2-6-1995

An Efficient Interactive Algorithm for Regular Language Learning

Rajesh G. Parekh
Iowa State University

Vasant Honavar
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/cs_techreports

Part of the Artificial Intelligence and Robotics Commons, and the Theory and Algorithms Commons

Recommended Citation
http://lib.dr.iastate.edu/cs_techreports/144

This Article is brought to you for free and open access by the Computer Science at Iowa State University Digital Repository. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
An Efficient Interactive Algorithm for Regular Language Learning

TR95-02
Rajesh Parekh & Vasant Honavar

February 6, 1995

Iowa State University of Science and Technology
Department of Computer Science
226 Atanasoff
Ames, IA 50011
AN EFFICIENT INTERACTIVE ALGORITHM FOR
REGULAR LANGUAGE LEARNING

Rajesh Parekh & Vasant Honavar
Artificial Intelligence Research Group
CS TR #95-02
February 1995
An Efficient Interactive Algorithm for Regular Language Learning

Rajesh Parekh & Vasant Honavar
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall,
Iowa State University,
Ames, IA 50011. U.S.A.
parekh@cs.iastate.edu & honavar@cs.iastate.edu

Abstract

This paper presents an efficient algorithm for learning regular grammars. A knowledgeable teacher provides the learner with a structurally complete set of strings that belong to L(G), the language corresponding to an unknown regular grammar G. This structurally complete set of positive examples implicitly specifies a lattice which represents the hypothesis space of candidate grammars that contains the unknown grammar G. The learner searches this lattice by posing queries about the membership in G of particular strings and uses the teacher’s response to eliminate parts of the hypothesis space. The learner maintains at all time, a compact representation of the lattice in the form of two sets S and G that correspond (respectively) to the set of most specific and most general grammars consistent with the set of positive examples provided and the queries answered by the teacher up to that point in time. The correctness of the algorithm is established by proving that at least one element of the lattice, G*, that is equivalent to the unknown grammar G is contained in the hypothesis space represented by S and G and that the algorithm terminates upon identifying such an element.
1 Introduction

Grammar inference is an important machine learning problem with many applications of practical significance in pattern recognition and language acquisition [Fu, 82, Honavar, 94]. Regular grammars, although limited in their descriptive power (as compared to context-free and context-sensitive grammars), represent a particularly useful class of formal grammars for practical applications for several reasons including: Every finite grammar is regular; Context-free grammars can be closely approximated by regular grammars [Fu, 82]; Regular grammars are probably the most tractable of all formal grammars for machine learning [Natarajan, 92]. This paper develops an algorithm for learning regular grammars within an active learning framework in which in addition to the sample strings provided by the teacher, the learner uses the teacher’s responses to membership queries to efficiently search the space of candidate grammars. The learner’s task is to correctly infer an unknown regular grammar using the data provided by the teacher in the form of positive samples and answers to queries.

The paper is organized as follows: Section 1 introduces the necessary terminology and defines the version of the grammar inference problem studied in this paper; Section 2 describes the grammar inference algorithm; Section 3 details the proof of correctness; Section 4 ends with summary, discussion of related work, and directions for future research on this topic.

1.1 Regular Grammars and Finite State Automata

A grammar is a finite set of rewrite (production) rules of the form \( \alpha \rightarrow \beta \) where \( \alpha \) and \( \beta \) are sequences of symbols. These rewrite rules are applied recursively to generate strings. The set of all strings generated by a grammar \( G \) is referred to as its language \( L(G) \). Different classes of formal languages are obtained by placing particular restrictions on the form of the production rules. Regular grammars have rules of the form \( A \rightarrow aB \) or \( A \rightarrow b \) where \( A \) and \( B \) are called non-terminals and \( a \) and \( b \) are called terminals. Strings generated by the grammar can contain only terminals. Finite State Automata (FSA) are recognizers for regular grammars in that they accept only strings that belong to the language of the grammar. A deterministic FSA \( A \) is a quintuple \( A = (Q, \delta, \Sigma, q_0, F) \) where, \( Q \) is a finite set of states, \( \Sigma \) is the finite set of input symbols called the alphabet, \( F \subseteq Q \) is the set of accepting states,
\(q_0 \in Q\) is the start state and \(\delta\) is the transition function \(Q \times \Sigma \rightarrow Q\) giving the next state of the automaton upon reading a particular symbol. FSA are traditionally represented using state transition diagrams. Fig. 1 shows a sample FSA. Here \(Q = \{Q_0, Q_1\}\), \(\delta = \{\delta(Q_0, a) = Q_1, \delta(Q_0, b) = Q_0, \delta(Q_1, b) = Q_0\}\), \(\Sigma = \{a, b\}\), \(q_0 = Q_0\) and \(F = \{Q_0\}\). Since \(q_0 \subseteq F\), this automaton accepts the null string \((\lambda)\). \(L(G) = \{\lambda, b, ab, bb, abb, \ldots\}\). The equivalent regular grammar is represented by the following production rules:

\[
S \rightarrow \lambda, \ S \rightarrow bS, \ S \rightarrow aA, \ A \rightarrow bS.
\]

\(S\) stands for the start symbol.

![Finite State Automaton](image)

**Figure 1: Finite State Automaton**

### 1.2 The Grammar Inference Problem

The grammar inference problem [Biermann & Feldman, 72, Parekh & Honavar, 95] is defined as follows: For an unknown grammar \(G\), given a finite set of positive examples \(S^+\) that belong to \(L(G)\), and possibly a finite set of negative examples \(S^-\), infer a grammar \(G^*\) equivalent to \(G\). Many variants of the inference problems can be defined by placing different restrictions on the sample sets \(S^+\) and \(S^-\) and the interaction of the learner with the teacher or the environment. We present a method for inference of a FSA corresponding to \(G^*\) which is equivalent to the unknown grammar \(G\). In this paper, \(S^+\) is restricted to be a *structurally complete* set in order to facilitate a theoretical analysis of the problem domain. A structurally complete set covers each production rule of \(G\) at least once. Equivalently if \(M_G\) is the FSA corresponding to the grammar \(G\), then each transition of \(M_G\) must be covered at least once by some example string \(x \in S^+\). (Work on extension of the algorithm that relax the structural completeness assumption is in progress). Additional ex-
amples, both positive and negative, are provided by the teacher in the form of answers to queries posed by the learner.

2 Grammar Inference Algorithm

The teacher provides a set of positive samples $S^+$ which implicitly defines a lattice $\Omega$ of candidate grammars or the initial hypothesis space that is guaranteed to contain the unknown grammar [Pao & Carr, 78, Parekh & Honavar, 93]. The learner generates strings and queries the teacher about their membership in the unknown grammar $G$. At all times, the learner maintains two sets of lattice elements — $S$ and $G$ — which correspond respectively to the most specific and most general grammars consistent with the data that has been gathered by the learner so far. Thus, $\Theta = [S, G]$ provides a compact representation of the hypothesis space at all time. The teacher’s response to a membership query results in pruning of the hypothesis space while ensuring that the target grammar is not eliminated in the process. (See section 3 for a proof of this claim). The interaction between the learner and the teacher proceeds until a single grammar $G^+$ that is equivalent to the unknown target grammar $G$ is left in the hypothesis space.

2.1 Lattice of Grammars Specified by $S^+$

This section explains the construction of the lattice $\Omega$ given the set $S^+$. First, an automaton called the canonical automaton, $M_{S^+}$, that accepts every string in $S^+$ and no other is constructed. This canonical automaton provides a path from the start state to an accepting state for each string in the set $S^+$. For example, suppose the grammar $G$ of the automaton $M_G$ in Fig. 1 is to be inferred. Suppose the teacher provides a structurally complete set of strings $S^+ = \{\lambda, abb\}$ from $L(G)$. The corresponding canonical automaton $M_{S^+}$ is shown in Fig. 2. The lattice $\Omega$ of candidate grammars can be explicitly constructed by systematically merging the states of the canonical automaton $M_{S^+}$ to form partitions. Each such partition $P$ of states of $M_{S^+}$ yields an element of $\Omega$. The language corresponding to the automaton defined by $P$ is a superset of the language of $M_{S^+}$. Thus, successive state mergings yield more and more general languages. (See below for details). The lattice constructed from the canonical automaton (Fig. 2) is depicted in Fig. 3. A canonical
automaton with \( m \) states yields an initial hypothesis space that contains:
\[
E_m = \sum_{j=0}^{m-1} \binom{m-1}{j} E_j
\]
grammars where \( E_0 = 1 \). Therefore, explicit representation of the hypothesis space is generally not feasible in practice. The proposed grammar inference algorithm therefore represents the hypothesis space at all time implicitly using \( \Theta = [S, G] \) as outlined earlier and explained in Section 2.2. Each element of the lattice (i.e., a partition \( P \) of the states of

