Meta-algorithm GENMODEL: Generalizing over three learning settings using observation tables

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

Technical report 09-2 University of Trier

Abstract. We present a learning algorithm for regular languages that unifies three existing ones for the settings of minimally adequate teacher learning, learning from membership queries and positive data, and learning from positive and negative data, respectively. We choose these three algorithms as an example to back up the conjecture that the learning process of every algorithm for the class of regular languages founded on the retrieval of equivalence classes under the Myhill-Nerode relation can be mapped to an observation table as introduced by Angluin [1]. Different aspects of this generalization and suggestions for possible (architectural and theoretical) extensions are discussed in the second part of the paper.

Key words: Generalization, regular, exact learning, observation table

1 Introduction

The area of grammatical inference centers on the study of learning algorithms, i.e., algorithms that infer a description (e.g., a grammar or an automaton) of an unknown formal language from given examples and/or other information sources. For this kind of learning processes various conceivable settings have been delineated, and based on those quite an amount of algorithms have been developed. One of the language classes that have been studied most thoroughly with respect to their algorithmical learnability so far is the class of regular languages.

We present an algorithm learning regular languages that incorporates three existing ones for the settings of minimally adequate teacher learning ([1]), learning from membership queries and positive data ([2]), and learning from positive and negative data ([3]). We choose these three algorithms as an example to demonstrate that in the field of grammatical inference there exist a number of learning algorithms for regular languages which can be seen to be founded on a common notion, viz. the retrieval of the appropriate set of equivalence classes under the Myhill-Nerode relation, and more importantly, in doing so we would like to sustain the hypothesis that in all cases this process can be performed using the concept of an observation table as introduced by Angluin [1] in 1987.

Different aspects of this generalization and suggestions for possible (architectural and theoretical) extensions are discussed in the second part of the paper. We conceive this work as a further step to work out as many as possible of the basic features and universal principles that most of the existing learning algorithms seem to have in common, which may hopefully lead to an even more unified base for the theory of grammatical inference in general in the future.

2 Preliminaries

Definition 1. *Let Σ be an alphabet. Then u is a* prefix (suffix) *of w – denoted by $u \preceq w$ ($w \succeq u$) – iff there is v with $w = uv$ ($w = vu$) for $u, v, w \in \Sigma^*$. If $u \preceq w$ then w is an* extension *of u. Let $Pref(X) := \{u \in \Sigma^* | \exists w \in X : u \preceq w\}$ and $Suff(X) := \{u \in \Sigma^* | \exists w \in X : w \succeq u\}$ for $X \subseteq \Sigma^*$.*

The type of learner we consider infers a regular string language L over some fixed alphabet Σ, or rather, the canonical automaton recognizing L, from given samples of L and $\Sigma^* \setminus L$ and answers to certain kinds of queries, and 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 elements from some set S, the columns by elements from some set E.

Definition 2. *A triple $T = (S, E, obs)$ with $S, E \subseteq \Sigma^*$ finite, non-empty for some alphabet Σ is called an* observation table *if $obs : S \times E \longrightarrow \{0, 1, *\}$ 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 an observation table $T = (S, E, obs)$ and $s \in S$, the observed behaviour of s is $row(s) := \{(se, obs(s, e)) | e \in E\}$, and $row(S)$ is defined as $\{row(s) | s \in S\}$. A table or row not containing any pairs (s, e) with $obs(s, e) = *$ is complete.

Definition 3. *Two elements r and s are* obviously different ($r <> s$) *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 two sets RED and BLUE where $uv \in$ RED $\Rightarrow u \in$ RED for $u, v \in \Sigma^*$ (prefix-closedness), and BLUE $:= \{sa \in S \setminus$ RED$| s \in$ RED$, a \in \Sigma\}$, i.e., BLUE contains those one-symbol extensions of RED elements in S that are not in RED. The generalized model of a learner we are about to present moves elements successively from BLUE to RED and for every moved element fills up BLUE with its available one-symbol extensions from a third "supply" set WHITE.

Definition 4. *T is* closed *iff $\neg \exists s \in$ BLUE $: \forall r \in$ RED $: r <> s$. T is* weakly consistent *iff $\forall s_1, s_2 \in$ 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 5 rules out the cases in which hidden differences might prove fatal:

Definition 5. *T is* strongly consistent *iff it is* weakly consistent *and, for $s_1, s_2 \in$ RED or for $s_1 = s_xa$, $s_2 = s_ya \in S$ with $a \in \Sigma$, $s_x, s_y \in$ RED and $\neg(s_x <> s_y)$: If $\neg(s_1 <> s_2)$ then $row(s_1)$ and $row(s_2)$ are complete.*

Definition 6. *A* finite-state automaton *is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ with finite input alphabet Σ, finite non-empty state set Q, start state $q_0 \in Q$, set of accepting 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 *maps at most one state to any pair from* $Q \times \Sigma$ *the automaton is* deterministic *(a* DFA*). If* δ *maps a state to every pair in* $Q \times \Sigma$ *the automaton is* total*, otherwise* partial*.* δ *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).* $\mathcal{A}(w) = 1$ *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* $\neg \exists q \in Q : (q_0, w) \mapsto q \in \delta$*.*

From an observation table $T = (\text{RED} \cup \text{BLUE}, E, obs)$ with $\varepsilon \in E$ we can derive an automaton $\mathcal{A}_T = (Q_T, \Sigma, q_T, \delta_T, F_T)$ 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 the minimal number of states (see [1], Theorem 1). This DFA is the *canonical automaton* $\mathcal{A}_L$ *for* L and is unique up to isomorphism. However, if $\mathcal{A}_L$ is required to be total it contains a "failure state" for all strings that are not a prefix of some string in L (if there are any), which does not have to appear otherwise.

