

# An Incremental Interactive Algorithm for Regular Grammar Inference<sup>\*</sup>

Rajesh Parekh & Vasant Honavar

Artificial Intelligence Research Group  
Department of Computer Science  
Iowa State University, Ames, IA 50011, U.S.A.  
{parekh|honavar}@cs.iastate.edu

**Abstract.** We present provably correct interactive algorithms for learning *regular grammars* from positive examples and membership queries. A *structurally complete* set of strings from a language  $L(G)$  corresponding to a target regular grammar  $G$  implicitly specifies a lattice of finite state automata (FSA) which contains a FSA  $M_G$  corresponding to  $G$ . The lattice is compactly represented as a version-space and  $M_G$  is identified by searching the version-space using membership queries. We explore the problem of regular grammar inference in a setting where positive examples are provided intermittently. We provide an incremental version of the algorithm along with a set of sufficient conditions for its convergence.

## 1 Introduction

*Regular Grammar Inference* [3, 5, 9, 12] is an important machine learning problem with applications in pattern recognition and language acquisition. It is defined as the process of learning an unknown regular grammar ( $G$ ) given a finite set of positive examples  $S^+$ , possibly a finite, non-empty set of negative examples  $S^-$ , and possibly a knowledgeable teacher who can answer queries posed by the learner. We present an algorithm for *regular grammar inference* in an *active learning* framework wherein the learner's task is to infer the unknown grammar using the teacher supplied positive examples and answers to *membership* queries.

A *Regular Grammar* is a finite set of rewrite (production) 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*. These rules are applied recursively to generate *strings* (containing terminal symbols only). The *language*  $L(G)$  is the set of all strings generated by the grammar  $G$ . *Finite State Automata* (FSA) are recognizers for regular grammars. A *deterministic* FSA (DFSA),  $A$ , 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  $Q \times \Sigma \rightarrow Q$  that gives the next state of the automaton

---

<sup>\*</sup> Vasant Honavar is grateful to National Science Foundation (grant NSF IRI-9409580) and the John Deere Foundation for supporting his research. The authors would like to thank Professor Giora Slutzki for several helpful discussions related to this work.

upon reading a particular symbol. Fig. 1 shows the state transition diagram for a sample FSA. A *non-deterministic* FSA without  $\epsilon$ -transitions (NFSA) is defined exactly like the DFSA except that the transition function  $\delta: Q \times \Sigma \rightarrow 2^Q$  where  $2^Q$  is the power set of  $Q$ . Intuitively, there might be multiple transitions out of a state on any letter of the alphabet. DFSA and NFSA are equivalent in expressive power. Two FSA  $M_1$  and  $M_2$  are equivalent iff  $L(M_1) = L(M_2)$ . A *subautomaton* ( $A_s$ ) of a FSA  $A$  is a FSA and is defined by  $A_s = (Q_s, \delta_s, \Sigma, q_0, F_s)$  where  $Q_s \subseteq Q$ ,  $F_s \subseteq F$ , and  $\delta_s \subseteq \delta$  in the sense that  $\forall p, q \in Q_s$  and  $a \in \Sigma$ , if  $\delta_s(p, a) = q$  then  $\delta(p, a) = q$ . Clearly,  $L(A_s) \subseteq L(A)$ . For a detailed treatment of this subject see [10].

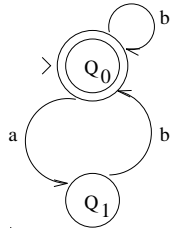


Fig. 1. Finite State Automaton

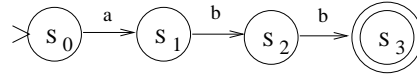


Fig. 2. MCA -  $M_{S^+}$

This paper is organized as follows: Sect. 2 describes a non-incremental version of the grammar inference algorithm and proves its correctness. Sect. 3 explains the incremental version of the algorithm. Finally, Sect. 4 provides a summary, discussion of related work, and directions for future research.

## 2 Inference given a Structurally Complete $S^+$

Initially, we present a non-incremental algorithm for inference of a FSA (not necessarily deterministic) that is equivalent to a FSA corresponding to the target regular grammar ( $G$ ). A knowledgeable teacher provides a set  $S^+$  of positive examples (i.e.,  $S^+ \subseteq L(G)$ ) which is restricted to be a *structurally complete* set in order to facilitate theoretical analysis. A structurally complete set of strings covers each production rule of  $G$  at least once. Equivalently, if  $M_G$  is a FSA corresponding to the grammar  $G$ , then each transition of  $M_G$  must be covered by at least one string in  $S^+$  and for each accepting state of  $M_G$  there must be at least one string in  $S^+$  which terminates in that accepting state. In what follows, we use the terms target grammar and target FSA interchangeably.