\[
M_{S'}
\]
corresponds to a FSA \( M \) constructed as follows: The states of \( M \) correspond to the cells of the partition \( P \). Each cell contains one or more states of \( M_{S'} \) that are grouped together in the partition \( P \). For the automaton \( M \), the start state is the cell which contains the start state \( s_0 \) of \( M_{S'} \), the accepting
states are the cells that contain one or more accepting states of $M_{S^+}$, the alphabet is the same as that of $M_{S^+}$, and the transition function for $M$, $\delta_M$, is defined on the basis of the transitions within the canonical automaton. If several states of $M_{S^+}$ are merged together in a cell $\alpha$ in $P$, then the transitions into each of those states become transitions into the state represented by $\alpha$ in $M$ and the transitions out of each of these states become the transitions out of the state represented by $\alpha$ in $M$. Transitions between two states that end up merged in cell $\alpha$ form self loops. The FSA corresponding to the partition $P_2$ (of Fig. 3) is shown in Fig. 4. The lattice $\Omega$ of grammars (or equivalently automata, or partitions) is ordered by the grammar cover relation. If each cell of a partition $P_i$ at one level of the lattice is contained in some cell of a partition $P_j$ in the level above, we say that $P_j$ covers $P_i$ ($P_i \subseteq P_j$). Let $M_i$ and $M_j$ be the FSA and $L_i$ and $L_j$ be the regular languages that correspond to the partitions $P_i$ and $P_j$ respectively. Clearly, if $P_i \subseteq P_j$, $L_j \supseteq L_i$. This is indicated in Fig. 3 by an arrow from $P_i$ to $P_j$. If there is an arrow from $P_i$ to $P_j$, we say that $P_i$ is an immediate lower-bound of $P_j$ and analogously, $P_j$ is an immediate upper-bound of $P_i$. Grammar covers is a transitive property. Thus, if $P_i \subseteq P_j$ and $P_j \subseteq P_k$ then $P_i \subseteq P_k$. Then we say that $P_i$ is more specific than or equal to (MSE) $P_k$ (which is conversely more general than or equal to (MGE) $P_i$). The MSE (MGE) test can be performed in polynomial time by just examining the cells of the partitions under consideration. The learner exploits this property while pruning the hypothesis space as explained below.

![Diagram](image)

**Figure 4:** FSA corresponding to the partition $P_2$
As already noted, the hypothesis space of candidate grammars is represented implicitly by $\Theta = [S, G]$. $S$ is the set of maximally specific elements of the hypothesis space that are consistent with all the data gathered by the learner at any time. Similarly, $G$ is the set of maximally general elements of the hypothesis space. Initially, $S = \{M_{S^+}\}$ and $G$ contains the most general element of the lattice $\Omega$ (i.e., the partition with all states of $M_{S^+}$ merged together in a single cell). During the search the learner constructs fully specified, deterministic finite state automata ($M_i = \{S, \delta_i, \Sigma, s_0, A\}$ and $M_j = \{T, \delta_i, \Sigma, t_0, B\}$) for partitions $P_i \in S$ and $P_j \in G$. $M_i$ and $M_j$ are compared for equivalence. If they are not equivalent then there exists at least one input string $y$ such that $\delta_i(s_0, y) \in A$ but $\delta_i(t_0, y) \notin B$ or vice-versa (in which case the roles of $M_i$ and $M_j$ are simply reversed). This string $y$ belongs to the difference machine $M_i - M_j = \{W, \delta_w, \Sigma, w_0, C\}$ where, $W = S \times T$, $w_0 = (s_0, t_0)$, $\delta_w((s,t), \sigma) = (\delta_i(s, \sigma), \delta_i(t, \sigma))$ for all $\sigma \in \Sigma$ and $C = \{(s,t) | s \in A \text{ and } t \in T - B\}$ [Harrison, 65]. A query of the form “$y \in G$?” is posed to the teacher. Based on the teacher’s response $\Theta$ is pruned and elements of $S$ and $G$ become progressively more general and more specific respectively.

