Generalizing over several learning settings

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

University of Trier

Abstract. We recapitulate regular one-shot learning from membership and equivalence queries, positive and negative finite data. We present a meta-algorithm that 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. The algorithm uses the concept of an observation table as a means to perform and document the inference process at the same time.

Keywords: Regular one-shot inference, query learning, sample learning

1 Introduction

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 delineated, and based on those quite a lot of algorithms have been developed. A language class studied most thoroughly with respect to algorithmical learnability so far is the class of regular languages.

In the learning model we assume a learner should take finitely many steps and then present a solution (*one-shot learning*). Motivated by existing algorithms for the one-shot inference of a regular language L we consider four kinds of information sources that can be accessible to a learner. Two of them involve a teacher, or oracle, who is able to answer queries pertaining to the membership of an element w (MQs; '$w \in L$?') or the equivalence of a description A with the target (EQs; '$L = \mathcal{L}(A)$?', resulting in a counterexample $C_L(A) \in (L \setminus \mathcal{L}(A)) \cup (\mathcal{L}(A) \setminus L)$ in case of a negative answer). 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 the target.

[1–3] have shown that regular languages cannot be learned from one kind of query/sample only. Three well-studied combinations of two such sources favourable to regular inference involve MQs and EQs [4, 5], MQs and positive data [6, 7], and positive and negative data [8, 9]. In these cases identification is possible with a polynomial number of queries or steps depending on the size of the data received throughout the process and the chosen description of the target.

We present a meta-algorithm intended as a generalization of existing and conceivable (polynomial) one-shot algorithms based on the retrieval of the correct set of equivalence classes under the Myhill-Nerode relation from a combination of the information sources introduced above. This includes a discussion of the combinations for which no such well-studied algorithms exist as for those named in the previous paragraph. The meta-algorithm is based on the system of an *observation table* which is a useful and relatively abstract means to perform and document the inference process at the same time. We also give information on the (different kinds of) complexity of the algorithm for various input constellations.

2 Preliminaries

The type of learner we consider infers a minimal automaton for a regular string language L over some fixed alphabet Σ from given information, and solves this task principally by means of an *observation table* in which it keeps track of the obtained information it has processed so far. The rows of the table are labeled by elements from some set S, the columns by elements from some set E.

Definition 1. *A triple $T = \langle S, E, obs\rangle$ with $S, E \subseteq \Sigma^*$ finite, non-empty is an* observation table *iff S is prefix-closed ($uv \in S \Rightarrow u \in S$ for $u, v \in \Sigma^*$) and $obs : S \times E \longrightarrow \{0, 1, *\}$ is a function with*

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

For $T = \langle S, E, obs\rangle$ and $s \in S$, the row of s *is $row(s) := \{(e, obs(s, e))|e \in E\}$, and $row(S) := \{row(s)|s \in S\}$. A table/row not containing any '$*$'s is* complete.

Definition 2. *Two elements $r, s \in S$ are* obviously different *(OD; denoted by $r <> s$) iff $\exists e \in E$ such that $obs(r, e) \neq obs(s, e)$ for $obs(r, e), obs(s, e) \in \{0, 1\}$.*

S is partitioned into RED and BLUE (according to criteria proper to each learner) where BLUE $\supseteq \{sa \in S\backslash\text{RED}|s \in \text{RED}, a \in \Sigma\}$, i.e., BLUE must contain those one-symbol extensions of RED elements that are not in RED themselves. Elements are moved successively from BLUE to RED and BLUE is filled up with all the available one-symbol extensions of a moved element from a third "supply" set WHITE.

Definition 3. *T is* closed *iff $\neg\exists s \in \text{BLUE} : \forall r \in \text{RED} : r <> s$. T is* weakly consistent *iff $\forall s_1, s_2 \in \text{RED}$, $s_1a, s_2a \in S$, $a \in \Sigma : s_1a <> s_2a \Rightarrow s_1 <> s_2$.*

We add 'weakly' because the $*$-symbol may mask differences that are not obvious yet. Definition 4 rules out the cases in which hidden differences might prove fatal:

Definition 4. *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.*

Definition 5. *A* finite-state automaton *is a tuple $\mathcal{A} = \langle\Sigma, Q, q_0, \delta, F\rangle$ with finite input alphabet Σ, finite non-empty state set Q, start state $q_0 \in Q$, set of final states $F \subseteq Q$, and transition relation $\delta \subseteq (Q \times \Sigma) \times Q$, the elements of which we write as mappings $(q_1, a) \mapsto q_2$. If δ is a function the automaton is* deterministic *(a* DFA*; we use the function notation). If δ maps a state to every pair in $Q \times \Sigma$*

the automaton is total. *The transition relation can be extended to* $\delta \subseteq (Q \times \Sigma^*) \times Q$ *with* $\{(q, \varepsilon) \mapsto q\} \subseteq \delta$ *and* $\delta \cap \{(q_1, \varepsilon) \mapsto q_2 | q_1 \neq q_2\} = \emptyset$ *and* $(q_1, aw) \mapsto q_2 \in \delta$ *for* $a \in \Sigma$, $w \in \Sigma^*$ *iff* $(q_1, a) \mapsto q_3 \in \delta$ *and* $(q_3, w) \mapsto q_2 \in \delta$ *for some* q_3. *The set accepted by* $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A}) = \{s \in \Sigma^* | \exists q \in F : (q_0, s) \mapsto q \in \delta\}$ *(a* regular *language). For DFA,* $\mathcal{A}(w) = 1$ *stands short for* $\exists q \in F : (q_0, w) \mapsto q \in \delta$, $\mathcal{A}(w) = 0$ *for* $\exists q \in Q \setminus F : (q_0, w) \mapsto q \in \delta$, *and* $\mathcal{A}(w) = *$ *for* $\neg \exists q \in Q : (q_0, w) \mapsto q \in \delta$.