The teacher provides a structurally complete set  $S^+$  which implicitly defines a lattice (or version space)  $\Omega$  of candidate FSA or the initial hypothesis space that is guaranteed to contain a FSA ( $M_G$ ) corresponding to the target grammar ( $G$ ) [14, 15, 4]. At all times, the learner maintains two sets of lattice elements —  $\mathcal{S}$  and  $\mathcal{G}$  — which correspond respectively to the most specific and most general FSA consistent with the data (sample strings, queries) processed so far. Thus,  $\Theta = [\mathcal{S}, \mathcal{G}]$  provides a compact representation of  $\Omega$ . The learner generates strings

and queries the teacher about their membership in  $G$ . The teacher's response to a query results in pruning of the hypothesis space while ensuring that  $M_G$  is not eliminated in the process. The interaction between the learner and the teacher proceeds until a single FSA (or a set of equivalent FSA) equivalent to  $M_G$  remains in  $\Omega$ .

## 2.1 Lattice of Grammars Specified by $S^+$

Given  $S^+$ , a FSA called the *maximal canonical automaton* (MCA),  $M_{S^+}$ , that accepts every string in  $S^+$  and no other is constructed. This MCA 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 FSA  $M_G$  in Fig. 1 is to be inferred and the teacher provides  $S^+ = \{abb\}$ . The corresponding MCA  $M_{S^+}$  is shown in Fig. 2. The lattice  $\Omega$  of candidate grammars can be explicitly constructed by systematically merging the states of the MCA  $M_{S^+}$  to form partitions. Each partition yields an element of  $\Omega$ . The language corresponding to the FSA defined by a partition is a superset of  $L(M_{S^+})$ . Thus, successive state mergings yield progressively more general languages as explained below. The lattice constructed from the MCA (Fig. 2) is depicted in Fig. 3. A MCA with  $m$  states yields an initial hypothesis space that contains:

$$E_m = \sum_{j=0}^{m-1} \binom{m-1}{j} E_j \text{ grammars where } E_0 = 1.$$

Therefore, explicit representation of  $\Omega$  is generally impractical. The proposed algorithm represents  $\Omega$  implicitly using  $\Theta = [\mathcal{S}, \mathcal{G}]$ .

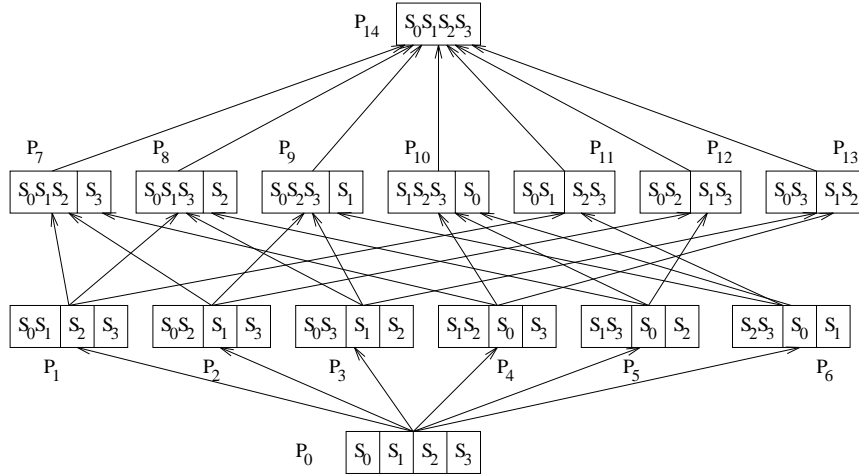


Fig. 3. Lattice -  $\Omega$

Each element of the lattice (i.e., a partition of the set of states of  $M_{S^+}$ ) represents a FSA  $M$ . The states of  $M$  correspond to the blocks of the partition. For the FSA  $M$ , the start state is the block which contains the start state of

$M_{S^+}$ , the accepting states are the blocks which 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  $M_{S^+}$ . If several states of  $M_{S^+}$  are merged together in a block  $c$  in a partition, then the transitions into (out of) each of those states become transitions into (out of) the state represented by  $c$  in  $M$ . Transitions between two states that get merged in block  $c$  form self loops on the state resulting from the merger. The FSA  $M_2$  corresponding to the partition  $P_2$  (of Fig. 3) is shown in Fig. 4.