The Myhill-Nerode equivalence relation $\equiv_L$ 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 [4]) states that I_L is finite iff it can be recognized by a finite-state automaton. The total canonical automaton $\mathcal{A}_L$ has exactly I_L states, and each state can be seen to represent an equivalence class under $\equiv_L$. All learning algorithms mentioned in this paper can be conceived to start out with a provisional set of equivalence classes and then to try and converge to the Myhill-Nerode relation by splitting up or merging these classes, according to the obtained information. Note the correspondence between the table $T = (S, E, obs)$ representing $\mathcal{A}_L$ and the equivalence classes of L under $\equiv_L$ (reflected by symbols S and E): S contains strings whose rows are candidates for *states* in $\mathcal{A}_L$, and the elements of E – '*contexts*', as we will call them – can be taken as *experiments* which prove that two strings in S belong to distinct equivalence classes and that their rows should represent two different states of the automaton. RED can be seen as the set of strings whose rows' status as a state in the final automaton is already fixed, and BLUE contains the remaining candidates that are "visible" to the learner so far.

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

Definition 7. *A finite set* X *is* representative *for a regular language* L *with canonical automaton* $\mathcal{A}_L = (Q_L, \Sigma, q_L, \delta_L, F_L)$ *if* $X \subseteq Pref(L)$ *and for each transition* $(q_1, a) \mapsto q_2 \in \delta_L$ *(*$q_1, q_2 \in Q_L$*,* $a \in \Sigma$*) there is* $w \in X$ *and* $u, v \in \Sigma^*$ *such that* $w = uav$ *and* $(q_L, u) \mapsto q_1 \in \delta_L$*.*

Definition 8. *A finite set* X *is* separative *for a regular language* L *with total canonical automaton* $\mathcal{A}_L = (Q_L, \Sigma, q_L, \delta_L, F_L)$ *if for all* $q_1, q_2 \in Q_L$ *there is* $w \in X$ *and* $v \in \Sigma^*$ *such that* $w \succeq v$ *and* $(q_a \in F_L \wedge q_b \in (Q_L \setminus F_L)) \vee (q_b \in F_L \wedge q_a \in (Q_L \setminus F_L))$ *for* $(q_1, v) \mapsto q_a, (q_2, v) \mapsto q_b \in \delta_L$*.*

Intuitively, X is representative for L if in order to parse the elements of X every transition of $\mathcal{A}_L$ has to be used at least once, and X is separative for L if for every pair of different states of $\mathcal{A}_L$ there is some suffix in X proving that these two states do indeed represent two different equivalence classes of L.

3 A meta-algorithm for three learning settings

3.1 Algorithm GENMODEL

The algorithm GENMODEL we will present covers three learning paradigms (at least): Minimally adequate teacher (MAT) learning, learning from membership queries and positive data, and learning from positive and negative data.

In the MAT setting the learner is helped by a teacher able to answer membership (MQs; '$w \in L$?', $w \in \Sigma^*$) and equivalence queries (EQs; 'Is this automaton equivalent to $\mathcal{A}_L$?'). The prototypical algorithm for this setting, L^*, was developed by Angluin [1] in 1987. The main loop of L^* builds a closed, consistent observation table T and asks for a counterexample $C_L(\mathcal{A}_T) \in (L \setminus \mathcal{L}(\mathcal{A}_T)) \cup (\mathcal{L}(\mathcal{A}_T) \setminus L)$ until $\mathcal{A}_T = \mathcal{A}_L$. L^* has been adapted to other structures such as trees (see [5]). We use the fact that every tree algorithm can be applied to strings when conceived as non-branching trees with the last symbol as the root label.

Angluin [6] also introduced an algorithm learning from MQs and positive data, which was adapted to trees in [2] (ALTEX). The ALTEX algorithm builds a (closed) table T from a representative positive sample of L and adds elements to E distinguishing equivalence classes of L ('*separating contexts*') until T is consistent and $\mathcal{A}_T$ deterministic, which is tantamount to $\mathcal{A}_T = \mathcal{A}_L$ (see [2]).

In the third setting the learner receives no external help at all, but is given a finite positive and negative sample from which it is possible to infer L. The algorithm we will consider for this setting (RPNI; see [3]) builds the prefix automaton from the positive sample and then merges its states as long as this is consistent with the negative sample until all states are processed and $\mathcal{A}_T = \mathcal{A}_L$.

The architecture of our meta-algorithm incorporates features of all three, and its syntax is inspired by the MAT algorithm for regular tree languages in [5].

Let the input of the algorithm be a tuple (X_+, X_-, MQ, EQ) consisting of a positive and a negative finite sample of L, and two Boolean values indicating if there is a teacher able to answer MQs and/or EQs including the provision of counterexamples in case of a negative answer to the latter. In its current form, the output of the algorithm is defined for the following constellations:

- $(X_+, X_-, 1, 1)$ (MAT learning) where $X_+ \cup X_- = \emptyset$,
- $(X_+, X_-, 1, 0)$ where $X_- = \emptyset$ and X_+ is representative for L,
- $(X_+, X_-, 0, 0)$ where X_+ is representative and X_- separative for L.