The equivalence relation $\equiv_L$ for a language L is defined by: $r \equiv_L s$ iff $re \in L \Leftrightarrow se \in L$ for all $r, s, e \in \Sigma^*$. The *index* of L is $I_L := |\{[s_0]_L | s_0 \in \Sigma^*\}|$ where $[s_0]_L$ denotes the equivalence class containing s_0. The Myhill-Nerode theorem (see for example [10]) states that I_L is finite iff L is recognized by a finite-state automaton, i.e., is regular. A total DFA $\mathcal{A}_L$ with I_L states and each state recognizing a different equivalence class under $\equiv_L$ is unique up to isomorphism and minimal with respect to the number of states.

From a table $T = \langle S, E, obs\rangle$ with $\varepsilon \in E$ we derive an automaton $\mathcal{A}_T = \langle \Sigma, Q_T, q_T, F_T, \delta_T\rangle$ with $Q_T = row(\text{RED})$, $q_T = row(\varepsilon)$, $F_T = \{row(s) | s \in \text{RED}, obs(s, \varepsilon) = 1\}$, and $\delta_T = \{(row(s), a) \mapsto q | \neg(q <> row(sa)),\ s \in \text{RED},\ a \in \Sigma,\ sa \in S\}$. If T is strongly consistent $\mathcal{A}_T$ is deterministic. The DFA for a language L derived from a closed and strongly consistent table has at most I_L states (see [4], Theorem 1). If this DFA is total it is isomorphic to $\mathcal{A}_L$, otherwise it may lack the "failure state" for all strings that are not a prefix in L (if they exist).

All learning algorithms mentioned 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 $\equiv_L$ 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 strings whose rows are candidates for *states* in the minimal DFA for L, and E contains *experiments* – or '*contexts*', as we will say – proving that two strings in S do belong to distinct equivalence classes and should represent two different states.

Another concept we will need is the prefix automaton for some set of strings. Let $Pref(X) := \{u \in \Sigma^* | \exists w \in X, v \in \Sigma^* : uv = w\}$, and $Suff(X) := \{u \in \Sigma^* | \exists w \in X, v \in \Sigma^* : vu = w\}$ for $X \subseteq \Sigma^*$. We write $u \preceq w$ if u is a prefix of w.

Definition 6. *The prefix automaton for* $X \subseteq \Sigma^*$ *is a (generally non-total) DFA* $PA(X) := \langle \Sigma, Q, q_0, F, \delta\rangle$ *with* $Q = \{\{x\} | x \in Pref(X)\}$, $q_0 = \{\varepsilon\}$, $F = \{\{x\} | x \in X\}$ *and* $(q_1, a) \mapsto q_2 \in \delta$ *for* $a \in \Sigma$ *iff there are* $x \in q_1$, $y \in q_2$ *such that* $y = xa$. *By* q_w *we denote a state containing* $w \in \Sigma^*$ *as an element in its label.*

The states of $PA(X)$ are labeled by singleton sets of strings. GENMODEL unites some of these sets during the process, according to the information processed so far, so that at each step each state is labeled by the set of all strings ending in it the learner has already found.

Finally, we will have to classify language samples that are given to the learner:

Definition 7. *A finite set* $X \subseteq L$ *is* representative *for a language* L *with minimal DFA* $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta\rangle$ *iff for each transition* $(q_1, a) \mapsto q_2 \in \delta$ *with* $q_1, q_2 \in Q$ *and* $a \in \Sigma$ *there are* $w \in X$ *and* $u, v \in \Sigma^*$ *such that* $w = uav$ *and* $(q_0, u) \mapsto q_1 \in \delta$, *and for all* $q \in F$ *there is* $w \in X$ *such that* $(q_0, w) \mapsto q \in \delta$.

Definition 8. *A finite set $X \subseteq \Sigma^* \setminus L$ is* separative *for a language L with minimal DFA $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ if for all $q_1 \neq q_2 \in Q$ there are $w \in X$ and $u, v \in \Sigma^*$ with $w = uv$ such that $\delta(q_L, u) = q_1 \vee \delta(q_L, u) = q_2$ and $(q_a \in F \wedge q_b \in (Q \setminus F)) \vee (q_b \in F \wedge q_a \in (Q \setminus F))$ for $(q_1, v) \mapsto q_a, (q_2, v) \mapsto q_b \in \delta$.*

Intuitively, X is representative for L if to parse the elements of X every transition of $\mathcal{A}$ has to be used at least once, and for every final state there is $w \in X$ ending in it. Note that as $X \subseteq L$ in this case $\mathcal{A}$ cannot be required to be total. Intuitively, X is separative for L if for any two distinct states of $\mathcal{A}$ there is some string $w \in X$ with a prefix leading up to one of them and a suffix proving that these states should represent different equivalence classes under $\equiv_L$ because $w \notin L$.