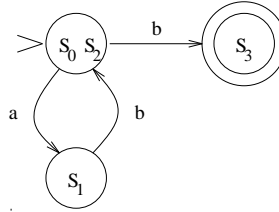


Fig. 4. FSA corresponding to the partition  $P_2$

The lattice  $\Omega$  of grammars (or equivalently FSA, or partitions) is ordered by the *grammar cover* relation. This property is exploited during the search for  $M_G$ . If each block of a partition  $P_i$  at one level of the lattice is contained in some block of a partition  $P_j$  in the level above, we say that  $P_j$  *covers*  $P_i$  ( $P_i \preceq P_j$ ). 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  $P_i \preceq P_j$ ,  $L_i \subseteq L_j$ . 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 cover is a transitive property. Thus, if  $P_i \preceq P_j$  and  $P_j \preceq P_k$  then  $P_i \preceq P_k$  and 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 efficiently by just examining the partitions under consideration.

## 2.2 Query-Aided Bi-Directional Search of the Lattice

$\Omega$  is implicitly represented by  $\Theta = [\mathcal{S}, \mathcal{G}]$  where  $\mathcal{S}$  is the set of maximally *specific* elements and  $\mathcal{G}$  is the set of maximally *general* elements of  $\Omega$  that are consistent with all the data gathered by the learner at any time. Initially,  $\mathcal{S} = \{P_0\}$ , the partition corresponding to the MCA  $M_{S^+}$  and  $\mathcal{G} = \{P_{E_m-1}\}$ , the partition corresponding to the most general element of  $\Omega$  i.e., the one with all states of the MCA merged together in a single block. Note that  $E_m$  is the total number of partitions in the lattice  $\Omega$ . The learner constructs FSA  $M_i = \{S, \delta_s, \Sigma, s_0, A\}$  and  $M_j = \{T, \delta_t, \Sigma, t_0, B\}$  corresponding to partitions  $P_i \in \mathcal{S}$  and  $P_j \in \mathcal{G}$  respectively.  $M_i$  and  $M_j$  are compared for equivalence. If they are not equivalent then there exists a string  $y$  such that  $\delta_s(s_0, y) \in A$  but  $\delta_t(t_0, y) \notin B$  or vice-versa (in which case the roles of  $M_i$  and  $M_j$  are simply reversed). The shortest such string  $y$

belonging to the language of the difference machine  $M_i - M_j = \{W, \delta_w, \Sigma, w_0, C\}$  where,  $W = S \times T$ ,  $w_0 = (s_0, t_0)$ ,  $\delta_w((s,t), \sigma) = (\delta_s(s, \sigma), \delta_t(t, \sigma))$  for all  $\sigma \in \Sigma$  and  $C = \{(s,t) \mid s \in A \text{ and } t \in T - B\}$  [7] forms the query “ $y \in L(G)$ ”? that is posed to the teacher. Based on the teacher’s response  $\Theta$  is pruned and elements of  $\mathcal{S}$  and  $\mathcal{G}$  become progressively *more general* and *more specific* respectively. Since the lattice  $\Omega$  defines a partial order and the MSE (MGE) test can be performed efficiently on the elements of the lattice, the *version-space* algorithm [13] can be adapted for candidate elimination.

**Algorithm:**

1. Set  $\mathcal{S} = \{P_0\}$  and  $\mathcal{G} = \{P_{E_m - 1}\}$ .
2. While there exists an element  $P_i \in \mathcal{S}$  and  $P_j \in \mathcal{G}$  such that the corresponding FSA  $M_i \not\equiv M_j$  pick the shortest string  $y \in L(M_i - M_j)$  or  $L(M_j - M_i)$  and pose the query  $y \in L(G)$ ? Based on the teacher’s response modify  $\Theta$  as described below (see candidate elimination).
3. Return the FSA corresponding to the partition to which the search of the lattice ( $\Omega$ ) converges.