We assume for convenience that the number I of states of $\mathcal{A}_L$ is given, which however represents no (inadmissible) additional help for the learner (see below). Note that for EQ $= 0$ the final automaton is generally partial and contains no failure state so that we set $I = I_L - 1$ in the cases where the total minimal automaton for L would contain one. In all other cases (including $L = \emptyset$) $I = I_L$.

We also assume that the components of the tuple (X_+, X_-, MQ, EQ) are visible to the subprocedures. Let $T = (\text{RED} \cup \text{BLUE}, E, obs)$, WHITE, and $O = (Q_O, \Sigma, q_O, \delta_O, F_O)$ be global variables for the observation table, the set from which BLUE is filled up, and the membership oracle (see below), respectively.

The main body of GENMODEL looks like this:

```
INIT;
while |RED| < I
    if T is not closed CLOSURE
    else SEPCONT;
    UPDATE;
return A_T.
```

The procedure INIT (given below) initializes the table T, WHITE, and the membership oracle O with the given information sources. It recurs to two other procedures: POOL returns the set of all strings that can ever be considered as candidates for representatives of states under the given input, and MQORACLE yields an instance of the best membership oracle the learner can hope for. Of course for MQ = 1 this is trivial, and MQORACLE will return an arbitrary correct total finite-state automaton (a black box) $\mathcal{O}_L$ for the language L that can be used for MQs. However, for MQ = 0 we are forced to exploit other sources of information – GENMODEL builds an imperfect membership oracle based on the given data, which is updated during the process every time the learner gains a new insight. This oracle is initialized with the prefix automaton $PA(X_+)$ built from the positive sample. A prefix automaton is a generally partial DFA:

Definition 9. *Let the prefix automaton for $X \subseteq \Sigma^*$ be defined as $PA(X) = (Q, \Sigma, q_0, \delta, F)$ with $Q = \{\{x\} | x \in Pref(X)\}$, $q_0 = \{\varepsilon\}$, $F = \{\{x\} | x \in X\} \subseteq Q$ and $(q_1, a) \mapsto q_2 \in \delta$ for $a \in \Sigma$ iff $\exists x \in q_1, y \in q_2$ such that $y = xa$.*

The states of $PA(X)$ are labeled by singleton sets of strings. This comes in handy for GENMODEL, as we unite some of these sets during the process, according to the information processed so far. This way, each state will be labeled by the set of all strings ending in it the learner has already found. Let q_w stand short for the state containing $w \in \Sigma^*$ in its label (q_w is unambiguously defined).

```
procedure INIT
    O := MQORACLE;
    P := POOL;
    RED := {ε};
    BLUE := Pref(P) ∩ Σ;
    WHITE := Pref(P) \ (RED ∪ BLUE);
    E = {ε};
    obs := {(ε, ε) ↦ O(ε)}.
procedure MQORACLE
    if MQ = 1 return O_L
    else if X_+ ≠ ∅ return PA(X_+).
procedure POOL
    if X_+ ≠ ∅ return X_+ else return Σ*.
```

We will now discuss the two most essential subroutines of GENMODEL:

```
procedure CLOSURE
    find s ∈ BLUE such that ∀s0 ∈ RED : s <> s0;
    RED := RED ∪ {s};
    BLUE := (BLUE \ {s}) ∪ {s1 ∈ WHITE|∃a ∈ Σ : s1 = sa}.

procedure SEPCONT
    if MQ = 1
       if EQ = 1 c := C_L(A_T)
       else if X+ ≠ ∅ [find s0 ∈ BLUE ∪ WHITE, e0 ∈ E ∪ Suff(X+) such that
                              A_T(s0e0) ≠ O(s0e0) ∧ A_T(s0e0) = * ⇒ O(s0e0) = 1;
                       c := s0e0];
       find s ∈ BLUE, e ∈ Σ+ such that c = se;
       E := E ∪ {e};
    else MERGENEXT;
          find s ∈ BLUE such that ∀s0 ∈ BLUE : |q_s0| = 1 ⇒ s ⪯ s0  (q_s0 ∈ Q_O);
          for s2 ∈ RED do
             c := PREVENTMERGE(q_s2, q_s, O);
             find x ∈ q_s2 ∪ {s}, e ∈ Σ+ such that c = xe;
             E := E ∪ {e};
             if x = s [obs(s2e) := 1; obs(se) := 0]
             else [obs(s2e) := 0; obs(se) := 1];
          if |RED| = I − 1 MERGENEXT.
```

CLOSURE relies on the fact that there is at least one BLUE element obviously different from all RED elements, which is then promoted to RED. BLUE is filled up with all one-symbol extensions of that element from WHITE. CLOSURE is the only way to promote BLUE elements to RED, which entails that all RED elements are pairwise obviously different at any time and that for every BLUE element there is exactly one element matching it in RED. The purpose of CLOSURE can be conceived to fix a string's status as the (only!) "official" representative of a separate equivalence class under $\equiv_L$, and its row must be a separate state of $\mathcal{A}_L$.