3 The algorithm GENMODEL

The input of our meta-algorithm consists of a tuple $IP = \langle EQ, MQ, X_+, X_- \rangle$ with two Boolean values indicating if there is a teacher answering EQs and/or MQs, a positive, and a negative finite sample of L. 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. Let $T = \langle \text{RED} \cup \text{BLUE}, E, obs \rangle$ and $O = \langle \Sigma, Q_O, q_O, F_O, \delta_O \rangle$ 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 with $L \subseteq \Sigma^*$ for the target language L.

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

```
Input: A 4-tuple $IP = \langle EQ, MQ, X_+, X_- \rangle$, an alphabet $\Sigma$.
Output: A DFA.
1   INIT;
2   while WHITE $\neq \emptyset$
3       if $T$ is not closed CLOSURE
4       else NEXTDIST
5   return $\mathcal{A}_T$.
```

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 := $\{\varepsilon\}$;
9       BLUE := $P \cap \Sigma$;
10      $E = \{\varepsilon\}$;
11      WHITE := $P \setminus (\text{RED} \cup \text{BLUE})$;
12      UPDATE.
```

INIT initializes the membership oracle O (procedure MQORACLE), and T. It resorts to the procedure POOL to obtain the set of all strings 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 single element ε (the start state), whereas BLUE contains candidates representing states to which there exists a transition from one of the states in RED. WHITE is the set of the remaining candidates from which BLUE will be filled up. The cells of the initial table are filled by the procedure UPDATE.

```
procedure POOL
13        if $IP = \langle 0, 1, \emptyset, X_- \rangle \ \wedge \ X_- \neq \emptyset \ \ X_+ := \{w \in \Sigma^* | |w| = \lceil 1 + \sqrt{2n_- + 1} \ \rceil\}$
14        if $X_+ \neq \emptyset \ \wedge \ \exists v \in X_+ : |v| \geq 2$ return $Pref(X_+)$
15        else return $\Sigma^{\leq 2}$.
```

POOL builds a suitable set of candidates using all information available at the moment. Note that if a sample is non-empty we depend on X_+ being representative and X_- separative for L. Therefore, if $X_+ \neq \emptyset$ with sufficiently long strings such that WHITE will not be empty POOL returns $Pref(X_+)$ (line 14), otherwise the pool is initialized with the set of all strings up to length 2 (line 15). Also note that if we use X_+ the output automaton will not contain a failure state. Line 13: If $X_- \neq \emptyset$ and we have MQs to exploit it we can build a representative sample from the information included in X_- about the number of states in $\mathcal{A}_L$: Let n_- be the length of all elements of X_- added up, which is also the maximal cardinality of $Suff(X_-)$. In the worst case, every suffix in X_- distinguishes a different pair of states, of which there are $(I_L^2 - I_L)/2$. From the resulting inequation $n_- \leq (I_L^2 - I_L)/2$ we compute an upper bound for I_L and take the set of all members of L up to that length as a representative sample X_+ since the longest shortest representative of a state in $\mathcal{A}_L$ is at most of length I_L. Observe that unfortunately in this case $|X_+|$ can be exponential with respect to $|X_-|$.

```
procedure MQORACLE
16        if $MQ = 1$ return $\mathcal{O}_L$ else return $PA(X_+)$.
```

MQORACLE returns the best membership oracle the learner can hope for at the time. For $MQ = 1$ this is trivial: We assume a total DFA $\mathcal{O}_L$ recognizing L. Else the oracle is initialized by the prefix automaton recognizing X_+ (which for $X_+ = \emptyset$ is the all-rejecting automaton $\mathcal{A}_\emptyset = \langle \Sigma, \{\{\varepsilon\}\}, \{\varepsilon\}, \emptyset, \emptyset \rangle$). This imperfect oracle is developed during the process every time the learner gains a new insight.

```
procedure CLOSURE
17        while $T$ is not closed
18            find $s \in$ BLUE such that $\forall s_0 \in$ RED : $s <> s_0$;
19            RED := RED $\cup \{s\}$;
20            BLUE := (BLUE $\setminus \{s\}) \cup \{s_1 \in$ WHITE$|\exists a \in \Sigma : s_1 = sa\}$;
21            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 GENMODEL 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
22        $s_x$ := FINDNEXT;
23        if $s_x \neq \langle \varepsilon, \varepsilon \rangle$ MAKEOD($s_x$)
24        else if $X_+ = \emptyset \wedge X_- = \emptyset$ $\textsc{white} := \emptyset$
25             else $\textsc{blue} := \textsc{blue} \cup \textsc{white}$;
26                  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 the next call of CLOSURE will move this element to RED. If no such candidate is to be found FINDNEXT returns a pair $\langle \varepsilon, \varepsilon \rangle$ (and can thus be seen as a test for the termination criterion). In that case WHITE is emptied for the cases in which we use information 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
27        if $MQ = 1 \wedge (EQ = 1 \vee X_+ \neq \emptyset)$
28           if $EQ = 1 \wedge$ EQ($\mathcal{A}_T$) = 'no' $c := C_L(\mathcal{A}_T)$
29           else if $X_+ \neq \emptyset \ \wedge \ \exists s_0 \in \textsc{blue} \cup \textsc{white}$ and $e_0 \in E \cup Suff(X_+)$ such
                         that $\mathcal{A}_T(s_0e_0) \neq O(s_0e_0) \ \wedge \ (\mathcal{A}_T(s_0e_0) = * \Rightarrow O(s_0e_0) = 1)$
30                 (choose $s_0$ with minimal length) $c := s_0e_0$
31           return MINIMIZE($c$)
32        else if $MQ = 0$ MERGENEXT;
33                if $\exists s \in \textsc{blue} : \forall b \in \textsc{blue} : |q_b| = 1 \Rightarrow s \preceq b$ $(q_b \in Q_O)$
34                   return $\langle s, \varepsilon \rangle$
35        return $\langle \varepsilon, \varepsilon \rangle$.
```