**Candidate Elimination:**

1. If  $y$  is a positive example
  - (a) Remove any  $P_k \in \mathcal{G}$  such that the FSA  $M_k$  rejects  $y$ .
  - (b) Minimally generalize any  $P_l \in \mathcal{S}$  if  $M_l$  does not accept  $y$ . Retain only those partitions in  $\mathcal{S}$  that are MSE some partition in  $\mathcal{G}$ .
  - (c) Remove any element from  $\mathcal{S}$  that is MGE some other element in  $\mathcal{S}$ .
2. If  $y$  is a negative example
  - (a) Remove any  $P_k \in \mathcal{S}$  such that the FSA  $M_k$  accepts  $y$ .
  - (b) Minimally specialize any  $P_l \in \mathcal{G}$  if  $M_l$  accepts  $y$ . Retain only those partitions in  $\mathcal{G}$  that are MGE some partition in  $\mathcal{S}$ .
  - (c) Remove any element from  $\mathcal{G}$  that is MSE some other element in  $\mathcal{G}$ .

*Minimally generalizing* a  $P_l \in \mathcal{S}$  when  $M_l$  does not accept a positive example  $y$  involves replacing  $P_l$  by its immediate upper bounds. Any of these upper bounds not accepting  $y$  are minimally generalized and so on till all the generalizations thus obtained accept  $y$ . *Minimal specialization* is defined analogously for elements of  $\mathcal{G}$ . The choice of elements of  $\mathcal{S}$  and  $\mathcal{G}$  to be compared at each step is arbitrary. Alternatively, this may be guided by a suitable heuristic. The following example illustrates the working of the algorithm given  $S^+ = \{abb\}$  which defines the lattice shown in Fig. 3. The algorithm terminates with a solution  $M_9$  which is the target FSA. Note that at step 3, if  $M_3$  is compared with  $M_{13}$  (instead of  $M_9$ ) convergence would need an additional query.

**Example**

Step	$\Theta$	$M_i \equiv M_j ?$	Query $y$	Modified $\Theta$
1	$\mathcal{S} = \{P_0\}; \mathcal{G} = \{P_{14}\}$	$M_0 \not\equiv M_{14}$	$\lambda \in L(G)$	$\mathcal{S} = \{P_3\}; \mathcal{G} = \{P_{14}\}$
2	$\mathcal{S} = \{P_3\}; \mathcal{G} = \{P_{14}\}$	$M_3 \not\equiv M_{14}$	$a \notin L(G)$	$\mathcal{S} = \{P_3\}; \mathcal{G} = \{P_9, P_{13}\}$
3	$\mathcal{S} = \{P_3\}; \mathcal{G} = \{P_9, P_{13}\}$	$M_3 \not\equiv M_9$	$b \in L(G)$	$\mathcal{S} = \{P_9\}; \mathcal{G} = \{P_9\}$

### 2.3 Proof of Correctness

The correctness of the algorithm follows from the following theorems.

**Theorem 1**<sup>1</sup>: A FSA  $M_G$  corresponding to the target grammar  $G$  lies in the lattice  $\Omega$  defined by a structurally complete set of strings for  $M_G$ .

**Theorem 2**: Let  $P_{M_G}$  be the partition corresponding to the target FSA  $M_G$ . The following invariance condition holds at all times during the execution of the algorithm:  $\exists P_z \in \mathcal{G}$  and  $\exists P_y \in \mathcal{S}$  such that  $P_y \preceq P_{M_G} \preceq P_z$ .

**Proof**: By induction.

**Base Case**: Initially,  $\mathcal{S} = \{P_0\}$  and  $\mathcal{G} = \{P_{E_{m-1}}\}$ . Therefore, the hypothesis space  $\Theta = [\mathcal{S}, \mathcal{G}]$  implicitly includes the entire lattice  $\Omega$ . Theorem 1 guarantees that  $P_{M_G}$  lies within  $\Omega$ , and hence in the hypothesis space  $\Theta$ . Clearly, the invariance condition holds if we set  $P_y$  to  $P_0$  and  $P_z$  to  $P_{E_{m-1}}$ .

**Induction Hypothesis**: Assume that the invariance condition holds at some time during the execution of the algorithm (just before processing a query).