It is observed that in general each partition has more than one upper and lower bounds. Since the algorithm generates partitions dynamically it is likely that partitions implicitly eliminated earlier (by elimination of their upper or lower bounds — see below for details) are generated again. Considerable efficiency in terms of a reduction in the number of queries is achieved by maintaining two lists of partitions $S^-$ and $G^+$. $S^-$ stores the partitions that were eliminated from $S$ as a result of the corresponding FSA accepting a negative example (i.e., a query that got a negative response from the teacher). Analogously $G^+$ stores the partitions that were eliminated from $G$ as a consequence of the corresponding FSA failing to accept a positive example.

The rules for modification of $\Theta$ based on the nature of the query and the teacher’s response are as follows:

1. $y \in L(M_i - M_j)$ and $y \in L(G)$ (i.e., $M_j$ rejects a positive example).

   (a) Eliminate $P_j$ from $G$ and place it on list $G^+$.
(b) Minimally generalize each partition $P_k$ in $S$ where $P_k \subseteq P_j$ but retain only those generalizations that are not MSE $P_j$.

(c) Remove any partition in $S$ that is MGE some partition in $S^–$.

(d) Eliminate any partition in $S$ that is MGE some partition in $S$.

(e) Eliminate any partition in $S$ that is not MSE some partition in $G$ and any partition in $G$ that is not MGE some partition in $S$.

2. $y \in L(M_j - M_i)$ and $y \notin L(G)$ (i.e., $M_i$ accepts a negative example).

(a) Eliminate $P_i$ from $S$ and place it on list $S^–$.

(b) Minimally specialize each partition $P_i$ in $G$ where $P_i \subseteq P_i$ but retain only those specializations that are not MGE $P_i$.

(c) Remove any partition in $G$ that is MSE some partition in $G^+$.

(d) Eliminate any partition in $G$ that is MSE some partition in $G$.

(e) Eliminate any partition in $S$ that is not MSE some partition in $G$ and any partition in $G$ that is not MGE some partition in $S$.

3. $y \in L(M_j - M_i)$ and $y \in L(G)$ (i.e., $M_i$ rejects a positive example).

(a) Minimally generalize the partition $P_i$.

(b) Remove any partition in $S$ that is MGE some partition in $S^–$.

(c) Eliminate any partition in $S$ that is MGE some partition in $S$.

(d) Eliminate any partition in $S$ that is not MSE some partition in $G$ and any partition in $G$ that is not MGE some partition in $S$.

4. $y \in L(M_j - M_i)$ and $y \notin L(G)$ (i.e., $M_j$ accepts a negative example).

(a) Minimally specialize the partition $P_j$.

(b) Remove any partition in $G$ that is MSE some partition in $G^+$.

(c) Eliminate any partition in $G$ that is MSE some partition in $G$.

(d) Eliminate any partition in $S$ that is not MSE some partition in $G$ and any partition in $G$ that is not MGE some partition in $S$.

5. $M_i \equiv M_j$. No action is taken in this case and another pair of automata is compared for equivalence.
Elimination of candidate solutions proceeds as explained above until all the partitions in $\mathcal{S}$ and $\mathcal{G}$ correspond to equivalent automata. As shown in Section 3, each such automaton is guaranteed to be equivalent to the unknown target automaton corresponding to the target language $L(G)$. The following example illustrates the working of the algorithm given $S^+ = \{\lambda, abb\}$ which defines the lattice shown in Fig. 3. (The choice of elements of $\mathcal{S}$ and $\mathcal{G}$ to be compared at each step is arbitrary).

**Example**

1. $\mathcal{S} = \{P_0\}; \mathcal{G} = \{P_{14}\}$
   Compare $M_0$ with $M_{14}$
   $M_0 \neq M_{14}$; Query: $y = a \in L(G)$? (NO)
   $\mathcal{S} = \{P_0\}; \mathcal{G} = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}\}$

2. $\mathcal{S} = \{P_0\}; \mathcal{G} = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}\}$
   Compare $M_0$ with $M_{13}$
   $M_0 \neq M_{13}$; Query: $y = abb \in L(G)$? (YES)
   $\mathcal{S} = \{P_1, P_2, P_3, P_4, P_5, P_6\}; \mathcal{G} = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}\}$

3. $\mathcal{S} = \{P_1, P_2, P_3, P_4, P_5, P_6\}; \mathcal{G} = \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}\}$
   Compare $M_1$ with $M_{13}$
   $M_1 \neq M_{13}$; Query: $y = a \in L(G)$? (NO)
   $\mathcal{S}^+ = \{P_1\}; \mathcal{S} = \{P_2, P_3, P_4, P_5, P_6\}; \mathcal{G} = \{P_9, P_{10}, P_{12}, P_{13}\}$