```
procedure MINIMIZE($c$)
36        find $s \in \textsc{blue}$, $a \in \Sigma$, $e \in \Sigma^*$ such that $c = sae$;
37        if $\exists s' \in \textsc{red} : \neg(s' <> s) \wedge (O(c) = 1 \Leftrightarrow O(s'ae) = 1) \ \wedge \ s'a \in \textsc{blue}$
38           return MINIMIZE($s'ae$)
39        else return $\langle s, ae \rangle$.
```

Various parts of FINDNEXT are inspired by the algorithms in [4, 5] (L^*; MQs & EQs), [7] (MQs & positive data), and [9] (RPNI; positive & negative data). If $MQ = 1$ we can exploit a counterexample c. If $EQ = 1$ then c can be obtained from the teacher (line 28). Else if $X_+ \neq \emptyset$ the learner tries to build c from an extension $T_{ext} = \langle S \cup \textsc{white}, E \cup Suff(X_+), obs_{ext} \rangle$ of T. We can show that this always succeeds if X_+ is representative because $S \cup \textsc{white}$ contains strings that are either OD from all RED elements or make T_{ext} inconsistent (see App. A.1). We choose a shortest prefix s_0 for c to reduce the number of MQs in MINIMIZE. At least one prefix of c must be an undetected distinct state of the solution, but as this prefix might not be in BLUE MINIMIZE is called to replace the unique[1] prefix of c in BLUE by a RED string with the same row such that the result is a

[1] There is at least one prefix of c in RED (ε) and the one-symbol extension of the longest one is in BLUE as in the present cases either $\textsc{blue} = \textsc{red} \cdot \Sigma$, or c is constructed with

counterexample as well. This is repeated as often as possible, eventually yielding a string $s'e'$ with $s' \in$ BLUE and e' distinguishes s' from all elements in RED. For $MQ = 0$ we continue the improvement of O by merging states unless there is information preventing it. MERGENEXT (called in line 40, given below) retrieves all strings representing states that can be but have not yet been merged with some other state in O and do not have a prefix *not* fulfilling this property in BLUE. These strings are found by the cardinality of the state labels containing them (labels of states resulting from a merge contain more than one string). The mergeability of two states is tested via COMPATIBLE which checks if a given automaton (here: O with the two states merged) correctly rejects all elements of X_-. When such a string $b \in$ BLUE is found the corresponding state q_b is merged with an arbitrary other state of O it can be merged with. The merge is done by RECMERGE which calls MERGE and then recursively “repairs” the possible non-determinism introduced by that merge. BLUE is filled up with the successors of b, and WHITE is updated by UPDATE. Note that b stays in BLUE. After the call of MERGENEXT either all strings in BLUE correspond to states resulting from a merge or there is a (smallest) string representing a non-mergeable state. This string should be a distinct state of the solution as well and is returned along with the arbitrary string ε in order to meet the requirements of the interface but for $MQ = 0$ this second string will not be used (see below). Also note that the tests in line 33 and 40 will always fail for $X_+ = \emptyset$ since $O = \mathcal{A}_\emptyset$ and BLUE $= \emptyset$. In all cases that were not covered by the distinctions in lines 27–34 we have to state that we cannot reliably find another candidate to move and return $\langle \varepsilon, \varepsilon \rangle$.

```
procedure MERGENEXT
40      while $\exists b \in$ BLUE : $|q_b| = 1$ ($q_b \in Q_O$) $\wedge$ $\neg\exists s \in$ BLUE : [$s \preceq b$ $\wedge$ $\neg\exists t \in$ RED
                such that COMPATIBLE(RECMERGE($q_t, q_s, O$)) ($q_t, q_s \in Q_O$)]
41          find $r \in$ RED such that COMPATIBLE(RECMERGE($q_r, q_b, O$));
42          $O$ := RECMERGE($q_r, q_b, O$);
43          BLUE := BLUE $\cup \{w \in$ WHITE$|\exists a \in \Sigma : w = ba\}$;
44          UPDATE.

procedure COMPATIBLE($A$)
45      if $A(w) = 1$ for some $w \in X_-$ return false else return true.

procedure RECMERGE($q_1, q_2, A$) [$q_1, q_2 \in Q_A$]
46      $A$ := MERGE($q_1, q_2, A$);
47      for $a \in \Sigma$ do
48          if $|D = \{q \in Q_A|((q_1 \cup q_2), a) \mapsto q \in \delta_A\}| > 1$ find $q_a \neq q_b \in D$;
49              $A$ := RECMERGE($q_a, q_b, A$)
50      return $A$.

procedure MERGE($q_1, q_2, A$) [$q_1, q_2 \in Q_A$]
51      $q_x := q_1 \cup q_2$;
52      $Q_A := (Q_A \setminus \{q_1, q_2\}) \cup \{q_x\}$;
53      if $q_2 \in F_A$ $F_A := (F_A \setminus \{q_1, q_2\}) \cup \{q_x\}$
```

a prefix from BLUE $\cup$ WHITE, and $S \cup$ WHITE is prefix-closed. Since in those cases BLUE $= \{sa \in S \setminus$ RED$|s \in$ RED$, a \in \Sigma\}$ there is only one such extension in BLUE.

