Polynomial learning of regular multi-dimensional tree languages in different settings: A meta-algorithm / a unified perspective

Anna Kasprzik (kasprzik@informatik.uni-trier.de)

Technical report 10-1 – University of Trier

Abstract. We recapitulate regular inference from membership and equivalence queries, positive and negative finite samples. We present a meta-algorithm which generalizes over as many settings involving one or more of those information sources as possible and covers the whole range of combinations allowing inference with polynomial complexity. We extend the structures that are learned to trees over arbitrarily many dimensions. The algorithm uses an observation table in order to retrieve the correct set of equivalence classes and to document the process at the same time.
Keywords: Regular inference, queries, samples, multi-dimensional trees

1 Introduction

1.1 Learning in a bounded number of steps

The area of grammatical inference is concerned with learning algorithms, i.e., algorithms that infer a description (e.g., a grammar or an automaton) for an unknown formal language from given information in finitely many steps. Various conceivable learning settings have been outlined, and based on those quite a lot of algorithms have been developed. One of the language classes studied most thoroughly with respect to algorithmical learnability so far is the class of regular languages on the Chomsky Hierarchy.

In our model of choice we assume that a learner should take a finite number of steps bounded by some measure with respect to the target and then present a solution. Motivated by several existing algorithms for the inference of a regular language L we consider four different kinds of information sources that can be accessible to a learner. Two of them consist in a teacher, or oracle, that is able to answer queries pertaining either to the membership status of an element w (*membership queries* (MQs); '$w \in L$?') or to the correctness of a description $\mathcal{A}$ for the target (*equivalence queries* (EQs); '$L = \mathcal{L}(\mathcal{A})$?'), and the teacher returns a *counterexample* $c_L(\mathcal{A}) \in (L \setminus \mathcal{L}(\mathcal{A})) \cup (\mathcal{L}(\mathcal{A}) \setminus L)$ in case of non-equivalence. The other two kinds are finite subsets of L (*positive samples*), or of its complement (*negative samples*), which in addition can fulfil certain significant properties with respect to a canonical description of L.

It has been shown (see [1–3]) that regular languages cannot be learned from one kind of query or sample only. Three well-studied combinations of two such

sources favourable to regular inference join MQs and EQs (see [4, 5]), MQs and a finite positive data set (see [6, 7]), and finite positive and negative data sets (see [8, 9]). In these cases the successful identification of L is possible with a polynomially bounded number of steps or queries depending on the size of a correct minimal target description and of the data received throughout the process.

1.2 Why multi-dimensional trees?

Most algorithms for the settings mentioned in the previous subsection have been adapted to regular tree languages (see [5, 7, 10]). This represents a first step of generalization when one conceives strings as non-branching trees with (a special symbol inserted at the beginning of the string as label for the single leaf,) the symbols of the string as node labels and the last symbol labeling the root. (Also note that consequently any negative results for strings apply to trees, and positive results for trees apply to strings as well). It is relatively easy to continue this extension to a certain generalized notion of trees over an arbitrary but fixed number of dimensions – we have done so for the setting joining MQs and EQs in [11] and we intend to complete the picture for further settings in the following.

So-called *multi-dimensional trees* were studied by Rogers (see [12, 13]) in an attempt to handle the linguistic formalism of Tree Adjoining Grammars (TAG; see [14]), which was developed in order to deal with certain non-context-free phenomena in natural languages, with regular means. Rogers mainly considers three-dimensional trees but also mentions some linguistic phenomena where four dimensions allow a more satisfactory analysis (see [12], Section 9.2). Thus, by learning multi-dimensional tree languages and then conceiving their elements as structural information (i.e., as a means to represent individual derivation processes) we can claim to be even able to infer string languages from a class beyond context-freeness. A closer study of their properties may be fruitful for computational linguistics as well as for formal language theory in general.

As Rogers contents himself with using an unpartitioned labeling alphabet, the author of this paper has developed a term-like notation for multi-dimensional trees in [15, 11] in order to permit the extension of all kinds of algorithms working with term-based tree representations to this more general kind of trees.

1.3 Content of this work

So we contribute to the completion of the picture under at least two aspects: We consider learning regular languages of tree-like structures comprising strings and classical trees, and we present a meta-algorithm intended as a generalization of existing and conceivable learning algorithms based on the retrieval of the correct set of equivalence classes under the Myhill-Nerode relation (with a polynomially bounded complexity) from a combination of the information sources introduced above. This includes a brief discussion of some combinations for which no such well-studied algorithms exist as for those named in Subsection 1.1. The meta-algorithm is what is sometimes referred to as *specializing* in the sense that it starts out with a single class which is then successively split up. It is based on the

system of an *observation table* which is a versatile and relatively abstract means to perform and document the inference process at the same time. We adapt the corresponding notions and terminology to the most general extent possible as we go along, and we have tried to factorize the various subprocedures as widely as possible as well. We give information on the (different kinds of) complexity of the algorithm for various input constellations (Subsection 3.7), and we compare four alternative methods of processing a counterexample in the Appendix.

2 Preliminaries

Multi-dimensional trees are structures over an arbitrary but fixed number of dimensions. They can be conceived as a generalization of the classical tree notion to the effect that the children of a node are not merely ordered but arranged into another multi-dimensional structure of the overall number of dimensions minus 1. Thus, the base is given by zero-dimensional trees or points, one-dimensional trees correspond to strings, and classical trees are two-dimensional, where the fact that the children of a node are arranged in a linear order is recaptured in the perspective that they form a (one-dimensional) string.[1]
We denote sequences (like tuples) using angle brackets, and the concatenation operation for sequences by the symbol '$\cdot$'. Let '$\bullet$' be a special counting symbol.

Definition 1. *Let* ${}^0\bullet := \{\bullet\}$*, and let* ${}^d\bullet$ *for* $d \geq 1$ *be the smallest set such that* $\langle\rangle \in {}^d\bullet$ *and* $\langle x_1, \ldots, x_n, y\rangle \in {}^d\bullet$ *for* $\langle x_1, \ldots, x_n\rangle \in {}^d\bullet$*,* $n \geq 1$*, and* $y \in {}^{d-1}\bullet$*. We define the set of* 0-dimensional tree domains *by* $\mathbb{T}^0 := \{\emptyset, \{\bullet\}\}$ *and the set* $\mathbb{T}^d$ *of all* d-dimensional tree domains *for* $d \geq 1$ *as the set of all finite* $\mathsf{T} \subseteq {}^d\bullet$ *satisfying* $\forall x, x' \in {}^d\bullet : x \cdot x' \in \mathsf{T} \Rightarrow x \in \mathsf{T}$ *and* $\forall y' \in {}^d\bullet : \{y'' \in {}^{d-1}\bullet \mid y' \cdot \langle y''\rangle \in \mathsf{T}\} \in \mathbb{T}^{d-1}$.

For $d \geq 1$, we define a d-dimensional tree labeling alphabet Σ^d as a set of symbols in which each symbol is associated with some $(d-1)$-dimensional tree domain. Let Σ^d_t denote the set of all symbols associated with $t \in \mathbb{T}^{d-1}$, and let ε^d denote the empty d-dimensional tree without nodes. We define the rank $\mathit{rank}(f)$ of a symbol $f \in \Sigma^d_t$ as the number of nodes in t. The set $\mathbb{T}_{\Sigma^d}$ of all *d-dimensional trees over* Σ^d for some given $d \geq 0$ and alphabet Σ^d shall be defined as follows:

Definition 2. *Let* Σ^0 *be a set of (constant) symbols, and let* $\mathbb{T}_{\Sigma^0} := \{\varepsilon^0\} \cup \Sigma^0$*. For* $d \geq 1$ *we define* $\mathbb{T}_{\Sigma^d}$ *as the smallest set of expressions such that* $\varepsilon^d \in \mathbb{T}_{\Sigma^d}$ *and* $s = f[s_1, \ldots, s_n]_t \in \mathbb{T}_{\Sigma^d}$ *for all* $f \in \Sigma^d_t$ *with* $t \in \mathbb{T}^{d-1}$ *where (uniformly throughout this work)* n *is the number of nodes in* t *and* $s_1, \ldots, s_n \in \mathbb{T}_{\Sigma^d} \setminus \{\varepsilon^d\}$

[1] Note the difference to the more common generalization from strings to trees sketched in Subsection 1.2: From the new perspective strings are not two-dimensional trees in which "accidentally" no node has more than one child but one-dimensional trees in which the child structure of every node is a (zero-dimensional) point by definition. However, we would still arrange the symbols of the string in the same way with the last symbol as label for the overall root in a conceivable formal translation of a classical string into a one-dimensional tree.

are rooted breadth-first[2] *in that order at the nodes of a* $(d-1)$*-dimensional tree over the tree domain* t*. For* $t = \langle\rangle$ *we simply write* s *as* f*. We call* $s_1, \ldots, s_n$ *the* direct subtrees *of* s*, and we define the* set of all subtrees *of* s *as* $Subt(s) := \{s\} \cup Subt(s_1) \cup \ldots \cup Subt(s_n)$*, and* $Subt(S) := \{s' \mid \exists s'' \in S : s' \in Subt(s'')\}$ *for* $S \subseteq \mathbb{T}_{\Sigma^d}$*. A node labeled by a symbol from* $\Sigma^d_{\langle\rangle}$ *is a* (d-dimensional) leaf*. The* depth $dpt(s)$ *of* s *is defined as the length of the longest path from some* d*-dimensional leaf to the root in* s*, and the* size $|s|$ *of* s *("big", "small") is defined as the number of nodes in* s*. Any set* $L \subseteq \mathbb{T}_{\Sigma^d}$ *represents a* d-dimensional tree language*.*

From here on we will assume that Σ^d is always finite. This way the maximal rank in Σ^d is fixed, which is of some importance for reflections on complexity.