4. $\mathcal{S} = \{P_2, P_3, P_4, P_5, P_6\}; \mathcal{G} = \{P_9, P_{10}, P_{12}, P_{13}\}$
   Compare $M_2$ with $M_{13}$
   $M_2 \neq M_{13}$; Query: $y = b \in L(G)$? (YES)
   $\mathcal{G}^+ = \{P_{13}\}, \mathcal{S} = \{P_2, P_5, P_6\}; \mathcal{G} = \{P_9, P_{10}, P_{12}\}$

The algorithm proceeds in this manner and after a few additional queries terminates with the solution $\mathcal{S} = \{P_9\}$ and $\mathcal{G} = \{P_9\}$.
3 Proof of Correctness

The proof of correctness of our algorithm directly follows from the two theorems stated below:

**Theorem 1.** The target grammar $G$ lies in the lattice $\Omega$ (see section 2 for details) defined by a structurally complete set of strings $S^+$ from $G$.

**Proof:** The proof of theorem 1 is originally due to Pao and Carr [78]. It is included below for completeness.

Let $Z_i = \{S, \delta_i, \Sigma, s_0, A\}$ and $Z_j = \{T, \delta_i, \Sigma, t_0, B\}$ be two deterministic finite state machines. The state $s_i$ of $Z_i$ is said to be similar to the state $t_j$ of $Z_j$ (denoted as $s_i \sim t_j$) if one of the following two conditions is fulfilled:

1. $s_i = s_0$ and $t_j = t_0$ i.e. the start states of the two finite state machines.
2. There exists a state $s_x$ of $Z_i$ and $t_y$ of $Z_j$ such that:
   
   (a) $s_x \sim t_y$, and
   
   (b) $\delta_s(s_x, \gamma) = s_i$ and $\delta_t(t_y, \gamma) = t_j$.

**Lemma 1:** Two deterministic finite state machines are behaviorally equivalent if the following two conditions hold:

1. $s_i \sim t_j$ for $s_i \in S$ and $t_j \in T$ is a one-to-one and onto function, and
2. If $s_i \sim t_j$ then $s_i \in A$ if $t_j \in B$.

**Proof:**

Let $\psi$ be a one-to-one and onto function mapping the set $S$ to the set $T$ such that $\psi(s_i) = t_j$ if $s_i \sim t_j$. If we can prove the following then by condition 2 above we can conclude that the two finite state machines are behaviorally equivalent.

$$x \in \text{path}(s_0, s_i) \iff x \in \text{path}(\psi(s_0), \psi(s_i)) \text{ for all } s_i \text{ of } Z_i$$

(1)

We prove $x \in \text{path}(s_0, s_i) \implies x \in \text{path}(\psi(s_0), \psi(s_i))$ for all $s_i$ of $Z_i$ by mathematical induction on the length of string $x$ (i.e. $|x|$).
**Base Case:**
If $|x| = 1$ i.e. $x$ is an edge from state $s_0$ to a state $s_i$ of $Z_i$. We know that $s_0 \sim t_0$ and there is a state $t_j$ such that $s_i \sim t_j$. Therefore, $x$ is an edge from $t_0$ to state $t_j$ of $Z_j$. Since, $t_0 = \psi(s_0)$ and $t_j = \psi(s_i)$ we have proved the base case.

**Induction Hypothesis:**
Assume that the lemma is true for $|x| \leq n$.
If $|x| = n$, then $x \in \text{path}(s_0, s_{pm}) \iff x \in \text{path}(\psi(s_0), \psi(s_{pm}))$ for a state $s_{pm}$ of $Z_i$.