```
54        if $q_1 = q_A$  $q_A := q_x$
55        $\delta_A := (\delta_A \setminus (\{(q,a) \mapsto q_y | q \in Q_A, a \in \Sigma, y \in \{1,2\}\} \cup$
                         $\{(q_y,a) \mapsto q | q \in Q_A, a \in \Sigma, y \in \{1,2\}\})) \cup$
                  $\{(q,a) \mapsto q_x | (q,a) \mapsto q_y \in \delta, q \in Q_A, a \in \Sigma, y \in \{1,2\}\} \cup$
                  $\{(q_x,a) \mapsto q | (q_y,a) \mapsto q \in \delta, q \in Q_A, a \in \Sigma, y \in \{1,2\}\}$;
56        return $A$.
```

```
procedure MAKEOD($\langle s,e \rangle$)
57        for $r \in$ RED do
58           if $\neg(s <> r)$
59              if MQ = 1  $E := E \cup \{e\}$
60              else $c$ := PREVENTMERGE($q_r, q_s, O$);
61                    find $x \in \{y | \delta_O(q_\varepsilon, y) = q_r\} \cup \{s\}$, $e_r \in \Sigma^*$ with $c = xe_r$;
62                    $E := E \cup \{e_r\}$;
63                    $obs(r, e_r) := 1$; $obs(s, e_r) := 0$.
```

```
procedure PREVENTMERGE($q_1, q_2, A$)  [$q_1, q_2 \in Q_A$]
64        $A :=$ RECMERGE($q_1, q_2, A$);
65        return $w \in X_-$ such that $A(w) = 1$.
```

MAKEOD is called if FINDNEXT has returned a pair $\langle s,e \rangle$ with $s \neq \varepsilon$ so that we know that s should be moved to RED as a distinct state of the target by CLOSURE. For $MQ = 1$ there is only one RED element r that is *not* OD from s (as the RED elements are all pairwise OD, and all rows of S are complete), and the given e is a context distinguishing s and r, so we add e to E. For $MQ = 0$ the row of s contains only '$*$'s and we have to make s OD from every RED element r "by hand": We find a counterexample $c \in X_-$ that prevents the merge of q_r and q_s using PREVENTMERGE and a suffix e_r of c leading from q_r or q_s to a final state (note that $X_- \neq \emptyset$ as in all other cases with $MQ = 0$ FINDNEXT returns $\langle \varepsilon, \varepsilon \rangle$). As c should not be accepted e_r is a context distinguishing s and r. We add e_r to E and fill the two corresponding cells of T with differing values – note that they do not have to be correct as they are only once compared in the next call of CLOSURE, and T will be updated completely just before termination.