Definition 3. *Let* $\square \notin \Sigma^d$ *be a special symbol associated with* $\langle\rangle$ *(a leaf label). A tree* $e \in \mathbb{T}_{\Sigma^d \cup \{\square\}}$ *in which* $\square$ *occurs exactly once is called a* context*, and the set of all contexts in* $\mathbb{T}_{\Sigma^d \cup \{\square\}}$ *is denoted by* $\mathbb{C}_{\Sigma^d}$*. For* $e' \in \mathbb{C}_{\Sigma^d}$ *and* $s \in \mathbb{T}_{\Sigma^d} \setminus \{\varepsilon^d\}$*,* $e'[\![s]\!]$ *denotes the tree obtained by substituting* s *for the leaf labeled by* $\square$ *in* e'*. For* $e'' \in \mathbb{C}_{\Sigma^d}$*, define* $cdp(e'')$ *as the length of the path from* $\square$ *to the root in* e''*. For* $L \subseteq \mathbb{T}_{\Sigma^d}$ *or* $L \subseteq \mathbb{C}_{\Sigma^d}$*,* $Cont(L) := \{e'' \in \mathbb{C}_{\Sigma^d} \mid \exists s' \in Subt(L) : e''[\![s']\!] \in L\}$*.*

Definition 4. *For* $d \geq 1$*, a* d-dimensional finite-state tree automaton (d-FTA) $\mathcal{A}$ *is a tuple* $\langle \Sigma^d, Q, F, \delta \rangle$ *where* Σ^d *is a finite* d*-dimensional tree labeling alphabet,* Q *is the finite set of states,* $F \subseteq Q$ *is the set of accepting states, and* δ *is the transition relation defined by a set of mappings of the form* $\langle f, t(q_1, \ldots, q_n)\rangle \mapsto q$ *with* $f \in \Sigma^d_t$*,* $q \in Q$*, and* $t(q_1, \ldots, q_n)$ *denoting a* $(d-1)$*-dimensional tree over* t *whose nodes are labeled breadth-first in that order by* $q_1, \ldots, q_n \in Q$*. From this relation we can derive a function* $\delta^* : \mathbb{T}_{\Sigma^d} \longrightarrow 2^Q$ *with* $\delta^*(f[s_1, \ldots, s_n]_t) = \{q \in Q \mid \exists \langle f, t(\delta^*(s_1), \ldots, \delta^*(s_n))\rangle \mapsto q \in \delta\}$*. The language accepted by* $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A}) := \{s \in \mathbb{T}_{\Sigma^d} \mid \delta^*(s) \in F\}$ *(and is classified as* recognizable *or* regular*).*
We will also write $\mathcal{A}(s') = 1$ *for* $s' \in \mathbb{T}_{\Sigma^d}$ *if* $\delta^*(s') \cap F \neq \emptyset$ *and* $\mathcal{A}(s') = 0$ *if* $\delta^*(s') \neq \emptyset \wedge \delta^*(s') \cap F = \emptyset$*, and* $\mathcal{A}(s') = *$ *if* $\delta^*(s') = \emptyset$*.*
If $\forall \langle f, t(q_1, \ldots, q_n)\rangle \mapsto q \in \delta : \neg\exists \langle f, t(q_1, \ldots, q_n)\rangle \mapsto q' \in \delta : q' \neq q$ *then* $\mathcal{A}$ *is* deterministic (a DFTA)*, and we write* $\delta^*(s'') = q''$ *for* $\delta^*(s'') = \{q''\}$*.*
If $\mathcal{A}$ *is a DFTA and for all* $t \in \mathbb{T}^{d-1}$ *with* $\Sigma^d_t \neq \emptyset$ *and all* $f \in \Sigma^d_t$*,* $q_1, \ldots, q_n \in Q$ *there is* $\langle f, t(q_1, \ldots, q_n)\rangle \mapsto q''' \in \delta$ *then* $\mathcal{A}$ *is* total*.*

Let the equivalence relation $\equiv_L$ for a language $L \subseteq \mathbb{T}_{\Sigma^d}$ be defined such that $r \equiv_L s$ for $r, s \in \mathbb{T}_{\Sigma^d}$ iff $e[\![r]\!] \in L \Leftrightarrow e[\![s]\!] \in L$ for all $e \in \mathbb{C}_{\Sigma^d}$. The *index* I_L of L is the number $|\{[s_0]_L \mid s_0 \in \mathbb{T}_{\Sigma^d}\}|$ where $[s_0]_L$ denotes the equivalence class containing s_0. The Myhill-Nerode theorem (see for example [16] for strings, and

[2] The breadth-first traversal is of course an ad hoc settlement, any other spatial arrangement based on a total ordering of the nodes of t would do as well. Although the expression 'breadth-first' may seem slightly incongruous in connection with multi-dimensionality at first glance we use it to evoke the correct intuition of an ordering where (a) a node has precedence over all other nodes it dominates with respect to any dimension, and (b) a node dominated by another node p with respect to some dimension has precedence over all nodes dominated by p with respect to a higher dimension, as opposed to 'depth-first' where in (b) precedence is reversed.

[17] for classical trees) states that I_L is finite iff L is recognizable by a finite-state automaton. With the definition of a d-FTA above it is easy to see that the theorem naturally extends to multi-dimensional trees as well. As a consequence, for every recognizable d-dimensional tree language L there is a total DFTA $\mathcal{A}_L$ with I_L states and each state recognizing a different equivalence class under $\equiv_L$. The index I_L is the smallest possible number of states in a total DFTA for L, and $\mathcal{A}_L$ is unique up to isomorphism. If $\mathbb{T}_{\Sigma^d} \setminus \mathit{Subt}(L) \neq \emptyset$ there is a unique non-total DFTA for L with one less state (i.e., it lacks the failure state) and the smallest possible number of transitions. In cases where it matters, let us denote the total minimal DFTA for L by $\mathcal{A}_L^\bullet$ and the not necessarily total minimal DFTA without a failure state for L by $\mathcal{A}_L^\circ$ (but note that for $\mathbb{T}_{\Sigma^d} \setminus \mathit{Subt}(L) = \emptyset$ $\mathcal{A}_L^\bullet$ and $\mathcal{A}_L^\circ$ coincide).

The type of learner we will consider tries to infer a state-minimal DFTA for an unknown regular tree language L over some fixed alphabet Σ^d from given information. It solves this task principally by means of an *observation table* in which it keeps track of the information it has obtained and processed so far. The rows of the table are labeled by trees from some set S, the columns are labeled by contexts from some set E.

Definition 5. *Let $L \subseteq \mathbb{T}_{\Sigma^d}$. The triple $\langle S, E, obs\rangle$ with $S \subseteq \mathbb{T}_{\Sigma^d}$ and $E \subseteq \mathbb{C}_{\Sigma^d}$ finite, E non-empty, is called an* observation table *if S is* subtree-closed*, i.e., for all $f[s_1, \ldots, s_n]_t \in \mathbb{T}_{\Sigma^d}$ with $f \in \Sigma_t^d$ and $s_1, \ldots, s_n \in \mathbb{T}_{\Sigma^d}$, if $f[s_1, \ldots, s_n]_t \in S$ then $s_1, \ldots, s_n \in S$, and $obs : \mathbb{T}_{\Sigma^d} \times \mathbb{C}_{\Sigma^d} \longrightarrow \{0, 1, *\}$ is a total function with*

$$obs(s, e) = \begin{cases} 1 & \text{if } e[\![s]\!] \in L \text{ is confirmed,} \\ 0 & \text{if } e[\![s]\!] \notin L \text{ is confirmed,} \\ & \text{if unknown.} \end{cases}$$

The row *of an element $s \in S$ is defined as $row(s) := \{(e, obs(s, e)) \mid e \in E\}$, and $row(S) := \{row(s) \mid s \in S\}$. A row or table not containing the $*$-symbol is called* complete*. Two elements $r, s \in S$ are called* obviously different *(OD; denoted by $r <> s$) iff $\exists e \in E$ such that $obs(r, e) \neq obs(s, e)$ for $obs(r, e), obs(s, e) \in \{0, 1\}$.*

The row labeling set S is further partitioned into two sets RED and BLUE (according to criteria proper to each learner), i.e., $\text{RED} \cup \text{BLUE} = S \wedge \text{RED} \cap \text{BLUE} = \emptyset$. We call the elements of $\text{RED}_{\Sigma^d}^S := \{f[s_1, \ldots, s_n]_t \in S \setminus \text{RED} \mid s_1, \ldots, s_n \in \text{RED}\}$ the *one-symbol extensions of* RED *(from S)*. As an additional condition, BLUE must contain $\text{RED}_{\Sigma^d}^S$ as a subset.During the learning process, elements are moved successively from BLUE to RED and BLUE is filled up with the smallest set of trees such that BLUE still meets the given condition from a third "supply" set WHITE.

Definition 6. *Let $T = \langle S, E, obs\rangle$ be an observation table, and $S = \text{RED} \cup \text{BLUE}$. T is* closed *iff $\neg\exists s \in \text{BLUE} : \forall r \in \text{RED} : r <> s$. T is* weakly consistent *iff, for all $t \in \mathbb{T}^{d-1}$, $f \in \Sigma_t^d$, $s_1, \ldots, s_n, s_1', \ldots, s_n' \in S$, $\neg(s_i <> s_i')$ for all i with $1 \leq i \leq n$ and $f[s_1, \ldots, s_n]_t, f[s_1', \ldots, s_n']_t \in S$ implies $\neg(f[s_1, \ldots, s_n]_t <> f[s_1', \ldots, s_n']_t)$.*

We add 'weakly' because the $*$-symbol may mask differences that are not obvious yet. Definition 7 rules out the cases in which hidden differences might prove fatal:

Definition 7. *T is* strongly consistent *iff it is* weakly consistent *and, for all $s \in S$ and all $r \in \text{RED}$: If $\neg(s <> r)$ then $row(s)$ and $row(r)$ must be complete.*

From a table $T = \langle S, E, obs\rangle$ with $S = \text{RED} \cup \text{BLUE} \neq \emptyset$ we can derive a d-FTA $\mathcal{A}_T = \langle \Sigma^d, Q_T, F_T, \delta_T\rangle$ defined by

- $Q_T := row(\text{RED})$,
- $F_T := \{row(s) \mid s \in \text{RED} \wedge obs(s, \square) = 1\}$, and
- $\delta_T := \{\langle f, t(q_1, \ldots, q_n)\rangle \mapsto q \mid f \in \Sigma_t^d \wedge \exists s_1, \ldots, s_n \in \text{RED} : \exists s' \in S :$
  $s' = f[s_1, \ldots, s_n]_t \wedge \forall i \in \{1, \ldots, n\} : q_i = row(s_i) \wedge \neg(q <> row(s'))\}$.

