G. Álvarez, and J. Ruiz. Learning regular languages using nondeterministic finite automata. In *13th Conf. on Implementation Application Automata (CIAA)*, pages 92–101, 2008. 368, 371

[23] J. Geertzen and M. van Zaanen. Grammatical inference using suffix trees. In G. Paliouras and Y. Sakakibara, editors, *Proc. Int. Colloquium on Grammatical Inference*, volume 3264 of *LNAI*. Springer, 2004. 377

[24] E. Gold. Language identification in the limit. *Inform. Control*, 10:447–474, 1967. 357

[25] E. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978. 357, 358, 359

[26] A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. *Log. J. IGPL*, 14(5):729–744, 2006. 360, 378

[27] Z. S. Harris. *Methods in Structural Linguistics*. University of Chicago Press, Chicago, 1951. 377

[28] K. Lang, B. Pearlmutter, and R. Price. Results of the Abbadingo one DFA learning competition and a new evdence driven state merging algorihtm. In *Grammatical Inference*, pages 1–12, 1998. 367

[29] M. Leucker. Learning meets verification. In *FMCO 06: Proc. 5th Int. Symp. on Formal Methods for Components and Objects*, pages 127–151, 2006. 378

[30] L. Levy and A. Joshi. Skeletal structural descriptions. *Inform. Control*, 39:192–211, 1978. 371

[31] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009. 378

[32] W. Martens and J. Niehren. On the minimization of XML Schemas and tree automata for unranked trees. *J. Comput. System Sci.*, 73(4):550–583, 2007. 378

[33] J. Oncina and P. García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992. 364

[34] R. Parekh and V. Honavar. Learning DFA from simple examples. *Machine Learning*, 44:9–35, 2001. 377

[35] D. Peled, M. Vardi, and M. Yannakakis. Black box checking. *J. Autom. Lang. Comb.*, 7(2):225–246, 2002. 360, 378

[36] L. Pitt. Inductive inference, dfas, and computational complexity. In *AII 89: Proc. Int. Workshop on Analogical and Inductive Inference*, pages 18–44, 1989. 376

[37] L. Pitt and M. K. Warmuth. The minimum consistent dfa problem cannot be approximated within and polynomial. In *Proc. Twenty-first Ann. ACM Symp. Theor. Comput.*, 1989. 358

[38] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoret. Comput. Sci.*, 76:223–242, 1990. 371

[39] Y. Sakakibara and H. Muramatsu. Learning context-free grammars from partially structured examples. In *Proc. Int. Colloquium on Grammatical Inference*, pages 229–240. Springer, 2000. 377

[40] E. Schmidt. *Succinctness of Description of Context-Free, Regular and Unambiguous Languages*. PhD thesis, Cornell University, 1978. 370

[41] B. A. Trakhtenbrot and Y. M. Barzdin. *Finite automata*. North-Holland Publishing Co., Amsterdam, 1973. Behavior and synthesis, Translated from the Russian by D. Louvish, English translation edited by E. Shamir and L. H. Landweber, Fundamental Studies in Computer Science, Vol. 1. 364

[42] L. Valiant. A theory of the learnable. *Comm. ACM*, 27:1134–1142, 1984. 352, 375

[43] M. van Zaanen. *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis, School of Computing, University of Leeds, UK, 2001. 377

[44] M. Vázquez de Parga, P. García, and J. Ruiz. A family of algorithms for non deterministic regular languages inference. In *11th Conference on Implementation Application Automata (CIAA)*, pages 265–274, 2006. 371

[45] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2. Springer, 1997. 368