```
procedure UPDATE
66        WHITE := WHITE $\setminus$ BLUE;
67        if MQ = 1 $\vee$ WHITE $= \emptyset$  $obs := \{(s,e) \mapsto O(se) | s \in$ RED $\cup$ BLUE, $e \in E\}$
68        if IP = (EQ, $1, \emptyset, \emptyset$) WHITE := BLUE $\cdot\, \Sigma$.
```

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 available information, provided that it was sufficient. For the cases with empty samples but $MQ = 1$ we have to fill up WHITE with all one-symbol extensions of BLUE (which has the effect that in these cases the output automaton will be total).

GENMODEL is intended as a generalization of existing and conceivable algorithms for those settings where one-shot 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 a way that can be intuitively called appropriate for cases where (polynomial) one-shot inference is not possible as well.

We call an information source *non-void* for queries if $MQ = 1/EQ = 1$, for a positive sample if it is representative, and for a negative sample if it is separative.

Theorem 1. *a. Let L be the regular target language. GENMODEL terminates for any input after at most $2I_L - 1$ main loop executions and returns a DFA.*
b. For any 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 minimal DFA for L.

Proof. We will 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 observing the relevant case distinctions.

- $\langle 0, 0, \emptyset, \emptyset \rangle$: Returns $\mathcal{A}_\emptyset$ after one execution (T is closed, FINDNEXT $= \langle \varepsilon, \varepsilon \rangle$).
- $\langle 0, 1, \emptyset, \emptyset \rangle$: Terminates as soon as T is closed as FINDNEXT $= \langle \varepsilon, \varepsilon \rangle$. T is closed after at most two executions, and $\mathcal{A}_T$ can have at most two states.
- $\langle 1, 0, \emptyset, \emptyset \rangle$: Returns $\mathcal{A}_\emptyset$ after one execution – see $\langle 0, 0, \emptyset, \emptyset \rangle$.
- $\langle 0, 0, X_+, \emptyset \rangle$: Returns $\mathcal{A}_{\Sigma^*}$ after one execution given that every $a \in \Sigma$ figures in X_+ (T is closed, FINDNEXT merges all states of O so that WHITE $= \emptyset$).
- $\langle 0, 0, \emptyset, X_- \rangle$: Returns $\mathcal{A}_\emptyset$ after one execution – see $\langle 0, 0, \emptyset, \emptyset \rangle$.

Of course for $MQ = 1$ or $EQ = 1$ a learner can continue querying possible strings or DFAs in any order, but as polynomial identification is not guaranteed [2, 3] the behaviour of FINDNEXT is supposed to represent "reasonable resignation". For the other three cases listed above the output seems rather intuitive as well.

- $\langle 1, 1, \emptyset, \emptyset \rangle$: We emulate the well-known algorithm L^* [4] except that instead of adding the prefixes of a counterexample to S we add a single suffix to E, which however does not affect the correctness of the algorithm (see Appendix A.2). This version needs $O(I_L)$ EQs and $O(I_L^2 \cdot |\Sigma| + I_L \cdot n)$ MQs where n is the length of the longest counterexample (for details see Appendix A.3).
- $\langle 0, 1, X_+, \emptyset \rangle$: $\mathit{Pref}(X_+)$ is processed incrementally in the manner of L^*, but we rely on a lemma proven for the algorithm in [7] to compute distinctions (see Appendix A.1). This constellation needs $O(n_+^2 + n_+ \cdot I_L)$ MQs where n_+ is the sum of the lengths of all strings in X_+ (see Appendix A.3).
- $\langle 0, 0, X_+, X_- \rangle$: We emulate the algorithm RPNI [8, 9] and in addition record our progress in a table. The overall complexity amounts to $O(n_+^3 \cdot n_-)$ steps where n_- is the sum of the lengths of all strings in X_- (see Appendix A.3).

The next three cases are 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 (line 13, see above) which however may be exponential in size with respect to $|X_-|$ so that the number of MQs is not polynomial with respect to the size of the given data.

- $\langle 1, 0, X_+, \emptyset\rangle$: See $\langle 0, 0, X_+, \emptyset\rangle$. Let us suppose we wanted to handle this case in a similar way as the previous four: We would have to test the mergeability of states in 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 not less complex than ignoring all counterexamples and simply asking an EQ for the result of every possible *set* of merges in O, of which there are exponentially many. Therefore, since we cannot proceed as in the cases where inference is possible with a polynomial number of steps or queries this case is eclipsed from GENMODEL 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 final states of the solution that are not even represented in the current version of O such that the compatibility check is too weak. If we make the merge we might have to undo it on the reception of another positive counterexample, which is a situation we want to avoid. This case is therefore eclipsed from GENMODEL by case distinctions as well.

Input containing more than two non-empty sources is treated by choosing one of the options above where the solution for MQs and EQs is preferred over the one for MQs and a positive sample as a result from the integrated case distinctions.

Note that for $L = \emptyset$ any representative sample must be empty, and a separative sample for a DFA with just one state can be empty as well. It is easy to verify that in those cases GENMODEL also returns the correct solution.

Theorem 1b: As long as the termination criterion is not met, 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 above (also see the proofs of the emulated algorithms in [4, 7, 9]). Consequently, GENMODEL terminates after at most $2I_L - 1$ loop executions, and each string in RED represents a different equivalence class under $\equiv_L$. 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 $\text{RED} \cup \text{BLUE} = \mathit{Pref}(X_+)$ if we have or build a representative sample X_+ and $\text{BLUE} = \text{RED} \cdot \Sigma$ in the other cases no transition is missing and $\mathcal{A}_T$ is a minimal DFA for L. $\square$

4 Conclusion