SEPCONT relies on the fact that T is closed and strongly consistent but does not represent $\mathcal{A}_L$ so that there is a counterexample c. If the learner has access to a perfect membership oracle c is procured either by asking an EQ (as in L^*), which is answered by 'no' so that c is given by the teacher, or, for EQ = 0, by looking for c in an extension T_{ext} of T where the rows are labeled with $S \cup$ WHITE and the columns with $E \cup Suff(X_+)$, and whose cells can be filled via MQs. We can show that such a counterexample always exists using a correspondence between T_{ext} and the table built by the ALTEX algorithm by which it is guaranteed that T_{ext} contains elements the learner has not seen yet that are either obviously different from all elements in RED or make T_{ext} inconsistent (see A.2 for a proof). GENMODEL finds the prefix s of c in BLUE (see Lemma 1) and $e \in \Sigma^*$ such that $c = se$, and adds e to E. Since c is a counterexample, s is distinguished from its previous match in RED by e, and since the RED elements are pairwise obviously different s is now obviously different from all of them and must thus be promoted to RED by CLOSURE in one of the next loop executions.

Lemma 1. *For MQ = 1: For every counterexample c GENMODEL retrieves there is exactly one prefix of c in* BLUE.

Proof: There is at least one prefix of c in RED (ε is a prefix of every string) and the one-symbol extension of the longest such prefix is in BLUE as in the MAT case BLUE $=$ RED $\cdot\, \Sigma$, and in the case of learning from MQs and positive data c is constructed by appending a string to some element from BLUE $\cup$ WHITE, and RED $\cup$ BLUE $\cup$ WHITE is prefix-closed. By the definition of BLUE $= \{sa \in S \setminus \text{RED} | s \in \text{RED}, a \in \Sigma\}$ there can only be one such extension in BLUE. □

If the learner has no access to a perfect membership oracle (MQ = 0) we cannot make use of a counterexample the way described above because first we cannot complete a table (extended or not) in order to search for hidden differences or inconsistencies, and second, in the present case the only source the learner can draw a counterexample from is the negative sample and there is no guarantee that there is a prefix of that example in BLUE, which were two important conditions we relied on before. Since in this case the table generally contains a lot of $*$-symbols the available candidates must all be checked for inconsistencies with the help of the imperfect oracle O built from the prefix automaton for X_+ and made obviously different "by hand", if necessary. To do so, GENMODEL proceeds as follows: First the procedure MERGENEXT (given below) searches all strings in BLUE which correspond to states in O that can be but have not yet been merged with some other state, and of which there is no prefix in BLUE representing a non-mergeable state. MERGENEXT finds those strings by the cardinality of the state labels they are contained in (labels of states that result from a merge must contain more than one string) and by testing the mergeability of two states via the procedure COMPATIBLE, which checks if a given automaton (here: O with the two states merged) still correctly rejects all elements of X_-. As soon as such a string $s_0 \in$ BLUE is found, the corresponding state q_{s_0} is merged with an arbitrary other state of O it can be merged with (imitating RPNI, see Subsection 3.2). The merge is done by the procedure RECMERGE which calls MERGE and then recursively "repairs" the possible non-determinism introduced by that merge (COMPATIBLE, RECMERGE and MERGE are taken from the description of RPNI in [7] and adapted to automata whose states are labeled by sets of strings). BLUE is then filled up with the successors of s_0. Note that as s_0 stays in BLUE we are violating the definition of BLUE, but this is irrelevant since the algorithm will never consider these particular strings again.

```
procedure MERGENEXT
  while ∃s0 ∈ BLUE : |qs0| = 1 (qs0 ∈ QO) ∧ ¬∃s ∈ BLUE : [s ⪯ s0 ∧ ¬∃sa ∈ RED
              such that COMPATIBLE(RECMERGE(qsa, qs, O)) (qsa, qs ∈ QO)]
       find sb ∈ RED such that COMPATIBLE(RECMERGE(qsb, qs0, O));
       O := RECMERGE(qsb, qs0, O);
       BLUE := BLUE ∪ {sc ∈ WHITE | ∃a ∈ Σ : sc = s0a}.

procedure COMPATIBLE(A)
    if A(w) = 1 for some w ∈ X− return false else return true.

procedure RECMERGE(q1, q2, A) [q1, q2 ∈ QA]
    A := MERGE(q1, q2, A);
```

```
    for a ∈ Σ do
       if |D = {q ∈ Q_A|((q_1 ∪ q_2), a) ↦ q ∈ δ_A}| > 1 find q_a, q_b ∈ D st q_a ≠ q_b;
          A := RECMERGE(q_a, q_b, A);
    return A.

procedure MERGE(q_1, q_2, A) [q_1, q_2 ∈ Q_A]
    q_x := q_1 ∪ q_2;
    Q_A := (Q_A \ {q_1, q_2}) ∪ {q_x};
    if q_2 ∈ F_A  F_A := (F_A \ {q_1, q_2}) ∪ {q_x};
    if q_1 = q_A  q_A := q_x;
    δ_A := (δ_A \ ({(q, a) ↦ q_y|q ∈ Q_A, a ∈ Σ, y ∈ {1, 2}} ∪
                   {(q_y, a) ↦ q|q ∈ Q_A, a ∈ Σ, y ∈ {1, 2}})) ∪
           {(q, a) ↦ q_x|(q, a) ↦ q_y ∈ δ, q ∈ Q_A, a ∈ Σ, y ∈ {1, 2}} ∪
           {(q_x, a) ↦ q|(q_y, a) ↦ q ∈ δ, q ∈ Q_A, a ∈ Σ, y ∈ {1, 2}};
    return A.

procedure PREVENTMERGE(q_1, q_2, A) [q_1, q_2 ∈ Q_A]
    A := RECMERGE(q_1, q_2, A);
    return w ∈ X_− such that A(w) = 1.
```