**Induction Proof:**
Now we show that the lemma is true for $|x| = n + 1$. Let $x = a_1a_2a_3 \ldots a_n$. Since $x$ is a path from $s_0$ to $s_i$ then there exist states $s_{p1}, s_{p2}, \ldots, s_{pn}$ in $S$ such that $a_1$ is an edge from $s_0$ to a state $s_{p1}$, $a_j$ is an edge from $s_{pj-1}$ to a state $s_{pj}$ for $2 \leq j \leq n$, and $a_{n+1}$ is an edge from $s_{pn}$ to a state $s_i$. By the induction hypothesis, we know that $a_1a_2a_3 \ldots a_n$ is a path from $s_0$ to $s_{pm}$. Thus, $a_1a_2a_3 \ldots a_n$ is a path from $\psi(s_0)$ to $\psi(s_{pm})$. Now $s_0 \sim t_0$ and there is only one state $t_{pn} \sim s_{pm}$. From this we conclude that $a_1a_2a_3 \ldots a_n$ is a path from $t_0$ to $t_{pn}$. Since there is only one state $t_j$ of $Z_j$ which is similar to $s_i$ of $Z_i$ we conclude that $a_{n+1}$ is an edge from $t_{pn}$ to $t_j$. Thus, $x$ is a path from $t_0$ to $t_j$ i.e. $x$ is a path from $\psi(s_0)$ to $\psi(s_i)$. The converse can be proved by similar argument.

Given $S^+ \subseteq L(G)$ we constructed the canonical automaton $M_{S^+}$ as described in Section 2. We establish the following two lemmas.

**Lemma 2:** For each $s_i$ of $M_{S^+}$ there is exactly one state $t_j$ of $M_G$ such that $s_i \sim t_j$.

**Proof:**
For any string $x = a_1a_2a_3 \ldots a_n$ of $M_{S^+}$ we know that there exist states $s_{p1}$, $s_{p2}, \ldots, s_{pm}$ in $S$ such that $a_1$ is an edge from $s_0$ to a state $s_{p1}$, $a_j$ is an edge from $s_{pj-1}$ to a state $s_{pj}$ for $2 \leq j \leq n$. Since $x$ is contained in $L(G)$ there are states $t_{r_1}, t_{r_2}, \ldots, t_{rn}$ in $M_G$ such that $a_1$ is an edge from $t_0$ to a state $t_{r_1}$, $a_j$ is an edge from $t_{rj-1}$ to a state $t_{rj}$ for $2 \leq j \leq n$. By definition of similarity of states we know that $s_0 \sim t_0$ and from above we conclude that $s_{pj} \sim t_{rj}$ for $1 \leq j \leq n$. Since the languages generated by finite state machines are deterministic (unambiguous), there is one and only one way to generate a
string a given string \( x \in L(G) \). Thus, we have proved that for each \( s_i \) of \( M_{S^+} \) there is exactly one \( t_j \) of \( M_G \) such that \( s_i \sim t_j \).

**Lemma 3:** For each \( t_j \) of \( M_G \) there is at least one state \( s_i \) of \( M_{S^+} \) such that \( s_i \sim t_j \).

**Proof:**

Since the construction of \( S^+ \) requires that each edge of the machine \( M_G \) must be covered at least once, we know that for any edge of \( M_G \) say ‘a’ from \( t_{r_k} \) to \( t_{r_l} \) there exist states \( s_{p_k} \) and \( s_{p_l} \) of \( M_{S^+} \) with the edge ‘a’ connecting them. Thus, for each \( t_j \) of \( M_G \) there is at least one state \( s_i \) of \( M_{S^+} \) such that \( s_i \sim t_j \).

**Construction of the solution automaton \( M_G^* \).**

Merge each state \( s_{p_1}, s_{p_2}, \ldots, s_{p_k} \) of the automaton \( M_{S^+} \) such that each \( s_{p_n} \sim t_j \) where \( n = 1, 2, \ldots, k \) into a single state \( s_i \). Thus, \( s_i \in M_{S^+} \sim t_j \in M_G \).

The new machine thus obtained is called \( M_G^* \). From lemma 2 we know that each \( s_{p_m} \) (\( n = 1, 2, \ldots \)) is merged into exactly one \( s_i \). From lemma 3 it is clear that for each \( t_j \) of \( M_G \) there is exactly one \( s_i \) of \( M_G^* \). Thus, there is a one-to-one and onto correspondence between \( s_i \) and \( t_j \). A state \( s_F \) of \( M_G^* \) is a final/accepting state iff a state \( s_{p_G} \) of \( M_{S^+} \) is merged into \( s_F \) and \( s_{p_G} \) is an accepting state for \( M_{S^+} \). Thus, using lemmas 1-3, and the construction of the solution automaton it follows that \( M_G^* \equiv M_G \).