Note that as S is subtree-closed $\neg\exists q' \in Q_T : \mathcal{L}(q') = \emptyset$, i.e., all states can be reached. If T is not closed $\mathcal{A}_T$ cannot be total (as there is at least one one-symbol extension of RED labeling a BLUE row that is not a state of $\mathcal{A}_T$, and thus there is no transition for that symbol from the corresponding rows in RED). If T is strongly consistent then $\mathcal{A}_T$ is deterministic. A DFTA derived from a closed and strongly consistent table T is minimal with respect to the number of states, i.e., $I_{\mathcal{L}(\mathcal{A}_T)} - 1 \leq |Q_T| \leq I_{\mathcal{L}(\mathcal{A}_T)}$. (Also note that while theoretically $\mathcal{A}_T$ may have a failure state without being total this never happens with tables that are built by any concrete learning algorithm considered here.)

An aside on definitions: The definition of an observation table for strings in [4] features the additional condition that E should be *suffix-closed*, and the corresponding definition for classical trees in [5] accordingly features the condition that (a) $\forall e \in E : \{e' \in \mathbb{C}_{\Sigma^d} \mid \exists e'' \in \mathbb{C}_{\Sigma^d} : e'[\![e'']\!] = e\} \subseteq E$ ("*generalization-closedness*") and moreover requires that (b) $\forall s \in Subt(e) : s \notin \mathbb{C}_{\Sigma^d} \Rightarrow s \in \text{RED}$ ("*S-composure*"). Note that none of these properties is essential for the extraction of an automaton from a table – the only relevant function of E is to create rows that can be compared cell by cell. – The article [5], as a (in contrast to the more refined version [18]) relatively straightforward adaptation of [4], also contains a lemma stating that if T is a complete closed and strongly consistent observation table then $\mathcal{A}_T(c[\![s]\!]) = 1$ iff $c[\![s]\!] \in L$ for all $s \in S$ and all $c \in E$. However, this only holds for tables fulfilling the conditions (a) and (b) given above (otherwise there may be transitions needed to parse $c[\![s]\!]$ correctly that are not provided by RED $\cup$ BLUE). – Thus, unless $\mathcal{L}(\mathcal{A}_T) = L$ the automaton $\mathcal{A}_T$ may fail to predict the content of a cell in T correctly. However, we must note as a new lemma that the essential point from [4] and [5] stays preserved: (1) If T is (not necessarily complete or closed but) strongly consistent then $\mathcal{A}_T$ is a state-minimal DFTA for $\mathcal{L}(\mathcal{A}_T)$ with $I_{\mathcal{L}(\mathcal{A}_T)} - 1$ or $I_{\mathcal{L}(\mathcal{A}_T)}$ states. This is a direct consequence of the Myhill-Nerode theorem and of the definition of Q_T. (2) As soon as $\mathcal{L}(\mathcal{A}_T) = L$, we know that $\mathcal{A}_T$ has $I_L - 1$ or I_L states, and the strong consistency of T ensures concurrence with the transitions in $\mathcal{A}_L^\bullet$.

All learning algorithms figuring in this paper can be conceived to start out with a provisional set of equivalence classes and then try and converge to the partition induced by the equivalence $\equiv_L$ for the target language L by splitting up or merging these classes, according to the obtained information. In a table $T = \langle S, E, obs\rangle$ the set S contains trees whose rows are candidates for *states* in

the minimal DFTA for L, and E contains *experiments* proving that two elements of S do belong to distinct classes and should represent two different states.

Another concept we will need is the *subtree automaton* for a finite set of trees.

Definition 8. *Define the* subtree automaton (STA) *for a finite set* $X \subseteq \mathbb{T}_{\Sigma^d}$ *as* $STA(X) := \langle \Sigma^d, Q, F, \delta \rangle$ *with* $Q = \{\{s\} \mid s \in Subt(X)\}$, $F = \{\{s\} \mid s \in X\}$, *and* $\delta = \{\langle f, t(s_1, \ldots, s_n)\rangle \mapsto f[s_1, \ldots, s_n]_t \,|\, f \in \Sigma_t^d \wedge s_1, \ldots, s_n, f[s_1, \ldots, s_n]_t \in X\}$.

The states of $STA(X)$ are labeled by singleton sets of trees. Our meta-algorithm will unite some of these sets during the learning, according to the information that has been processed so far, so that at each step each state is labeled by the set of all trees ending in it the learner has found up to the present stage.
Finally, we will have to classify language samples that are given to the learner:

Definition 9. *A finite set* $X_+ \subseteq L$ *is* representative *for a tree language* $L \subseteq \mathbb{T}_{\Sigma^d}$ *with* $\mathcal{A}_L^\circ = \langle \Sigma^d, Q, F, \delta \rangle$ *iff for every transition* $\langle f, t(q_1, \ldots, q_n)\rangle \mapsto q \in \delta$ *there is* $f[s_1, \ldots, s_n]_t \in Subt(X_+)$ *with* $q_i = \delta^*(s_i)$ *for* $1 \leq i \leq n$, *and for all accepting states* $q_f \in F$ *there is an element* $s \in X_+$ *with* $\delta^*(s) = q_f$.
A finite set $X_- \subseteq \mathbb{T}_{\Sigma^d} \setminus L$ *is* separative *for* L *iff for all* $q_1, q_2 \in Q$ *with* $q_1 \neq q_2$ *there is* $s'' \in X_-$, $s' \in \mathbb{T}_{\Sigma^d}$, *and* $e \in \mathbb{C}_{\Sigma^d}$ *with* $s'' = e[\![s']\!]$ *such that* $\delta^*(s') = q_1 \vee \delta^*(s') = q_2$ *and, for all* $s_1, s_2 \in \mathbb{T}_{\Sigma^d}$ *with* $\delta^*(s_1) = q_1$ *and* $\delta^*(s_2) = q_2$, *we have* $(\delta^*(e[\![s_1]\!]) \in F \wedge \delta^*(e[\![s_2]\!]) \in (Q \setminus F)) \vee (\delta^*(e[\![s_2]\!]) \in F \wedge \delta^*(e[\![s_1]\!]) \in (Q \setminus F))$.

Intuitively, X_- is separative for L if for any two distinct states of $\mathcal{A}_L^\circ$ there is $s_0 \in X_-$ consisting of a subtree ending in one of them and a context proving that these states represent different classes under $\equiv_L$ based on the fact that it leads from one of them into an accepting state but not from the other.

3 Meta-algorithm GENMULTI

The input of our meta-algorithm consists of a tuple $IP = \langle EQ, MQ, X_+, X_- \rangle$ containing two Boolean values indicating if there is a teacher answering EQs and/or MQs, respectively, a positive, and a negative finite sample of the target language L in that order. We assume that the components of IP are visible as global variables throughout all procedures, as well as all other variables that are not explicitly passed on, such as the set C of all counterexamples obtained so far.[3] Let $T = \langle S, E, obs \rangle$ with $S = \text{RED} \cup \text{BLUE}$ and $\mathcal{O} = \langle \Sigma^d, Q_O, F_O, \delta_O \rangle$ always be defined by the current values of their respective components, with $obs(s, e) := *$ if the value has not been set explicitly. We also give the *smallest* alphabet Σ^d such that $L \subseteq \mathbb{T}_{\Sigma^d}$.[4]

[3] We maintain C in order to extract the maximal amount of information and thus to keep down complexity in practice since a counterexample may yield several distinctions, and some of those may not even be possible to detect at the time when the counterexample is first retrieved. Literary note: A similar idea was studied for the case of learning classical regular tree languages from MQs and EQs in [19].

[4] For reasons of convenience we assume $\varepsilon^d \notin L$. If this option is desired nevertheless it can be introduced and handled via special cases. Moreover, we assume that L

3.1 Getting started

We will now present GENMULTI step by step. The main body is simple:

```
Input: A 4-tuple IP = ⟨EQ, MQ, X+, X−⟩, an alphabet Σ^d.
Output: A d-DFTA.
1     INIT;
2     while WHITE ≠ ∅
3         if T is not closed CLOSURE
4         else NEXTDIST
5     return A_T.
```

The table T is initialized. Then, while there is still information left to process ('WHITE $\neq \emptyset$') we check for closedness and if T is closed we check if we can still find states in our current hypothesis automaton that should be split up.

```
procedure INIT
6     P := POOL;
7     O := MQORACLE;
8     RED := {a} for some arbitrary a ∈ Σ^d_⟨⟩;
9     BLUE := P ∩ (Σ^d_⟨⟩ \ {a});
10    E := {□};
11    WHITE := P \ (RED ∪ BLUE);
12    C := ∅;
13    UPDATE.
```

INIT initializes the membership oracle $\mathcal{O}$ (see procedure MQORACLE), and T. It resorts to the procedure POOL in order to obtain the set of all trees we want to consider as candidates under the given input at present. RED is the set of already processed candidates that were fixed to represent a state in the final automaton, and is initialized with a tree whose single node is labeled by an arbitrarily chosen leaf labeling symbol from $\Sigma^d_{\langle\rangle}$, whereas BLUE contains candidates representing states to which there exists a transition from a combination of states in RED. WHITE is the set of remaining candidates from which BLUE will be filled up. The value of C is set to $\emptyset$ to indicate that no counterexample has yet been retrieved. The cells of the initial table are filled by UPDATE (Subsection 3.5).

```
procedure MQORACLE
14    if MQ = 1 return O_L else return STA(X+).
```

MQORACLE returns the best membership oracle the learner can hope for at the time. For $MQ = 1$ this is trivial: We instantiate it with a total DFTA $\mathcal{O}_L$ recognizing L. Otherwise the oracle is initialized by the subtree automaton for X_+ (which is the all-rejecting automaton if $X_+ = \emptyset$). This imperfect oracle is developed further during the process every time the learner gains a new insight.