When all mergeable prefixes have been processed any one-symbol extension $s \in$ BLUE of the longest of these prefixes corresponds to a state that is non-mergeable with any other in O already represented in RED (and should thus represent a separate state of $\mathcal{A}_L$). SEPCONT finds s and, for every $s_2 \in$ RED, retrieves a string from X_- preventing a merge of q_{s_2} with q_s (recall that X_- is separative for L). It uses the procedure PREVENTMERGE, which relies on the fact that the two given states cannot be merged and returns a string $c \in X_-$ that would be accepted by the given automaton if they had been. From this counterexample c the algorithm computes the context e separating s and s_2, where e must be the suffix of c leading from the state resulting from a merge of q_s and q_{s_2} to a final state, adds e to E, and updates the cells (s, e) and (s_2, e) of T "by hand" depending on the gained information from which state e should lead to a final state of O and from which not. Clearly after this loop s is obviously different from all RED elements and will be promoted in the next main loop execution by CLOSURE. When there is but one state missing from $\mathcal{A}_T$ we know that a string corresponding to this state will be promoted to RED in the next loop execution and thus MERGENEXT is called again to merge all states in O corresponding to the remaining elements in BLUE∪WHITE with states corresponding to elements in RED to make sure that O evolves into the perfect membership oracle $\mathcal{A}_L$ so that T is completed by the next call of UPDATE.

```
procedure UPDATE
    WHITE := WHITE \ BLUE;
    obs := {(s, e) ↦ O(se)|s ∈ RED ∪ BLUE, e ∈ E};
    if EQ = 0 ∧ |RED| = I − 1 ∧ T is not closed BLUE := BLUE ∪ WHITE.
```

UPDATE clears the candidates that have been moved to BLUE out of WHITE and fills in more cells of the table with the help of the oracle O in case the sets S or E have been modified. Usually the change in the output of a function is

assumed to be implied as soon as its domain has been changed, but we prefer to build *obs* explicitly as a set of mappings (like δ in Definition 6) in order to state more clearly where *obs* gets its information, namely from O, and also to allow the procedure SEPCONT to modify individual values of *obs* more easily when learning from positive and negative data. For EQ = 0, if there is one more state missing from $\mathcal{A}_L$ and the needed separating context is already in E so that T is not closed we have to move the remaining candidates to BLUE to ensure that no transition is lacking (recall that X_+ is representative for L). Again this violation of the definition of BLUE is not fatal for the correctness of the algorithm.

Theorem 1. *GENMODEL terminates after at most* $2I-1$ *loop executions and the final table correctly represents* $\mathcal{A}_L$*.*

Proof: In each main loop execution either CLOSURE promotes a BLUE element to RED or SEPCONT adds a context to E separating a BLUE element from its single match in RED so that it is promoted to RED by CLOSURE in one of the following loop executions. Consequently, the algorithm must terminate after $2I-1$ loop executions at most, and every string in RED represents a different equivalence class under $\equiv_L$. Due to the pairwise obvious difference between the RED elements and the completeness of the table T is always strongly consistent and $\mathcal{A}_T$ deterministic. Since BLUE = RED$\cdot\Sigma$ for EQ = 1 and X_+ is representative for L and RED $\cup$ BLUE $= Pref(X_+)$ in the other cases when $|\text{RED}| = I$ no transition is missing and $\mathcal{A}_T$ corresponds to the minimal DFA $\mathcal{A}_L$. □

Note that knowing I is not essential for the termination of GENMODEL as there exists at least one additional criterion for each of the three underlying learning settings: We could use an EQ for MAT learning, check if T_{ext} is consistent when learning from MQs and positive data, and stop when there are no more states of O to process when learning from positive and negative data.

3.2 The relationship of GENMODEL to L^*, ALTEX, and RPNI

GENMODEL is based on the same principle as the algorithm L^* by Angluin [1] insofar as GENMODEL, too, uses an observation table labeled by a partitioned set of candidates for representatives of states and a set of contexts, and starts out with a single equivalence class $[\varepsilon]$ which is split up successively according to the received information (sometimes this is referred to as a *specializing* algorithm). It was one of the purposes of this paper to show that this particular way of proceeding can be applied to a number of other learning settings as well.

Between L^* and GENMODEL there is only one major difference, which is the use of counterexamples: Whereas L^* adds the example and all its prefixes to RED GENMODEL adds a single suffix of it to E. However, both methods correctly lead to at least one more distinct row in RED (see A.1 for a proof).