GENMODEL represents a generalized learner starting out with a single equivalence class under the Myhill-Nerode relation which is then split up according to the obtainable information (sometimes referred to as a *specializing* algorithm). The process is executed and documented using an observation table which is built incrementally.[2] We have aimed to design GENMODEL as modular as possible as an inventory of the essential procedures in existing and conceivable polynomial one-shot regular inference algorithms of the considered kind. This may help to give clearer explanations for the interchangeability of information sources (see for example [15, 16]). Practically, an extended GENMODEL (see below) could be used as a template from which individual algorithms for hitherto unstudied scenarios can be instantiated (however, in concrete implementations one might prefer to minimize complexity by making more case distinctions, whereas we have taken care to minimize case distinctions in order to be as universal as possible).

GENMODEL offers itself to be extended in several directions. We could try to generalize over the type of objects: An adaptation to trees suggests itself as many of the integrated algorithms have been adapted to (multi-dimensional) trees already [5, 7, 17, 18], a main challenge probably being complexity. Further possibilities are graphs, matrices, and infinite strings. Then there are other kinds of sources of essential information for the computation of distinctions which might be integratable here, such as correction queries [19], active exploration [20], distinguishing functions [21], and many more. The third direction concerns an extension of the learned language class beyond regularity (for example by using strategies as in [22] for even linear languages, or [23] for languages recognized by DFA with infinite transition graphs) and even beyond context-freeness [22, 24]. The development of GENMODEL may be of use in the concretization of an even more general model of learning in the sense of polynomial one-shot inference as considered here – also see the very interesting current work of Clark [25, 26].

References

1. Gold, E.: Language identification in the limit. Inf. & Contr. **10**(5) (1967) 447–474
2. Angluin, D.: Queries and concept learning. Mach. L. **2** (1988) 319–342
3. Angluin, D.: Negative results for equivalence queries. Mach. L. **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) 279–291

[2] We have chosen observation tables as a sufficiently abstract, intuitive, and versatile concept: From a completed table we can derive a DFA, but also other descriptions like an NFA (see [11]), or a grammar. For a survey of similar representations see [12]. Note that GENMODEL recasts some existing algorithms (RPNI and the one in [6]) in an incremental form in connection with a table. However, see [13] for an incremental RPNI, and [14] for learning from a stream of labeled examples and MQs.

6. Angluin, D.: A note on the number of queries needed to identify regular languages. Inf. & Contr. **51** (1981) 76–87
7. Besombes, J., Marion, J.Y.: Learning tree languages from positive examples and membership queries. In: ALT 2003. (2004) 440–453
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. Hopcroft, J., Ullmann, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Longman (1990)
11. Kasprzik, A.: Learning residual finite-state automata using observation tables. Technical report, University of Trier (2009)
12. 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. (1997) 53–72
13. Dupont, P.: Incremental regular inference. In: ICGI. (1996) 222–237
14. Parekh, R., Nichitiu, C., Honavar, V.: A polynomial time incremental algorithm for learning DFA. In: ICGI. (1998) 37–49
15. Parekh, R., Honavar, V.: On the relationship between models for learning in helpful environments. In: ICGI. (2000) 207–220
16. Jain, S., Kinber, E.: Learning languages from positive data and a finite number of queries. Information and Computation **204**(1) (2006) 123–175
17. Oncina, J., Garcia, P.: Inference of recognizable tree sets. Technical report, DSIC II/47/93, Universidad de Valencia (1993)
18. Kasprzik, A.: A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness. In: ICGI. (2008) 111–124
19. Tîrnăucă, C.: A note on the relationship between different types of correction queries. In: ICGI. (2008) 213–223
20. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: AII. (1989)
21. Fernau, H.: Identification of function distinguishable languages. Theoretical Computer Science **290**(3) (2003) 1679–1711
22. Fernau, H.: Even linear simple matrix languages: Formal language properties and grammatical inference. Theoretical Computer Science **289**(1) (2002) 425–456
23. Berman, P., Roos, R.: Learning one-counter languages in polynomial time. In: SFCS. (1987) 61–67
24. Yoshinaka, R.: Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In: ALT. (2009) 278–292
25. Clark, A.: A learnable representation for syntax using residuated lattices. In: Formal Grammar. (2009)
26. Clark, A.: Three learnable models for the description of language. (LATA 2010)

A Appendix

A.1 T_{ext} contains a counterexample for $\mathcal{A}_T$

Lemma 1. *Let $T = \langle S, E, obs \rangle$ and X_+ representative. As long as* WHITE $\neq \emptyset$ *$T_{ext} = \langle S \cup$ WHITE$, E \cup Suff(X_+), obs_{ext} \rangle$ contains a counterexample for $\mathcal{A}_T$.*

Proof: Either T_{ext} has $I_L - 1$ distinct rows and represents the solution so that as T cannot contain $I_L - 1$ distinct rows yet there must be at least one string $s_0 \in$ WHITE that is 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 $e_0 \in E \cup Suff(X_+)$ distinguishing s_0 from the state assigned to s_0 by $\mathcal{A}_T$.
– $\mathcal{A}_T(s_0) = *$ (and consequently $\mathcal{A}_T(s_0e_0) = *$): Since $s_0 \in Pref(X_+)$ we can find $e_0 \in Suff(X_+)$ such that $s_0e_0 \in X_+$ (and consequently $O(s_0e_0) = 1$).

In both cases s_0e_0 is a counterexample for $\mathcal{A}_T$ (as defined in line 29 – note that we need the second condition because O is total and $\mathcal{A}_T$ generally is not).