---

contains at least one tree of depth at least 1 (which also implies $d > 0$). This seems justifiable as finite languages are trivial to learn.

```
procedure POOL
15    if IP = ⟨0, 1, ∅, X_−⟩ ∧ X_− ≠ ∅
                          X_+ := {s ∈ T_{Σ^d} | dpt(s) ≤ ⌈√(2m_− + 1)⌉ ∧ O(s) = 1}
16    if EQ = 0 ∧ X_+ ≠ ∅ return Subt(X_+)
17    return {s'' ∈ T_{Σ^d} | dpt(s'') ≤ 1}.
```

POOL builds a suitable set of candidates using information available at the moment. If $EQ = 0$ and $X_+ \neq \emptyset$ then POOL returns $Subt(X_+)$ (line 16), otherwise P is initialized with the set of all trees up to depth 1. Note that if POOL uses X_+ the output automaton will not contain a failure state. If $X_- \neq \emptyset$ and $MQ = 1$ (line 15) we depend on X_- being separative because then we can extract information about the maximal number of states in $\mathcal{A}_L$ from X_- and build a representative sample via MQs: Let m_- be the number of nodes in all trees in X_- added up, which is also the maximal cardinality of $Cont(X_-)$. In the worst case, every element of $Cont(X_-)$ distinguishes a different one of the $(I_L^2 - I_L)/2$ possible state pairs. From the resulting inequation $m_- \leq (I_L^2 - I_L)/2$ we compute an upper bound for I_L (the rather ugly term $\lceil 1 + \sqrt{2m_- + 1} \rceil$) and take the set of all members of L up to that depth as a positive sample X_+ since the longest state repetition-free path in L can be at most of length $I_L - 1$. Observe that building this X_+ can take exponentially many MQs with respect to $|X_-|$.

3.2 Procedures of the main loop

```
procedure CLOSURE
18    while T is not closed
19        find s ∈ BLUE such that ∀r ∈ RED : r <> s;
20        RED := RED ∪ {s};
21        BLUE := (BLUE \ {s}) ∪
                  {s' ∈ WHITE | ∃f ∈ Σ_t^d : ∃r_1, ..., r_n ∈ RED : s' = f[r_1, ..., r_n]_t};
22        UPDATE.
```

CLOSURE is straightforward, it successively finds all elements preventing the closedness of T, moves them to RED, and calls UPDATE to fill up the table. Note that since CLOSURE is the only procedure moving elements to RED and since it only moves them if they are OD from every element in RED, the elements of RED are all pairwise OD as well, and every equivalence class of the target L does not have more than a single(!) official representative in the output.

```
procedure NEXTDIST
23    x := FINDNEXT;
24    if x ≠ ⟨ε^d, □⟩ MAKEOD(x)
25    else if MQ = 1 ∧ X_+ = ∅ WHITE := ∅
26         else BLUE := BLUE ∪ WHITE;
27              UPDATE.
```

NEXTDIST relies on T being closed and calls FINDNEXT to look for another candidate that should be fixed as a distinct state of the solution. Then T is modified by MAKEOD such that CLOSURE will move this element to RED in the

next call. If no such candidate can be found FINDNEXT returns a pair $\langle \varepsilon^d, \square \rangle$ (and thus the search for a next candidate can be seen as a test for termination). In that case WHITE is emptied for the cases in which we learn from queries only, for all other cases the remaining candidates are moved to BLUE in order not to lose the information contained in the pool and T is updated once more.

```
procedure FINDNEXT
28    if MQ = 1 ∧ (EQ = 1 ∨ X+ ≠ ∅)
29        if COMPATIBLE(A_T, O, C)
30              if ∃s0 ∈ S ∪ WHITE : ∃e0 ∈ E ∪ Cont(S ∪ WHITE) :
                    A_T(e0[[s0]]) ≠ O(e0[[s0]]) ∧ (A_T(e0[[s0]]) = * ⇒ O(e0[[s0]]) = 1)
31                        (choose a smallest s0 and e0); C := C ∪ {e0[[s0]]}
32              else if EQ = 1 ∧ EQ(A_T) = 'no' C := C ∪ {c_L(A_T)}
33                   else C := ∅
34        return MINIMIZE(PREVENT(A_T, O, C))
35    else if MQ = 0 MERGENEXT;
36             if ∃s ∈ BLUE : ∀s' ∈ BLUE : |q_s'| = 1 ⇒ s ⪯ s' (q_s' ∈ Q_O)
37                return ⟨s, □⟩
38    return ⟨ε^d, □⟩.

procedure COMPATIBLE(A, A', X) [X ⊆ T_Σ^d]
39    if ∀x ∈ X : A(x) = 1 ⇔ A'(x) = 1 return true else return false.

procedure PREVENT(A, A', X) [X ⊆ T_Σ^d]
40    if ∃x ∈ X : A(x) ≠ A'(x) ∧ A'(x) = 0 ⇒ A(x) ≠ *
41       (choose a smallest x); return x
42    else return ε^d.

procedure MINIMIZE(c)
43    if c = ε^d return ⟨ε^d, □⟩
44    if ∃e ∈ C_Σ^d : ∃s ∈ BLUE : e[[s]] = c ∧
                              ¬∃r ∈ RED : ¬(r <> s) ∧ (O(c) = 1 ⇔ O(e[[r]]) = 1)
45                                  (choose a smallest s); return ⟨s, e⟩
46    find e' ∈ C_Σ^d, s' ∈ BLUE, r' ∈ RED such that e'[[s']] = c ∧
                                  ¬(r' <> s') ∧ (O(c) = 1 ⇔ O(e'[[r']]) = 1);
47    return MINIMIZE(e'[[r']]).
```

We discuss the cases $MQ = 1$ and $MQ = 0$ in FINDNEXT separately below.

3.3 The case $MQ = 1$ in FINDNEXT

If $MQ = 1$ we can exploit a counterexample. If the set C no longer contains a counterexample (which we can check via the procedure COMPATIBLE) we try to find another one. First note that $\mathcal{A}_T$ might not predict all cells of the table correctly (see Section 2 above) and thus we can find a counterexample in T itself without asking further queries. Otherwise, the learner tries to retrieve one from an extension of T augmented with all candidates currently available from WHITE and with contexts that can be constructed using the set of candidates and E. For the setting where X_+ is used this extension corresponds to a table $T_{ext} =$

$\langle Subt(X_+), E \cup Cont(X_+), obs_{ext}\rangle$, and we can show that if X_+ is representative we can always derive a counterexample from T_{ext} because $Subt(X_+)$ must contain trees that are either OD from all RED elements or make T_{ext} inconsistent:

Lemma 1. *Let $T = \langle S, E, obs\rangle$, and let X_+ be representative for L. As long as* WHITE $\neq \emptyset$ *we can derive a counterexample for $\mathcal{A}_T$ from the extended table $T_{ext} = \langle Subt(X_+), E \cup Cont(X_+), obs_{ext}\rangle$ with T_{ext} complete.*

Proof: Observe that as T_{ext} is complete the function obs_{ext} is unambiguously defined. Either T_{ext} has the right number of distinct rows and represents a DFTA for L so that as T cannot have that number of distinct rows yet there must be at least one $s_0 \in$ WHITE OD from all RED elements. We distinguish two cases:

- $\mathcal{A}_T(s_0) \in \{0, 1\}$: As s_0 is OD from all RED elements there must be $e_0 \in E \cup Cont(X_+)$ distinguishing s_0 from the state assigned to s_0 by $\mathcal{A}_T$.
- $\mathcal{A}_T(s_0) = *$ (and consequently $\mathcal{A}_T(e_0[\![s_0]\!]) = *$): As $s_0 \in Subt(X_+)$ we can find $e_0 \in Cont(X_+)$ with $e_0[\![s_0]\!] \in X_+$ (and consequently $\mathcal{O}(e_0[\![s_0]\!]) = 1$).

In both cases $e_0[\![s_0]\!]$ is a counterexample for $\mathcal{A}_T$ (as defined in line 30 – note that we need the second condition because $\mathcal{O}$ is total and $\mathcal{A}_T$ does not have to be). If T_{ext} does *not* represent a DFTA for L we exploit the fact that it comprises the entire information provided by X_+. T_{ext} corresponds to a table that would have been built at an intermediate stage by the algorithm learning from MQs and a positive sample in [7] (or rather our adaptation to multi-dimensional trees integrated into GENMULTI), and consequently the lemmata proven in [7] can be applied to T_{ext} as well: Lemma 5 states that as soon as the table is consistent it represents a DFTA for L. Lemma 2 states that as long as the table does not represent a DFTA for L it features an inconsistency involving $s_1, \ldots, s_n, s'_1, \ldots, s'_n, s, s' \in Subt(X_+)$, $t \in \mathbb{T}^{d-1}$, $f \in \Sigma_t^d$ with $s = f[s_1, \ldots, s_n]_t$, $s' = f[s'_1, \ldots, s'_n]_t$ such that $\neg(s_i <> s'_i)$ for all i with $1 \leq i \leq n$ but $s <> s'$ with $s \in L \wedge s' \notin L$. Since $s <> s'$ there must be $e \in E \cup Cont(X_+)$ distinguishing s from s'. As T is closed and (due to the fact that all RED elements are pairwise OD and that T is complete) strongly consistent s and s' cannot be both in S, and as $\neg(s_i <> s'_i)$ for all i with $1 \leq i \leq n$, we have $\delta_T^*(s) = \delta_T^*(s')$ and $\delta_T^*(se) = \delta_T^*(s'e)$, and consequently either se or $s'e$ must be a counterexample with respect to $\mathcal{A}_T$. □

Observe that for the setting where we learn from queries only line 30 corresponds to a consistency check for a table T' with RED$'$ = RED∪BLUE and BLUE$'$ = WHITE as in that case we define WHITE as the set of all one-symbol extensions of S.