Adapting ALTEX to the 'L^*' pattern is relatively easy since it uses a table labeled by candidates for representatives of states and contexts as well. ALTEX starts with a table whose rows are labeled by $Pref(X_+)$ and the columns by $Suff(X_+)$ (with X_+ a positive representative sample of L), and adds separating

contexts it computes from inconsistencies in the table until the latter is consistent, which corresponds to the determinism of the derived automaton and (as X_+ is representative) to its equivalence with $\mathcal{A}_L$. Thus in general ALTEX starts with several distinct equivalence classes but still has to split them up in order to arrive at the relation $\equiv_L$. Note that although ALTEX has the "head start" of having all the candidates and some contexts separating them at its disposal from the beginning this disadvantage is evened out for GENMODEL when learning from MQs and positive data by giving the algorithm access to a table enriched with the complete information available at every call of SEPCONT.

Adapting RPNI to the 'L^*' pattern requires more work. The most striking difference is that RPNI does not use a table but starts out with a special automaton – the prefix automaton for the given positive sample X_+ – and then merges its states as far as this is consistent with the given negative sample X_- (and since the states can be seen as provisional equivalence classes whose number is successively reduced RPNI may be referred to as a *generalizing* algorithm). However, we have shown that the actions of RPNI can be mapped to an 'L^*'-style table as well: GENMODEL constructs a table T starting with a single row and column labeled by ε and simultaneously builds an automaton O from $PA(X_+)$ in the manner of RPNI. Each state of $PA(X_+)$ corresponds to an element of $Pref(X_+)$, the set of candidates for RED. These states are then tested for mergeability successively (respecting the prefix order on the corresponding strings): In RPNI, if a hitherto unprocessed state q_s is mergeable with any of the already established ones those two states are merged, which is imitated in GENMODEL by leaving s in BLUE, but if it is not RPNI marks it down as a separate state of $\mathcal{A}_L$ and GENMODEL does the same by adding a context separating s from r to E for every $r \in$ RED, which it computes from some string in the separative sample X_- preventing the merge of q_s and q_r in O, causing s to be promoted to RED by CLOSURE in the next loop execution. Due to this parallelism RPNI and GENMODEL end up distinguishing the same number of states. Since we assume that X_+ is representative and no transitions of $PA(X_+)$ are erased all necessary transitions are present in the automaton built by RPNI, and since finally RED $\cup$ BLUE $= Pref(X_+)$ in T the same is true for GENMODEL. Furthermore, as O is used as a membership oracle and evolves into $\mathcal{A}_L$ as well T is completed by the last call of UPDATE: Completeness of BLUE together with the fact that RED is pairwise obviously different is sufficient to make T strongly consistent and $\mathcal{A}_T$ deterministic so that $\mathcal{A}_T$ represents $\mathcal{A}_L$ when $|\text{RED}| = I$.

Another aim in designing GENMODEL was to optimize certain features of the other algorithms by trying to slim down the table wherever possible: The fact that in GENMODEL (as opposed to L^* and ALTEX) the RED elements are all pairwise obviously different entails that in RED no equivalence class under $\equiv_L$ is represented twice, and the fact that for MQ $= 1$ we add a single separating context only in order to promote a BLUE element to RED entails that E contains no "useless" contexts (as opposed to ALTEX, for example). This stays true for MQ $= 0$, although we may need one separating context for every RED element in order to promote one BLUE element to RED in the worst case.

4 Generalizing even further – reflections on other constellations of information sources

Although the output of the version of GENMODEL presented above is well defined for MAT learning, learning from MQs and positive data, and learning from positive and negative data (with the data fulfilling certain properties) only, we believe the algorithm to be potentially more general than that, i.e., that with minor modifications it can be made to correctly infer a regular language (in the sense of Theorem 1 and all its preconditions) when given other suitable combinations of information sources as well. Of course, one way to verify this is to check all possible constellations systematically. However, it has already been shown by various authors ([8–10]) that regular languages cannot be learned from one of the named information sources alone, i.e., one kind of queries or sample only. In this section, let us briefly reflect which combinations of two of those could be of interest for an extension of our meta-algorithm.

Arguably, MQs are a very fundamental means of gaining information since they enable a learner to fill in the cells of an observation table completely. It seems that one class of suitable combinations joins MQs and any method of retrieving a counterexample, from which a separating context can be computed as described in 3.1. Three of those methods that are already known to us are:

- EQs (MAT learning). Counterexamples are given by the teacher.
- A representative sample of L. Counterexamples can be computed from inconsistencies in a table extended by the prefixes and suffixes of the sample.
- A separative sample of L, either positive or negative. Like a teacher, such a sample is just another source of counterexamples.

However, from our (successful) adaptation of RPNI to the 'L^*' pattern we deduce that it is possible to do without MQs if we combine a positive representative sample of L with one of the other two methods of obtaining a counterexample given above, i.e., either a separative sample of L or EQs. In that case we would use counterexamples not only in order to compute the necessary separating contexts for strings corresponding to non-mergeable states but to check the mergeability of states in the auxiliary automaton that is built simultaneously with the table as well. Note that when choosing the option of checking the mergeability of two states via a query what we would ask is not an equivalence but rather a consistency query, in the sense that the automaton $\mathcal{A}_{TM}$ resulting from the merge may not contradict $\mathcal{A}_L$ for any string s with $\mathcal{A}_{TM}(s) \neq *$.

An aside about equivalence queries: Since in GENMODEL I is given, EQs are not needed as termination test (see 3.1). As long as $|\text{RED}| < I$ (i.e., any time the main loop is entered and queries are used at all) an EQ must be answered negatively and a counterexample given so that in the context of this work we prefer to reconstrue this kind of queries as mere requests for a counterexample.