**Theorem 2:** Let \( P_{M_G^*} \) be the partition corresponding to the target automaton \( M_G^* \). The following invariance condition holds at all times during the execution of the algorithm:

\[ \exists P_x \in \mathcal{G} \text{ and } \exists P_y \in \mathcal{S} \text{ such that } P_y \subseteq P_{M_G^*} \subseteq P_x \]

**Proof:** By induction

**Base Case:** Initially, \( \mathcal{S} \) contains only one partition — that corresponding to the canonical automaton \( M_{S^+} \) and \( \mathcal{G} \) contains only one partition — that corresponding to the most general automaton in the lattice. Therefore, the hypothesis space \( \Theta = [\mathcal{S}, \mathcal{G}] \) implicitly includes the entire lattice \( \Omega \). Theorem 1 guarantees that a partition corresponding to the target grammar lies within \( \Omega \), and hence in the hypothesis space \( \Theta \). Clearly, the invariance condition holds if we set \( P_y \) to be the \( M_{S^+} \) and \( P_x \) to be the most general automaton in \( \Omega \).
**Induction Hypothesis**: Assume that the invariance condition holds at some time during the execution of the algorithm (just prior to the processing of a query \( q \)).

**Induction Proof**: We prove that the invariance condition continues to hold after processing a query \( q \). Consider a query \( q \) comparing \( M_i \in \mathcal{S} \) with \( M_j \in \mathcal{G} \).

1. \( y \in L(\text{M}_i;\neg\text{M}_j) \) is the query and \( M_j \) does not accept a positive example.

   (a) \( P_j \) is eliminated from \( \mathcal{G} \). Then it must be the case that \( P_{M_G^*} \not\subseteq P_j \) (otherwise \( M_G^* \) fails to accept a positive example which would be a contradiction). Thus a partition other than \( P_j \) in \( \mathcal{G} \) plays the role of \( P_y \) in satisfying the invariance condition.

   (b) Each partition \( P_k \) in \( \mathcal{S} \) such that \( P_k \subseteq P_j \) minimally generalized but only those partitions that are not MSE \( P_j \) are retained. No \( P_k \) removed could be \( P_{M_{G^*}} \) or else \( M_G^* \) would not accept the positive example. If a partition \( P_k \) removed was serving as \( P_y \), we show that one of the partitions placed on \( \mathcal{G} \) (by minimally generalizing \( P_k \)) takes over as \( P_y \) thereby preserving the invariance condition. In the previous step we established that \( P_{M_G^*} \not\subseteq P_j \). This means that \( P_{M_G^*} \) has at least two states \( (s_1 \) and \( s_2 \)) in the same cell that are in different cells in \( P_j \) (by the grammar covers property). Also in the given scenario, \( P_k \subseteq P_{M_G^*} \) and \( P_k \subseteq P_j \). This is possible only if \( s_1 \) and \( s_2 \) are in different cells in \( P_k \) as well. Since an immediate upper bound of a partition is obtained by fusing two cells, one upper bound of \( P_k \) will have the two cells containing states \( s_1 \) and \( s_2 \) merged together. This particular partition is no longer a lower bound of \( P_j \) but is still a lower bound of \( P_{M_G^*} \). This partition is designated as the new \( P_y \) and we now have \( P_z \subseteq P_{M_{G^*}} \subseteq P_y \).

   (c) Partitions in \( \mathcal{S} \) that are MSE some partition in \( \mathcal{S}^- \) are removed. None of these partitions could correspond to \( P_y \) or else since \( P_{M_{G^*}} \) is MGE \( P_y \), \( M_G^* \) would accept a negative example.

   (d) Any partition in \( \mathcal{S} \) that is MGE some other partition in \( \mathcal{S} \) is eliminated. If the designated \( P_y \) is eliminated because a partition \( P_m \) was less general than \( P_y \), \( P_m \) takes over the role of \( P_y \).
(e) Any partition in $\mathcal{S}$ that is not MSE some partition in $\mathcal{G}$ and correspondingly any partition in $\mathcal{G}$ that is not MGE some partition in $\mathcal{S}$ is eliminated. Since $P_y \subseteq P_z$ neither partition is eliminated.

In each step above the invariance is preserved.

2. $y \in L (M_j - M_i)$ is the query and $M_i$ accepts a negative example. This case is analogous to the previous one.

3. $y \in L (M_j - M_i)$ is the query and $M_i$ does not accept a positive example.

   (a) $P_i$ is generalized minimally. If $P_i$ was the designated $P_y$ then one of its upper bounds just placed on $\mathcal{S}$ will take over the role of $P_y$.