By requiring the learner to look for a smallest s_0 and e_0 (successively) we would like to hint at a possibility to process the alternatives outlined above in a favourable order (we would divide the search process into three stages, starting with a table $T'' = \langle S \cup$ WHITE$, E, obs''\rangle$ and then extending the set of experiments in two steps as suggested by line 30) in order to reduce complexity in practice.
If all these attempts fail but $EQ = 1$ an EQ is asked. If the answer is negative the learner obtains a new counterexample from the teacher. Otherwise, since we cannot find another counterexample the learning process is – successfully or not – concluded, and C must be reset to $\emptyset$ in order to pass this information on.

Any actual counterexample $c \in C$ (which can be retrieved using the procedure PREVENT) must have at least one subtree representing an undetected distinct state of the target but this subtree might not be in BLUE. FINDNEXT calls MINIMIZE to look for a BLUE subtree s of c that can be replaced by a RED tree r with the same row such that the result is *not* a counterexample since if such a subtree exists then r and s should be OD and s is returned as a suitable candidate, along with the context e fulfilling $c = e[\![s]\!]$. Otherwise MINIMIZE replaces an arbitrary BLUE subtree of c by a RED one with the same row. Note that there is at least one subtree of c in BLUE as there is at least one subtree of c in RED (a leaf, labeled by some symbol from $\Sigma^d_{\langle\rangle}$) and the non-RED one-symbol extensions of those are in BLUE as in the present cases either *all* possible non-RED one-symbol extensions of RED trees are in BLUE, or c is constructed using a tree from BLUE $\cup$ WHITE, and $S \cup$ WHITE is subtree-closed. The resulting counterexample is resubmitted to MINIMIZE until we have recursively obtained $e''[\![s'']\!]$ such that $s'' \in$ BLUE and e'' distinguishes s'' from all trees in RED.
Line 43 covers the case where there is no actual counterexample available such that MINIMIZE cannot return a suitable candidate and returns ε^d.

3.4 The case $MQ = 0$ in FINDNEXT

For $MQ = 0$ we continue improving $\mathcal{O}$ by merging states unless there is information preventing it. MERGENEXT (called in line 35, given below) retrieves all BLUE trees representing states that can be but have not yet been merged with some state with a RED tree in its label and being minimal with respect to the partial order $\preceq$ which compares trees first by depth and then by their root labels observing a lexical order on Σ^d (order is important – see Subsection 3.6). These trees can be found by searching for state labels that are singletons since labels of states resulting from a merge must contain more than one tree. The mergeability of two states is checked via COMPATIBLE using $\mathcal{O}$ with the two states merged, the total all-rejecting automaton $\mathcal{A}^{\bullet}_{\emptyset}$, and X_- as input (observe that this amounts to a test if after the merge all elements of X_- are still correctly rejected). When an eligible tree $b \in$ BLUE is found the corresponding state q_b is merged with one of the suitable states containing a RED tree in their label. The merge is done by RECMERGE which calls MERGE and recursively "repairs" any non-determinism introduced by that merge. BLUE is filled up with the available one-symbol extensions of RED $\cup \{b\}$ from WHITE and WHITE is updated. Note that b stays in BLUE but cannot be considered as a candidate again.
After the call of MERGENEXT in FINDNEXT either all BLUE trees correspond to states resulting from a merge or there is a tree that is minimal with respect to $\preceq$ and represents a non-mergeable state. This tree should be a distinct state of the solution as well and is returned along with the context $\Box$ in order to meet the requirements of the interface, but for $MQ = 0$ this second component will not be considered (see Subsection 3.5). Also note that the tests in line 44 and 36 will always fail for $X_+ = \emptyset$ since in that case $\mathcal{L}(\mathcal{O}) = \emptyset$ and BLUE $= \emptyset$.

```
procedure MERGENEXT
48    while ∃b ∈ BLUE : |q_b| = 1 ∧ ∃r ∈ RED :
                COMPATIBLE(RECMERGE(q_r, q_b, O), A^•_∅, X_−) ∧ ∀s ∈ BLUE : |q_s| = 1 ∧
                                ∃x ∈ RED : COMPATIBLE(RECMERGE(q_x, q_s, O), A^•_∅, X_−)
                                                    ⇒ b ⋠ s  (q_b, q_r, q_s, q_x ∈ Q_O)
49        O := RECMERGE(q_r, q_b, O);
50        BLUE := BLUE ∪ {s_0 ∈ WHITE | ∃t ∈ T^{d−1} : ∃f ∈ Σ^d_t : ∃r_1, ..., r_n ∈ RED :
                                                                  s_0 = f[r_1, ..., r_n]_t};
51        UPDATE.

procedure RECMERGE(p, q, A) [p, q ∈ Q_A]
52    A := MERGE(p, q, A);
53    for t ∈ T^{d−1} with Σ^d_t ≠ ∅ do
54        for f ∈ Σ^d_t do
55            D := {q_0 ∈ Q_A | ∃⟨f, t(q_1, ..., q_n)⟩ ↦ q_0 ∈ δ_A ∧
                                                          ∃i ∈ {1, ..., n} : q_i = (p ∪ q)};
56            if |D| > 1 find p′ ≠ q′ ∈ D;
57                A := RECMERGE(p′, q′, A)
58    return A.

procedure MERGE(p, q, A) [p, q ∈ Q_A]
59    q_x := p ∪ q;
60    Q_A := (Q_A \ {p, q}) ∪ {q_x};
61    if q ∈ F_A  F_A := (F_A \ {p, q}) ∪ {q_x}
62    δ_A := (δ_A \ {⟨f, t(q_1, ..., q_n)⟩ ↦ q_0 | ∃q_i ∈ {q_0, q_1, ..., q_n} : q_i ∈ {p, q}}) ∪
       {⟨f, t(q′_1, ..., q′_n)⟩ ↦ q′_0 | ∃⟨f, t(q_1, ..., q_n)⟩ ↦ q ∈ δ_A ∧ ∀q′_i ∈ {q′_0, q′_1, ..., q′_n} :
                                  (q_i ∈ {p, q} ⇒ q′_i = q_x) ∧ (q_i ∉ {p, q} ⇒ q′_i = q_i)};
63    return A.
```

3.5 Making a distinction and wrapping up

In all cases that were not covered by the distinctions in lines 28–37 we have to state that we cannot reliably find another candidate to move and return $\langle \varepsilon^d, \Box \rangle$.

```
procedure MAKEOD(⟨s, e⟩)
64    for r ∈ RED do
65        if ¬(s <> r)
66            if MQ = 1  E := E ∪ {e}; UPDATE
67            else c := PREVENT(RECMERGE(q_r, q_s, O), A^•_∅, X_−);
68                 E′ := {e′ ∈ C_{Σ^d} | ∃x ∈ {y ∈ T_{Σ^d} | δ_O(y) ∈ {q_r, q_s}} : e′⟦x⟧ = c};
69                 E := E ∪ E′; choose e_r ∈ E′;
70                 obs(r, e_r) := 1; obs(s, e_r) := 0.
```

MAKEOD is called if FINDNEXT has returned a pair $\langle s, e \rangle$ with $s \neq \varepsilon^d$ so that we know that CLOSURE should move s to RED to represent a distinct state of the final automaton. For $MQ = 1$, as the elements in RED are pairwise OD and all rows of S are complete there is only one RED element r that is *not* OD from s, and the given e is a context distinguishing s and r, so we add a single experiment

e to E and let UPDATE fill in the cells of the newly created column. Note that there are various ways to exploit a counterexample (conceivable or actually used in the literature), several of which are discussed and shown to be equivalent to our method here in Appendix A.1. For $MQ = 0$ the row of s contains only $*$'s so that we have to make s OD from every RED element r "by hand": We obtain a counterexample $c \in X_-$ that forbids the merge of q_r and q_s via PREVENT – observe that such a counterexample must exist as in all other cases with $MQ = 0$ FINDNEXT would have returned $\langle \varepsilon^d, \square \rangle$. We retrieve all subtrees of c that end up in q_r or q_s and add the resulting complementary contexts to E. Then we randomly pick one of those contexts and fill the two associated cells of r and s with differing values in order to make CLOSURE promote s to RED in the next call. Note that these values do not have to be correct with respect to L since after the next call of CLOSURE they will never be of consequence again – however, we must make sure that all RED elements receive the same value (in our case: 1) because otherwise distinctions established by previous executions of the loop in MAKEOD can be obliterated. Thus, although strictly speaking *obs* may temporarily fail to meet the condition in Definition 5 this is not fatal as the cells of T will be overwritten completely (and, in case of a successful identification of L, correctly) by the very last call of UPDATE before the algorithm terminates.

```
procedure UPDATE
71    WHITE := WHITE \ BLUE;
72    if MQ = 1 ∨ WHITE = ∅
          obs := {(s, e) ↦ o | s ∈ S ∧ e ∈ E ∧
                                  O(e⟦s⟧) = 1 ⇒ o = 1 ∧ O(e⟦s⟧) ∈ {0, *} ⇒ o = 0}
73    if MQ = 1 ∧ X+ = ∅
          WHITE := {w ∈ T_Σ^d | ∃t ∈ T^{d−1} : ∃f ∈ Σ_t^d : ∃s_1, ..., s_n ∈ BLUE :
                                                          w = f[s_1, ..., s_n]_t}.
```

UPDATE clears the elements that were moved to BLUE out of WHITE and fills in the cells of T in case we have a perfect membership oracle which for $MQ = 1$ is true at any time and for $MQ = 0$ when we have processed all the information available, provided that it was sufficient. Usually a possible change in the output of a function is assumed to be implied as soon as its domain is changed but we prefer to build *obs* explicitly as a set of mappings in order to show clearly which procedure modifies which cells of the table. For cases with $MQ = 1$ in which $X+$ is not used we have to fill up WHITE using all one-symbol extensions of BLUE (which has the effect that in these cases the output automaton will be total).