If T_{ext} does *not* represent the solution we use the fact that T_{ext} contains the entire information provided by X_+ and corresponds to a table built by an algorithm learning from MQs and positive data described in [7] (adapted to strings by conceiving them as non-branching trees), and that consequently the lemmata applying to this table can be applied to T_{ext} as well: By contraposition of Lemma 4 in [7] (stating that as soon as the table is consistent it represents the solution) T_{ext} has an inconsistency involving $s_1, s_2 \in Pref(X_+)$, $a \in \Sigma$ such that $\neg(s_1 <> s_2)$ but $s_1a <> s_2a$. As T is closed, either $s_1a \in$ WHITE or $s_2a \in$ WHITE and can be taken as s_0. As $s_1a <> s_2a$ there must be $e_0 \in E \cup Suff(X_+)$ separating s_1a from s_2a, and either s_1ae_0 or s_2ae_0 is a counterexample for $\mathcal{A}_{T_{ext}}$ because $\neg(s_1 <> s_2)$ in T_{ext} although $O(s_1ae_0) \neq O(s_2ae_0)$. As T_{ext} contains at least as much information as T any counterexample for $\mathcal{A}_{T_{ext}}$ is one for $\mathcal{A}_T$ as well. □

A.2 Equivalence of four ways to use a counterexample for $MQ = 1$

Lemma 2. *Let T be an observation table, $MQ = 1$, and c a counterexample for $\mathcal{A}_T$. These methods all lead to the existence of one more distinct row in* RED*:*

(a) Add $Pref(c)$ to RED*.*
(b1) Add $Suff(c)$ to E.
(b2) Find $s \in$ BLUE, $e \in \Sigma^+$ with $c = se$ and add $Suff(e)$ to E.
(c) (Procedure MINIMIZE) Find $s \in$ BLUE, $e \in \Sigma^+$ with $c = se$. If there is $r \in$ RED with $\neg(r <> s)$ and re is a counterexample for $\mathcal{A}_T$ as well, replace s by r. Repeat this for the resulting string until no such r can be found. Let the result be $s'e'$ with $s' \in$ BLUE and $e' \in \Sigma^+$. Add e' to E.

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 $Pref(c)$ is OD from every RED element. Note that in GENMODEL this occurs for $EQ = 1$ only since when we use a positive sample c is constructed

with a prefix 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 containing $\mathit{Pref}(c)$ 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, thus creating another distinct row (also see [4]).
(c): As T is closed $s' \in$ BLUE has a match $r' \in$ RED but $s'e'$ is a counterexample whereas $r'e'$ is not. Consequently s' and r' should represent distinct states such that e' leads to a final state from one of them but there is no such final state for the other. Obviously s' and r' will be distinguished by adding e' to E.
(b1)/(b2): Consider (c) and the fact that $e' \in \mathit{Suff}(e) \subseteq \mathit{Suff}(c)$. □

Method (b1) was proposed in a footnote in Maler&Pnueli (1995).[3] Note that adding $\mathit{Suff}(c)$ finds *all* distinguishing suffixes of c, which may save some EQs but may also lead to redundant columns and thus unnecessary MQs. Method (c) is a combination of (b1)/(b2) and a method from a version of L^* for trees [5].

Remark: With (b2) and (c) E may not be suffix-closed anymore, which however is a not an essential property for the extraction of an automaton from an observation table. The fact that E fulfils it both in [4] and [7] is just a by-product of the way how those two algorithms compute distinguishing contexts.

A.3 Complexity of GENMODEL for different input constellations

- $\langle 1, 1, \emptyset, \emptyset \rangle$: The number of EQs needed is $O(I_L)$ because in a worst case every counterexample obtained via an EQ reveals just one more distinct state.
  The number of necessary MQs is $O(I_L^2 \cdot |\Sigma| + I_L \cdot |c_0|)$ with $|c_0|$ the length of the longest counterexample c_0 because (a) RED and E contain at most I_L elements each and BLUE contains at most $I_L \cdot |\Sigma|$ elements, so that $O((I_L + I_L \cdot |\Sigma|) \cdot I_L) = O(I_L^2 \cdot |\Sigma|)$ MQs are needed to fill the cells of T, and (b) as MINIMIZE needs $O(|c|)$ MQs for a counterexample c because it may have to check every prefix of c for substitutability by a RED element, processing $O(I_L)$ counterexamples takes another $O(I_L \cdot |c_0|)$ MQs in addition.
  Note that MINIMIZE significantly improves the MQ complexity with respect to the original L^* (see Subsection A.2 above, and also [12]).
- $\langle 0, 1, X_+, \emptyset \rangle$: $|\mathit{Pref}(X_+)|$ ($|\mathit{Suff}(X_+)|$) equals the sum n_+ of the lengths of all strings in X_+ if no two strings in X_+ have a common prefix (suffix) $\neq \varepsilon$. T_{ext} has $|\mathit{Pref}(X_+)|$ rows and $|\mathit{Suff}(X_+)| + O(I_L)$ columns, i.e., $O(n_+ \cdot (n_+ \cdot I_L)) = O(n_+^2 + n_+ \cdot I_L)$ cells to be filled via MQs. Note that as we rely on the representativity of X_+ the final table has $|\mathit{Pref}(X_+)|$ rows as well.
  The length of the longest counterexample c_0 the algorithm can construct is $O(I_L \cdot m)$ where m is the length of the longest string in X_+: We add at most I_L contexts, and every context can be m symbols longer than the previous one,

[3] Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Information and Computation 118(2), pp. 316–326 (1995)

so the length of the longest context is $O(I_L \cdot m)$. Obviously, the longest prefix is $O(m)$ symbols long, so the length of c_0 is $O(I_L \cdot m + m) = O(I_L \cdot (m+1)) = O(I_L \cdot m)$. As MINIMIZE needs $O(|c|)$ MQs to process a counterexample c we have to add this to the overall complexity: $O(n_+^2 + n_+ \cdot I_L + I_L \cdot m)$, but since $m \leq n_+$ this can be resimplified to $O(n_+^2 + 2n_+ \cdot I_L) = O(n_+^2 + n_+ \cdot I_L)$.

- $\langle 0, 0, X_+, X_- \rangle$: Since we exactly emulate RPNI [8, 9] GENMODEL is at least as complex as RPNI. Let n_- be the sum of the lengths of all strings in X_-. If no two strings in X_+ have a common prefix $\neq \varepsilon$ then $PA(X_+)$ has $n_+ - 1$ states and n_+ transitions. Therefore, testing if the automaton we are building from $PA(X_+)$ (wrongly) accepts a string in X_- takes $O(n_+ \cdot n_-)$ steps. We try to merge every state with a RED state ($O(n_+^2)$ steps), checking each merge against X_-, so building the oracle has a complexity of $O(n_+^3 \cdot n_-)$.
  In addition we fill a table of size $O(n_+^3)$ (containing $O(n_+)$ rows and $O(n_+^2)$ columns as we may add a different distinguishing context for every pair of candidates), but $O(n_+^3 \cdot n_- + n_+^3) = O(n_+^3 \cdot (n_- + 1)) = O(n_+^3 \cdot n_-)$.
  Remark: Due to the tree structure of the prefix automaton the merges under consideration can all be computed in a polynomial number of steps (see [9]). Note that in a concrete implementation we would rather arrange the strings in a total (e.g., length-lexical) order to avoid the cumbersome prefix checks in FINDNEXT and MERGENEXT, and we could also store the strings that prevent a merge and pass this information on to MAKEOD.