   (b) Any partition in $\mathcal{S}$ that is MGE some partition in $\mathcal{S}^-$ is removed. None of these partitions could be $P_y$ or else $P_G$ would also accept a negative example.

   (c) Any partition in $\mathcal{S}$ that is MGE some other partition in $\mathcal{S}$ is eliminated. If designated $P_y$ is eliminated because a partition $P_m$ was less general than $P_y$, $P_m$ takes over the role of $P_y$.

   (d) Any partition in $\mathcal{S}$ that is not MSE some partition in $\mathcal{G}$ and correspondingly any partition in $\mathcal{G}$ that is not MGE some partition in $\mathcal{S}$ is eliminated. Since $P_y \subseteq P_z$ neither partition is eliminated.

   In each step above the invariance is preserved.

4. $y \in L (M_j - M_i)$ is the query and $M_j$ accepts a negative example. This case is analogous to the previous one.

5. If $M_j \equiv M_i$ neither $\mathcal{S}$ nor $\mathcal{G}$ are altered and so the invariance condition is not violated.

Successive elimination of lattice elements by querying results in the scenario where all partitions left in $\Theta = [\mathcal{S}, \mathcal{G}]$ are equivalent to each other, and by the invariance condition, equivalent to the target automaton.
4 Summary and Discussion

Grammar inference is an important problem in machine learning with many applications of great practical significance including speech recognition, computational biology (e.g., DNA sequence recognition), language acquisition, syntactic pattern classification, and so on. Several versions of grammar inference problem have been studied extensively in the literature. Gold [78] has shown that uniquely identifying the target automaton from a set of labeled examples is an \textit{NP-complete} problem. In related work, Angluin [87] has proposed a polynomial time algorithm (\textit{L}), which allows the learner to infer the target grammar by posing both membership and automata equivalence queries. Porat and Feldman [91] make use of a lexicographically ordered sample whereas Rivest and Schapire [91] have suggested a mechanism based on \textit{homing sequences}. Vanlehn and Ball [87] have proposed a version-space approach to learning context-free grammars that returns a set of grammars consistent with the given sample set. Giles et al [91] use recurrent neural networks to learn FSA using positive and negative samples. The interested reader is referred to [Parekh & Honavar, 95] for a more detailed discussion of grammar inference techniques for machine learning.

In this paper, we have presented a provably correct mechanism for inference of regular grammars given a structurally complete set of samples and a teacher that responds reliably to membership queries. Our work is most closely related to the regular grammar inference algorithm proposed by Pao and Carr [78]. We have borrowed their idea of mapping the structurally complete set of examples to an ordered lattice. The main contributions of our work that is described in this paper are summarized below:

- The size of the hypothesis space \( \Theta \) defined by a set of structurally complete samples \( S^+ \) is too large to be represented explicitly or to be searched exhaustively. The algorithm proposed in this paper, unlike Pao’s algorithm, uses a compact representation of the hypothesis space in terms of \( S \) and \( G \).

- The proposed algorithm uses an efficient bidirectional search strategy inspired by Mitchell’s version space algorithm. This enables the learner to eliminate potentially large parts of the hypothesis space based on a single informative query.
The proof of correctness of the proposed algorithm given in this paper has identified several interesting properties of the hypothesis space $\Theta$ which point to promising directions for future research.

One promising direction that is also of considerable practical significance is currently under investigation. This involves a relaxation or complete elimination of the structural completeness assumption and replacing it with \textit{structural completeness in the limit}. That is, positive samples are provided one at a time by the teacher (interspersed with responses to queries generated by the learner) so that as the sample size grows, it eventually would include $S^+$. This would provide an efficient incremental, interactive inference algorithm for regular languages. Other promising directions for further exploration include: mechanisms for generating in some sense, most informative queries so as to speed up learning; extension of the proposed approach to regular tree grammars and attributed grammars; derivation of useful bounds on the number of queries needed to converge to the target grammar; empirical estimates of the expected case time and space complexity of the proposed grammar algorithm and its extensions; and statistically well-founded criteria for using a hypothesis space $\Theta$ to classify test strings before the learner has converged on a single target grammar.

\section*{Acknowledgements}

Vasant Honavar would like to acknowledge the support of the National Science Foundation (through the grant IRI-9409580) and the Iowa State University College of Liberal Arts and Sciences during this research.
References