3.6 Behaviour of GENMULTI

Various parts of GENMULTI were inspired by the algorithms that were studied in [4, 5] (L^*; MQs and EQs), [7] (MQs and a positive sample), and [9, 10] (RPNI; a positive and a negative sample) but we have modified, reordered, and generalized a range of subprocedures such that at some points GENMULTI behaves in a slightly different way – for example, procedures COMPATIBLE and PREVENT were inspired by the description of RPNI given in [9] but observe how we have

respecified them in order to be able to deal with the set C of counterexamples obtained so far in FINDNEXT, which in turn has made it necessary to apply a little trick (consult the all-rejecting automaton about the negative sample X_-) in order to preserve the use of COMPATIBLE and PREVENT when testing the mergeability of states in $\mathcal{O}$ for the case where $MQ = 0$ and we use X_+.

The meta-algorithm presented here is intended as a generalization of existing and conceivable algorithms for settings where inference is possible in polynomially many steps from the given information sources under consideration, which also implies that it is deterministic and does not guess or backtrack. However, we have taken care to make it behave in an intuitively appropriate way for the remaining cases in which (polynomial) inference is not guaranteed as well.

Let us call an information source *non-void* for queries if $MQ = 1$ or $EQ = 1$, respectively, for a positive sample if it is representative, and for a negative one if it is separative. A query-based information source is *non-empty* if it is non-void.

Theorem 1. *Let L be the regular target language. GENMULTI terminates for any input after at most $2I_L - 1$ executions of the main loop. For input including at least two non-void information sources except for $\langle 1, 0, X_+, X_- \rangle$ with X_+ or X_- void the output is a state-minimal DFTA for L. For $\langle 0, 0, X_+, X_- \rangle$ with neither X_+ nor X_- void $\langle X_+, X_- \rangle$ must fulfil an additional condition (see below).*

Before we write down an actual proof we discuss each constellation individually (in more or less detail). The cases where only brief explanations are given can be easily verified by going through the algorithm step by step while observing the relevant case distinctions. Recall that for the given alphabet Σ^d, $\mathcal{A}_\emptyset$ is the all-rejecting and $\mathcal{A}_{\mathbb{T}_{\Sigma^d}}$ is the all-accepting state-minimal DFTA for $\mathbb{T}_{\Sigma^d}$.

- $\langle 0, 0, \emptyset, \emptyset \rangle$: Returns $\mathcal{A}^\bullet_\emptyset$ after one execution – the table is closed and the procedure FINDNEXT returns $\langle \varepsilon^d, \square \rangle$.
- $\langle 0, 1, \emptyset, \emptyset \rangle$: Terminates as soon as the table is closed because FINDNEXT returns $\langle \varepsilon^d, \square \rangle$. The table is closed after at most two executions, and the resulting automaton can have at most two states.
- $\langle 1, 0, \emptyset, \emptyset \rangle$: Returns $\mathcal{A}^\bullet_\emptyset$ after one execution – see $\langle 0, 0, \emptyset, \emptyset \rangle$.
- $\langle 0, 0, X_+, \emptyset \rangle$: Returns $\mathcal{A}_{\mathbb{T}_{\Sigma^d}}$ after one execution provided that every $f \in \Sigma^d$ figures at least once in X_+ – the table is closed and FINDNEXT merges all states of $\mathcal{O}$ with the result that after the call of FINDNEXT WHITE $= \emptyset$.
- $\langle 0, 0, \emptyset, X_- \rangle$: Returns $\mathcal{A}^\bullet_\emptyset$ after one execution – see $\langle 0, 0, \emptyset, \emptyset \rangle$.

Of course for $MQ = 1$ or $EQ = 1$ a learner could continue querying possible trees or DFTAs in any order, but as polynomial identification is not ensured (not even for strings – see [2, 3]) the behaviour of FINDNEXT is supposed to represent the formal equivalent of a "reasonable resignation". For the other three cases listed above the output seems appropriate as well.

- $\langle 1, 1, \emptyset, \emptyset \rangle$: We emulate the well-known algorithm L^* from [4] or rather the version for classical trees developed in [5] which we have adapted to multi-dimensionality in [11], with some more modifications that can be shown not to affect the correctness of the algorithm as we have done in Appendix A.1.

- $\langle 0, 1, X_+, \emptyset\rangle$: $Subt(X_+)$ is processed incrementally in the manner of L^* but we compute distinctions by a method such that the correctness of GENMULTI can be seen directly from the correctness proof for the algorithm in [7] (see Subsection 3.3). Note that our approach is still based on an initially given finite set of data which for $d = 1$ distinguishes it from an algorithm learning string languages from MQs and a stream of labeled data described in [20].
- $\langle 0, 0, X_+, X_-\rangle$: We emulate the algorithm RPNI [8, 9] and in parallel record our progress in a table. In this setting X_+ and X_- must fulfil an additional condition to make the learner identify. This is due to the following: We must avoid situations where we merge a state with another while it is still possible that a transition leading away from one of them may become an illegal possibility to reach an accepting state at a later stage. As a consequence, as long as not all one-symbol extensions of some subset of RED containing only trees up to a certain depth are processed we should not process trees strictly greater than those extensions with respect to $\preceq$ as otherwise the fact that after a merge all elements of X_- are still parsed correctly is not informative. Hence the given samples have to be favourable regarding the order in which we process our candidates. We have chosen an ordering by depth which at least prevents us from processing a tree before any of its subtrees. The lexical ordering of the root labels is arbitrary but must be observed equally for all depths and all roots for the same reason outlined above. Let
  - $sSubt(L) := \{s \in Subt(L) \mid \forall s' \in \mathbb{T}_{\Sigma^d} : s \equiv_L s' \Rightarrow s \preceq s'\}$ and
  - $kSubt(L) := \{f[s_1, \ldots, s_n]_t \in Subt(L) \mid s_1, \ldots, s_n \in sSubt(L)\}$.

  Suitable sets X_+ and X_- meet the following conditions (also compare [10]):
  (a) $\forall s \in kSubt(L) : \exists e \in \mathbb{C}_{\Sigma^d} : e[\![s]\!] \in X_+ \wedge (s \in L \Rightarrow e = \Box)$, and
  (b) $\forall s_1 \in sSubt(L) : \forall s_2 \in kSubt(L) : \neg(s_1 \equiv_L s_2) \Rightarrow \exists e \in \mathbb{C}_{\Sigma^d} : e[\![s_1]\!] \in X_+ \wedge e[\![s_2]\!] \in X_- \vee e[\![s_1]\!] \in X_- \wedge e[\![s_2]\!] \in X_+$.

  So for every state we need a $\preceq$-minimal tree leading up to it to represent it in the data – informally stated, otherwise one state of $\mathcal{O}$ can "overtake" another in the processing order when the latter is represented by an unnecessarily big candidate. Moreover, for all those representatives and all their one-symbol extensions (i.e., all candidates that should show up in RED and BLUE to make the learner identify), and for all non-mergeable pairs of those states and at least one suitable distinguishing context both representatives should appear with this context in the data – one in X_+ and the other in X_-. We would like to note that there is an alternative for the condition (b): For all contexts e in which one of the merging partners appears in X_+, the other partner must appear in X_- in a context e' such that any $e'_0 \in Subt(c') \cap \mathbb{C}_{\Sigma^d}$ with $cdp(e'_0) \leq 1$ is equal to or precedes any $e_0 \in Subt(e) \cap \mathbb{C}_{\Sigma^d}$ with $cdp(e_0) \leq 1$ with respect to $\preceq$, and in addition we require $e_0, e'_0 \in kSubt(L)$.

The next three cases seem of interest because to our knowledge there are no such well-studied algorithms for these settings as in the three cases listed above.

- $\langle 0, 1, \emptyset, X_-\rangle$: See $\langle 0, 1, X_+, \emptyset\rangle$. We build a positive sample X_+ (line 15 of the algorithm, see Subsection 3.2) which however may be exponential in size with

respect to $|X_-|$ so that likewise the number of MQs may not be polynomial with respect to the cardinality of the given data set in the input.

- $\langle 1, 0, X_+, \emptyset\rangle$: See $\langle 0, 0, X_+, \emptyset\rangle$. Let us suppose we wanted to handle this case in a way analogous to the previous four: We would have to test the mergeability of states in $\mathcal{O}$ via EQs. If X_+ is representative a positive counterexample reveals the existence of states that should be merged, and a negative one of states that should not have been. When we query the result of a merge (even without repairing non-determinism by further merges) and receive a positive counterexample we could either repeat the same EQ and wait for a negative one – but the number of positive ones may be infinite. Or we could query the next merge – but when (if!) we eventually get a negative counterexample we do not know which of the previous merges was illegitimate. So this method is just as complex as ignoring all counterexamples and simply querying the result of every possible *set* of merges in $\mathcal{O}$, of which there are exponentially many. Hence, since we cannot proceed as in cases where inference is ensured with a polynomial number of steps or queries we have chosen to eclipse this case from GENMULTI by the corresponding case distinctions.
- $\langle 1, 0, \emptyset, X_-\rangle$: This case is equally problematic to include in the present framework. First, observe that if X_- is separative negative counterexamples do not contribute additional information, and their number may be infinite at any step. Second, the set of positive counterexamples obtained so far may not be representative so that we cannot reliably detect an illegitimate merge because there may be accepting states of the solution that are not even represented in the current version of $\mathcal{O}$ such that the compatibility check is too weak. If we make the merge we might have to undo it on the receipt of another positive counterexample, which is a situation we want to avoid. This case is therefore eclipsed from GENMULTI by case distinctions as well.