Apparently two information sources fulfilling certain essential properties seem to suffice. For MQ = 1 the availability of more than one of the other sources represents no essential gain in terms of information since for example the possibility of asking EQs would make an additional separative or representative

sample of L as a source of counterexamples superfluous, and the possibility of asking MQs would in turn make the sample superfluous altogether. However, as a representative sample can serve the further purpose of supplying the set of potential candidates for RED, a constellation of three information sources that may make some sense is the combination of MQs with a representative sample of L and one of the other two ways to obtain counterexamples because then the set of candidates is finite and the derived automaton partial in general, if that is desired (e.g., to achieve more uniformity with other constellations the algorithm is defined for). For MQ $= 0$, combining a representative sample with a separative sample and EQs would be futile since in our algorithm both separative samples and EQs serve the same purpose of providing counterexamples only.

If we want GENMODEL to be defined for any conceivable input we also have to decide on its output in the case of insufficient information. If only one kind of queries is allowed and no data given a standard solution is to return a default automaton such as $\mathcal{A}_{\Sigma^*}$ or $\mathcal{A}_\emptyset$. However, if a sample is given the returned automaton should at least be consistent with the data. For a positive sample X_+ the most evident option would be $PA(X_+)$, and for a negative sample X_- we could return $\mathcal{A}_\emptyset$ or $\mathcal{A}_{\Sigma^* \setminus X_-}$. A third solution involves another version of finite-state automata sometimes found in the literature (see for example [7]), which adds explicit rejecting states and classifies the membership status of elements that are neither assigned a final nor a rejecting state as unknown – an output automaton of that definition would explicitly reject the sample but refrain from making a statement about any other elements, which is perhaps more satisfying.

For a further exploration concerning the combination of samples and queries see for example [11]. Another interesting paper is [12] in which it is shown that the number of EQs in the MAT setting can be reduced but that in return the number of MQs has to be increased by a certain factor, and vice versa. The general efficiency of GENMODEL remains to be studied formally, but since our construction either adopts or even improves the principal mechanisms of some of the prototypical algorithms for the respective settings the assumption that the efficiency of these algorithms is at least preserved seems relatively safe.

5 Conclusion

In this paper we have presented the learning algorithm GENMODEL, which is defined for MAT learning, learning from MQs and positive data, and learning from positive and negative data, and incorporates features of each of three prototypical algorithms (known as L^*, ALTEX, and RPNI) for these settings in its architecture. We have shown that the algorithm terminates and is able to correctly infer a regular language L in less than $2I$ executions of its main loop, where I is the number of states of the canonical automaton $\mathcal{A}_L$ for L, and we have pointed out some of its intrinsic and architectural properties in comparison to the ones of the three algorithms it unifies in order to make their relationship more explicit. Several other suitable combinations of information sources which could be integrated into the algorithm were surveyed and discussed.

One of the most important motivations for the creation of GENMODEL was an endeavour to find out more about the theoretical foundations of algorithmical language learning. We intended to show that quite a number of the settings and corresponding algorithms that have hitherto been described and developed in the area of grammatical inference can be seen to be founded on a common conception (which seems to be closely linked to the Myhill-Nerode equivalence relation) and that accordingly many of the existing algorithms can be remodeled into a common form (involving an observation table or a similar concept), and we did so by presenting a concrete algorithm of which we hope that it has the potential of incorporating as many different learning settings as possible. Our main goal in developing such a meta-algorithm was a more thorough understanding and thus an easier formulation of universal principles and conditions for formal learnability. On the other hand, it might also be possible to use it as a template from which individual learning algorithms for hitherto unstudied learning settings or language classes may be instantiated immediately.

Our suggestions for future work also concentrate mainly on the further exploration of GENMODEL's generalization potential and the theoretical implications for algorithmical learnability. There are other conceivable kinds of information sources that remain to be studied with respect to their applicability in the present framework, such as subset queries ('Does my result describe a subset of L?', where in case of a negative answer a counterexample is given; see [9, 11]) or so-called correction queries (a superordinate concept uniting several types of queries where in case of a negative answer the teacher returns some kind of corrected version of the queried item; see [13] for references), or oracles that deviate from a perfect neutral randomized one either by impeding the learner (giving no or even a wrong answer in a certain percentage of the cases) or by being more "helpful" in some way (for example by returning counterexamples of minimal length, or examples that distinguish a maximal number of elements, etc., thus possibly making the learner proceed more efficiently). Other directions for future research include the investigation of the conditions we may be able to impose on the information sources when GENMODEL is learning certain subclasses of the regular languages (such as reversible or locally testable languages; see [14, 15]), or a survey of other kinds of mathematical objects GENMODEL may be adapted to, such as trees (which should be easy since the incorporated algorithms have been adapted to trees already – see [5, 2, 16] – but there are some complexity issues to be solved), graphs, pictures, infinite strings, and many more.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
2. Besombes, J., Marion, J.Y.: Learning tree languages from positive examples and membership queries. In: ALT 2003. Volume 3244 of LNCS. Springer (2004) 440–453
3. 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