Input containing more than two non-empty sources is treated by choosing one of the options above where for $\langle 1, 1, X_+, X_-\rangle$ with $X_+ \neq \emptyset$ the solution for MQs and EQs is preferred over the one for MQs and X_+ and $\langle 1, 0, X_+, X_-\rangle$ is treated like $\langle 0, 0, X_+, X_-\rangle$ as a result of the integrated case distinctions. There are other options, for example for $\langle 1, 1, X_+, X_-\rangle$ with $X_+ \neq \emptyset$ we could use X_+ to establish the pool and only start generating more candidates if after processing the elements of P the EQ is still answered in the negative, which might improve practical complexity. Also note that for $\langle 0, 1, X_+, X_-\rangle$ we might miss a chance to succeed if X_+ is void but X_- is not – however, as there is absolutely no way of knowing if the given data sets are non-void we have made our choice as stated. Note that for $L = \emptyset$ any representative sample must be empty, and a separative sample for a DFTA with just one state can be empty. It is easy to verify that in those cases GENMULTI returns a (trivial) correct solution as well. Also note that if GENMULTI uses a void X_+ then the output is a DFTA for a subset of L as there may be states or transitions missing. With a representative X_+ and a void X_- the output is a DFTA for a superset of L because illegitimate merges create illegal possibilities to reach accepting states.

Proof of Theorem 1. As long as the termination criterion is not fulfilled, in each loop execution either a BLUE element is moved to RED by CLOSURE or NEXTDIST adds contexts distinguishing a BLUE element from all RED ones so that it is moved to RED in the next execution. This can be seen from the discussion of the individual procedures in the previous subsections (also compare the proofs for the algorithms described in [4, 7, 9]). Consequently, GENMULTI terminates after at most $2I_L - 1$ loop executions, and each tree in RED represents a different equivalence class under $\equiv_L$. If the given information is sufficient as specified above, due to the pairwise difference of the RED elements and the completeness of the table T is strongly consistent and $\mathcal{A}_T$ deterministic. Since RED $\cup$ BLUE $= Subt(X_+)$ if we have or build a representative sample X_+ and BLUE contains *all* possible one-symbol extensions of RED otherwise no transition is missing and $\mathcal{A}_T$ is a state-minimal DFTA recognizing L (note that in the cases where X_+ is used $\mathcal{A}_T$ is isomorphic to $\mathcal{A}_L^\circ$, and to $\mathcal{A}_L^\bullet$ otherwise). □

3.7 About complexity

As mentioned before, since despite the adaptation to multi-dimensionality the notion of rank for a symbol has not been changed in its essence (number of direct subtrees) and the maximal rank ρ is still bounded by the (finite!) alphabet Σ^d there is no complexity rise to be expected in comparison to classical trees. Let us consider the complexity of GENMULTI with respect to the number of steps taken and/or queries asked for those settings in which (polynomial) identification of the target language is guaranteed. We assume that the results of MQs are stored in order not to query any tree twice (this is especially relevant for the complexity caused by calls of the procedures in FINDNEXT and of UPDATE). Let n_+ be the number of nodes in the biggest tree in X_+, and let m_+ and m_- the number of nodes of all trees in X_+ and X_- added up, respectively.

- EQs and MQs: The number of EQs needed is $O(I_L)$ because in a worst case every counterexample obtained via an EQ may reveal just one more distinct state. The number of MQs needed is $O(I_L^{4\rho} + |c_0|I_L)$ where c_0 is the biggest counterexample received because (a) T itself contains $O((I_L + |\Sigma^d|I_L^\rho)I_L) = O(I_L^{\rho+1})$ cells to be filled but, due to the somewhat cumbersome check in line 30 where we consider WHITE for the construction of both candidates and contexts, in reality we fill an extended table of size $O((I_L + |\Sigma^d|I_L^\rho + |\Sigma^d|(I_L + |\Sigma^d|I_L^\rho)^\rho)^2 \cdot I_L) = O(I_L^{4\rho})$, and (b) MINIMIZE may have to check every subtree of a counterexample for substitutability by a RED element. Note that the use of the procedure MINIMIZE can significantly reduce the number of cells in T with respect to the other methods discussed in Appendix A.1. However, see [21] for references demonstrating that by a binary search method the number of $O(|c|)$ MQs needed to derive a distinguishing context from a counterexample c can be optimized to $O(\log |c|)$.
- MQs and a representative X_+: Observe that $|Subt(X_+)| = m_+$ if no two trees in X_+ have a common subtree, and $|Cont(X_+)| = m_+$ if no two trees in $Subt(X_+)$ have the same non-empty context in X_+. The table T_{ext} has

$|Subt(X_+)|$ rows and $|Cont(X_+)| + O(I_L)$ columns, i.e., $O(m_+(m_+ + I_L)) = O(m_+^2 + I_L m_+)$ cells to be filled via MQs. As we rely on the representativity of X_+ the final table constructed by GENMULTI has $|Subt(X_+)|$ rows as well. The biggest constructed counterexample can have $O((I_L+1)n_+)$ nodes (since the size of the biggest context increases by $O(n_+)$ nodes in each call of FINDNEXT) and thus the overall MQ complexity is $O(m_+^2 + I_L m_+ + I_L^2 n_+)$.

- Representative X_+ and separative X_-: As we faithfully emulate RPNI [8, 9] GENMULTI is at least as complex as RPNI. If no two trees in X_+ have a common subtree the automaton $STA(X_+)$ has m_+ states and m_+ transitions. Hence, testing if the automaton that we are developing based on $STA(X_+)$ (wrongly) accepts a tree in X_- takes $O(m_+ m_- \rho)$ steps. We try to merge every state with a RED state ($O(m_+^2)$ steps), and we check each merge against X_-, and thus the construction of our oracle has a complexity of $O(m_+^3 m_- \rho)$. In addition we fill a table of size $O(m_+ m_-)$ using candidates from $Subt(X_+)$ and contexts from $Cont(X_-)$. Note: Due to the tree structure of the STA the computation of potential merges is polynomial (see [9]). Also note that there is an incremental version of RPNI [22] which performs better in practice.
- MQs and a separative sample: Unfortunately building X_+ can cost exponentially many MQs with respect to $|X_-|$, but this is already the case for strings.

We disregard complexities such as parsing and comparing, and we also disregard the complexity of the teacher. However, our solution takes care to ask the minimal amount of EQs possible by storing every received counterexample in C.

Remark: One might wonder why there is no (inevitable) exponential blow-up when learning (multi-dimensional) trees instead of strings, i.e., why we can keep at least some complexity measures at a polynomial bay. For the case of learning from samples this is due to the fact that we have a fixed inventory of potential subtrees (the elements of RED) and we consider only those of their one-symbol extensions that can be found in the data as well. However, note that in cases where we learn without a positive sample, if the maximal rank ρ of the alphabet is not fixed but seen as a part of the input then filling up the sets BLUE and WHITE is indeed an exponential procedure with respect to ρ. See [18] for a remedy.[5]

4 Discussion

GENMULTI represents a generalized learner for regular multi-dimensional tree languages starting out with a single equivalence class under the Myhill-Nerode relation which is then split up according to the available information. The process is executed and documented using an incrementally built observation table.[6]

We have aimed to factorize our meta-algorithm into as many subprocedures as possible in order to obtain a uniform design for different settings with their various idiosyncrasies, and hence, unlike algorithms that were developed for one

[5] Remark: Trivially, an absolute lower bound for the size of T is $\Omega((I_L + x)\log I_L)$ with $x = 0$ for cases where we use a positive sample, and $x = |\Sigma^d| I_L^\rho$ otherwise.

[6] Also see [21] for a survey of other suitable representations (for the case of strings).

setting only we cannot exploit the whole range of conceivable strategies to reduce complexity. The design of GENMULTI may cause some objectionable artefacts[7] – however, this is probably the price to pay for a generalized perspective on the common task shared by all prototypical algorithms for the different settings under consideration (the retrieval of the correct set of equivalence classes), and we have tried to work out that perspective by fitting them into a common mould with special emphasis on the basic routines that regardless of the setting have to be run through alike. This may help to formulate even clearer explanations for the interchangeability of information sources (also see [23, 24]). Note that in the majority of the cases GENMULTI is at least not more complex than certain well-established algorithms for the individual settings, and we have sketched some possibilities (such as maintaining the set of all counterexamples obtained so far) to reduce complexity in practice. Moreover, maybe an extended GENMULTI (also see the next paragraph) could be used as a template or starting point from which more algorithms, for hitherto unstudied scenarios, can be instantiated.

GENMULTI offers itself for theoretical experimentation in various directions. We could try to generalize the type of objects even more and explore possibilities such as graphs, matrices, and infinite strings or trees.[8] Then there are other kinds of information sources throughout the literature that can be of help in the computation of distinctions, such as correction queries [25], active exploration [26], distinguishing functions [27], and many more, and we could try and modify GENMULTI such that they can be given as input parameters as well. A third direction centers around the fact that we can extend the language class that is learned beyond regularity and even beyond context-freeness by conveying structural information – as we have mentioned in the Introduction, multi-dimensional trees are one way but see [28] (even linear languages, control languages) or [29] (languages recognized by certain finite-state automata with infinite transition graphs) for strategies that could be adapted rather easily to other settings by using our meta-algorithm as well, and [30] (multiple context-free grammars) as a further option to investigate. We could also study the conditions that we may impose on the given input when GENMULTI is learning certain subregular classes such as the reversible or the locally testable languages (see [31, 32]). In summary, the development of GENMULTI may be of use in the concretization of an even more general model of learning in the sense of polynomial inference as considered here – also see the very interesting current work of Clark (for example [33]).