4. Hopcroft, J.E., Ullmann, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Longman (1990)
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. de la Higuera, C.: Grammatical inference. Unpublished manuscript.
8. Gold, E.M.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
9. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
10. Angluin, D.: Negative results for equivalence queries. Machine Learning **5** (1990) 121–150
11. Jain, S., Kinber, E.: Learning languages from positive data and a finite number of queries. Information and Computation **204**(1) (2006) 123–175
12. Balcázar, J.L., Díaz, J., Gavaldà, R., Watanabe, O.: A note on the query complexity of learning dfa. In: ALT 1992, Springer (1993) 53–62
13. Tîrnăucă, C.: A note on the relationship between different types of correction queries. In: ICGI 2008, Springer (2008) 213–223
14. Angluin, D.: Inference of reversible languages. JACM **29**(3) (1982) 741–765
15. Head, T., Kobayashi, S., Yokomori, T.: Locality, reversibility, and beyond: Learning languages from positive data. In: ALT 1998, Springer (1998) 191–204
16. Oncina, J., Garcia, P.: Inference of recognizable tree sets. In: Research Report DSIC - II/47/93. (1993)
17. 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 two ways to use a counterexample for MQ = 1

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

(a) Add c and all its prefixes to RED*.*
(b) Find $s \in$ BLUE, $e \in \Sigma^+$ with $c = se$ and add e to E.

Proof. (a) Either such a row is created directly if E already contains an appropriate separating context, or the table becomes inconsistent. To see the latter, assume that neither c nor any of its prefixes is obviously different from every RED element. Note that in GENMODEL this can occur in the MAT case only, since in the case of learning from MQs and positive data c is constructed taking some string s_0 obviously different from all RED elements as a prefix. We may therefore also assume that $\mathcal{A}_T(c) \in \{0, 1\}$ because in the MAT case $\mathcal{A}_T$ is total. As the automaton derived from the new table including c and its prefixes can assign a different state to c than $\mathcal{A}_T$ (c is a counterexample) although no new distinct row and thus no new state has been created this automaton is non-deterministic, which equals the inconsistency of the table. A consistency check (which would be needed when using this first method) would correct that by adding a separating

context distinguishing two RED elements that have not been obviously different before in the following loop execution, thus creating another distinct row.
(b) By Lemma 1 there is always a prefix s of c in BLUE. After adding $e \in \Sigma^+$ with $c = se$ to E the table is not closed anymore: Since $O(c) \neq \mathcal{A}_T(c)$ and since with this method the RED elements are pairwise obviously different because no elements are added except by closure s is distinguished by e from its single RED match and promoted to RED by closure in one of the next loop executions. □

Remark: With the second method suffix-closedness for E is given up, 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 L^* and in ALTEX is just a by-product of the two algorithms' ways to generate separating contexts.

The equivalence of adding the prefixes of a counterexample to the candidates and adding the suffixes to the contexts was first mentioned in a footnote in [17].

A.2 T_{ext} contains a counterexample for $\mathcal{A}_T$

Lemma 3. *Let $T = (S = \text{RED} \cup \text{BLUE}, E, obs)$. As long as $|\text{RED}| < I$ a counterexample for $\mathcal{A}_T$ can always be found in $T_{ext} = (S \cup \text{WHITE}, E \cup Suff(X_+), obs_{ext})$.*

Proof. Either T_{ext} contains I distinct rows and represents (the failure state-free) $\mathcal{A}_L$ so that as T cannot contain I distinct rows yet there must be at least one element $s_0 \in \text{WHITE}$ obviously different from all RED elements in T_{ext}. We find a counterexample for $\mathcal{A}_T$ by distinguishing two cases (note that there can be two reasons for a string to be rejected by an automaton – either it ends in a non-final state or there are transitions missing that would be needed to parse it):

- $\mathcal{A}_T(s_0) \in \{0, 1\}$: Since s_0 is obviously different from all RED elements there must be a context $e_0 \in E \cup Suff(X_+)$ separating s_0 and the representative of the state assigned to s_0 by $\mathcal{A}_T$, so s_0e_0 is a counterexample for $\mathcal{A}_T$.
- $\mathcal{A}_T(s_0) = *$ (and consequently $\mathcal{A}_T(s_0e_0) = *$): Since $s_0 \in \text{Pref}(X_+)$ we can find a context $e_0 \in Suff(X_+)$ such that $s_0e_0 \in X_+$ (and consequently $O(s_0e_0) = 1$), so s_0e_0 is a counterexample for $\mathcal{A}_T$.

If T_{ext} is not the final table we exploit the fact that $T_{ext} = (S \cup \text{WHITE} = \text{Pref}(X_+), E \cup Suff(X_+), obs_{ext})$ corresponds to T enriched with the information initially available to the ALTEX algorithm and that consequently the same lemmata that apply to the ALTEX table can be applied to T_{ext} as well: By contraposition of Lemma 4 in [2] (stating that as soon as the table is consistent it represents $\mathcal{A}_L$) T_{ext} must contain an inconsistency involving $s_1, s_2 \in \text{Pref}(X_+)$ and $a \in \Sigma$ such that $\neg(s_1 <> s_2)$ but $s_1a <> s_2a$. As T is closed, either $s_1a \in \text{WHITE}$ or $s_2a \in \text{WHITE}$, and we can take that element as s_0. Since $s_1a <> s_2a$ there must be some $e_0 \in E \cup Suff(X_+)$ separating s_1a from s_2a in T_{ext}, 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. □
It is easy to check that in all cases the two conditions defining a counterexample stated in the fifth line of the subprocedure SEPCONT are fulfilled.