[7] For example, when we learn from MQs and a positive sample we process the elements of $Subt(X_+)$ incrementally but keep referring to a table containing all of them, and we build a counterexample using a candidate that represents a distinct state of the solution and in general derive another candidate from it to be moved to RED (which is due to the constraint that we want to draw the representatives in RED from BLUE only). Also, when we learn from two samples we document our progress in a table which however we do not use to construct a hypothesis before the oracle is perfect.

[8] Note that as a precondition we would have to define an unambiguous concatenation operation for those objects, which might not be completely trivial to accomplish.

References

1. Gold, E.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
2. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
3. Angluin, D.: Negative results for equivalence queries. Machine Learning **5** (1990) 121–150
4. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
5. Drewes, F., Högberg, J.: Learning a regular tree language from a teacher. In: DLT 2003. Volume 2710 of LNCS., Springer (2003) 279–291
6. Angluin, D.: A note on the number of queries needed to identify regular languages. Information and Control **51** (1981) 76–87
7. Besombes, J., Marion, J.Y.: Learning tree languages from positive examples and membership queries. Theoretical Computer Science **382** (2007) 183–197
8. Oncina, J., Garcia, P.: Identifying regular languages in polynomial time. In Bunke, H., ed.: Advances in Structural and Syntactic Pattern Recognition. Volume 5 of Machine Perception and Artificial Intelligence. World Scientific (2002) 99–108
9. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
10. Oncina, J., Garcia, P.: Inference of recognizable tree sets. Technical report, DSIC II/47/93, Universidad de Valencia (1993)
11. Kasprzik, A.: A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness. In: ICGI 2008. Volume 5278 of LNAI., Springer (2008) 111–124
12. Rogers, J.: Syntactic structures as multi-dimensional trees. Research on Language and Computation **1** (2003) 265–305
13. Rogers, J.: wMSO theories as grammar formalisms. Theoretical Computer Science **293** (2003) 291–320
14. Joshi, A.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In D. Dowty, L. Karttunen, A.Z., ed.: Natural Language Processing. Cambridge University Press (1985)
15. Kasprzik, A.: Making finite-state methods applicable to languages beyond context-freeness via multi-dimensional trees. In J. Piskorski, B. Watson, A.Y., ed.: Post-Proceedings of FSMNLP 2008. IOS Press (2009) 98–109
16. Hopcroft, J., Ullmann, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Longman (1990)
17. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. (2005)
18. Drewes, F., Högberg, J.: Query learning of regular tree languages: How to avoid dead states. Theory of Computing Systems **40**(2)
19. Drewes, F., Högberg, J.: Extensions of a MAT learner for regular tree languages. In: SAIS 2006. (2006) 35–44
20. Parekh, R., Nichitiu, C., Honavar, V.: A polynomial time incremental algorithm for learning DFA. In: ICGI 1998. Volume 1433 of LNAI., Springer (1998) 37–49
21. Balcázar, J., Díaz, J., Gavaldà, R., Watanabe, O.: Algorithms for learning finite automata from queries: A unified view. In: Advances in Algorithms, Languages, and Complexity, Springer (1997) 53–72
22. Dupont, P.: Incremental regular inference. In: ICGI 1996, Springer (1996) 222–237
23. Parekh, R., Honavar, V.: On the relationship between models for learning in helpful environments. In: ICGI 2000. (2000) 207–220

24. Jain, S., Kinber, E.: Learning languages from positive data and a finite number of queries. Information and Computation **204**(1) (2006) 123–175
25. Tîrnăucă, C.: A note on the relationship between different types of correction queries. In: ICGI 2008, Springer (2008) 213–223
26. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: AII 1989, Springer (1989) 18–44
27. Fernau, H.: Identification of function distinguishable languages. Theoretical Computer Science **290**(3) (2003) 1679–1711
28. Fernau, H.: Even linear simple matrix languages: Formal language properties and grammatical inference. Theoretical Computer Science **289**(1) (2002) 425–456
29. Berman, P., Roos, R.: Learning one-counter languages in polynomial time. In: SFCS 1987, IEEE Computer Society (1987) 61–67
30. Yoshinaka, R.: Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In: ALT 2009. (2009) 278–292
31. Angluin, D.: Inference of reversible languages. JACM **29**(3) (1982) 741–765
32. Head, T., Kobayashi, S., Yokomori, T.: Locality, reversibility, and beyond: Learning languages from positive data. In: ALT 1998, Springer (1998) 191–204
33. Clark, A.: Towards general algorithms for grammatical inference. In: Proceedings of ALT. (2010)
34. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Information and Computation **118**(2) (1995) 316–326

A Appendix

A.1 Equivalence of four ways to use a counterexample for $MQ = 1$

There are various options that we could have chosen for GENMULTI of how to "milk" a counterexample for $MQ = 1$. Let us compare four of them:

Lemma 2. *For $MQ = 1$, let $T = \langle S, E, obs\rangle$ with $S = \text{RED} \cup \text{BLUE}$ be an observation table and c a counterexample for $\mathcal{A}_T$. The following four methods all lead to the existence of at least one more distinct row labeled by a* RED *element:*

a. Join $Subt(c)$ to RED.
b1. Join $Cont(\{c\})$ to E.
b2. Find $s \in$ BLUE and $e \in \mathbb{C}_{\Sigma^d}$ with $c = e[\![s]\!]$ and join $Cont(\{c\}) \setminus \{e' \in \mathbb{C}_{\Sigma^d} \mid \exists s' \in Cont(s) : e' = e[\![s']\!]\}$ to E.
c. (Procedure MINIMIZE) If there is $s \in$ BLUE and $e \in \mathbb{C}_{\Sigma^d} \setminus \{\square\}$ with $c = e[\![s]\!]$ but no $r \in$ RED with $\neg(r <> s)$ such that $e[\![r]\!]$ is a counterexample for $\mathcal{A}_T$ as well add $\{e\}$ to E. Otherwise find $s' \in$ BLUE, $e' \in \mathbb{C}_{\Sigma^d} \setminus \{\square\}$, and $r' \in$ RED with $c = e'[\![s']\!]$ and $\neg(r' <> s')$ such that $e'[\![r']\!]$ is a counterexample for $\mathcal{A}_T$ and repeat procedure MINIMIZE with input $e'[\![r']\!]$.

Proof. (a): Either such a row is created directly if E already contains a suitable separating context, or T becomes inconsistent. To see the latter, assume that no element of $Subt(c)$ is OD from every RED element. Note that in GENMULTI this occurs for $EQ = 1$ only since when we use a positive sample c is constructed using a subtree s_0 that is OD from all RED elements. We may also assume that

$\mathcal{A}_T(c) \in \{0,1\}$ since $\mathcal{A}_T$ is total for $EQ = 1$. As the automaton derived from the new table including $Subt(c)$ can obviously assign a different state to c than $\mathcal{A}_T$ although no new distinct row representing a separate state has been created this automaton must be non-deterministic, and the table inconsistent. A consistency check (which is necessary with this method) would correct that in the following loop execution by adding a context distinguishing two RED elements that have not been OD before to E, thus creating another distinct row (also see [4]).
(c): There is always a subtree of any counterexample we consider in BLUE – see Subsection 3.3 for an explanation. Suppose MINIMIZE returns $\langle s, e\rangle$ ($s \in$ BLUE). As T is closed there is $r \in$ RED with $\neg(r <> s)$ but $e[\![s]\!]$ is a counterexample whereas $e[\![r]\!]$ is not. Consequently s and r should represent distinct states such that e leads to an accepting state from one of them but there is no such accepting state for the other. Obviously s and r will be distinguished by adding e to E. Note that we could alternatively add s to RED (see [5]) but then we would need a consistency check that retrieves a separating context as is the case with method (a) because generally the RED elements would not all be pairwise OD.
(b1)/(b2): Consider (c) and the fact that the context e' added to E in (c) is an element of $Cont(\{c\}) \setminus \{e'' \in \mathbb{C}_{\Sigma^d} \mid \exists s'' \in Cont(s) : e'' = e[\![s'']\!]\} \subseteq Cont(\{c\})$. □

The equivalent of method (a) for strings is the one used in the original work by Angluin (algorithm L^*, see [4]). Method (b1) for strings was suggested in a footnote in [34]. Method (b2) is based on the reflection that there is at least one suitable context in $Cont(\{c\})$ that does not have an element of $Cont(\{s\})$ as a subtree and thus we can avoid creating redundant columns by excluding contexts with subtrees in $Cont(\{s\})$ from the set of contexts that we join to E. The method (c) which we have chosen for GENMULTI is loosely based on a method used in a version of L^* for classical trees given in [5] but among other details our method differs from the one in [5] in that we reduce the number of recursive calls to MINIMIZE by looking for an irreplaceable BLUE candidate first, and we do not add that candidate to RED immediately but pass it on along with the distinguishing context which we have derived from the counterexample.

Remark 1: Joining $Cont(\{c\})$ to E obviously introduces all distinguishing contexts that can be derived from c into the table at once but this may also lead to redundant columns. We proceed differently: We add one distinguishing context at a time but we keep c as long as it can be a counterexample and hence eventually we will have added all distinguishing contexts derivable from c as well.

Remark 2: With methods (b2) and (c) the set E easily fails to be generalization-closed and/or S-composed (see the aside in Section 2) but, as we have stated before, this is not an essential property for the extraction of an automaton from an observation table. The fact that E fulfils it both in [4, 5] (L^*; learning from MQs and EQs) and [7] (learning from MQs and a finite positive data set) is just a by-product of the way how those algorithms compute their next distinction.