**Induction Proof**: We prove that the invariance condition continues to hold after processing the query. If the query string  $y$  is a positive example:

1. Any  $P_k \in \mathcal{G}$  such that the FSA  $M_k$  rejects  $y$  is removed. Clearly, no such  $P_k$  could be  $P_z$  or else  $M_G$  would also reject  $y$ .
2. A partition  $P_l \in \mathcal{S}$  is minimally generalized if  $M_l$  does not accept  $y$ . Consider that the partition  $P_l$  generalized is  $P_y$ . Since  $P_y \preceq P_{M_G}$  there is a sequence of one or more generalizations leading from  $P_y$  to  $P_{M_G}$  such that at least one of these generalizations accepts the positive string  $y$ . This generalization (which could be  $P_{M_G}$ ) becomes the new  $P_y$ . Only those partitions in  $\mathcal{S}$  that are MSE some partition in  $\mathcal{G}$  are retained. Since the new  $P_y \preceq P_{M_G} \preceq P_z$  it is clear that the new  $P_y$  will not be eliminated from  $\mathcal{S}$ .
3. Finally, any partition in  $\mathcal{S}$  that is MGE some other partition in  $\mathcal{S}$  is removed. If the designated  $P_y$  is eliminated the partition in  $\mathcal{S}$  to which it is MGE takes over as the new  $P_y$ .

At the end of each step above we see that the invariance is preserved. A symmetric argument can be presented for the case when  $y$  is a negative example.

## 3 An Incremental Algorithm for Grammar Inference

Often in practical inductive learning scenarios the entire training data is not available to the learner at the start. This motivates the need for incremental learning algorithms that enable the learner to develop suitable hypotheses based on the available data and, when presented with additional training data, update the hypotheses appropriately without having to reprocess the previous data.

---

<sup>1</sup> The proof of theorem 1 is originally due to Pao and Carr [14] and has been reworked in [15]. It was also independently proven by Miclet (see [4]).

Suppose a set  $S_0^+$  of positive examples that is not necessarily structurally complete is provided at the start. We ask whether it is possible, using  $S_0^+$ , to infer a hypothesis that has a well-defined relationship with the target and whether it is possible to incrementally update the hypothesis as and when additional positive examples are provided until eventually the hypothesis converges to the target when a structurally complete set has been processed. The answer to this question is affirmative provided certain conditions are satisfied: First, the teacher must provide positive examples in non-decreasing order by length (see [16] for an explanation). Second, the teacher must provide an upper bound  $N$  on the number of states of the target automaton  $M_G$  (or by some other means indicate when the learner has processed a structurally complete set of samples). Third, only *safe queries* (see below) must be used for candidate elimination.

The algorithm works as follows: Let  $S^+$  be a structurally complete set of examples with respect to the target FSA  $M_G$ . Based on a set  $S_0^+$  of positive examples provided by the teacher, the learner constructs the lattice  $\Omega_0$  represented by  $\Theta_0 = [\mathcal{S}_0, \mathcal{G}_0]$  and proceeds to prune  $\Omega_0$  using the teacher's responses to safe queries. When the teacher provides an additional example the lattice is incrementally updated to  $\Omega_1$  (see Sect. 3.1) corresponding to  $S_1^+ = S_0^+ \cup \{s\}$ . This interleaving of *lattice expansion* and *safe candidate elimination* is repeated until at some step  $n$ ,  $S_n^+$  is structurally complete with respect to  $M_G$ . At this point, all queries are treated as safe and the algorithm converges to the target.

At any time, the teacher can only be expected to answer queries with respect to the target ( $M_G$ ). At step  $k$  in the inference process,  $S_k^+$  is structurally complete with respect to a subautomaton  $M_{G_k}$  of  $M_G$  (where  $L(M_{G_k}) \subseteq L(M_G)$ ). An example that is not accepted by  $M_G$  will not be accepted by  $M_{G_k}$ . However, a positive example not accepted by  $M_{G_k}$  might be accepted by  $M_G$ . This makes it unsafe to perform candidate elimination using positive query strings of length greater than that of the longest positive example in  $S_k^+$ . Such query strings are unsafe and are simply ignored by the learner. All other query strings are safe for candidate elimination. At some  $k = n$ , when  $S_k^+$  is structurally complete with respect to  $M_G$  any query can be safely used for candidate elimination.

Given an upper bound  $N$  on the number of states in  $M_G$ , it can be shown that there exists a structurally complete set  $S^+$  with respect to  $M_G$  such that no string in  $S^+$  has length greater than  $2N - 1$  [16]. Thus, if the teacher provides examples in increasing order by length, the learner can infer that a structurally complete set of examples has been processed when an example of length greater than  $2N - 1$  is provided. (Note that this does not require the teacher to provide every positive example of length up to  $2N - 1$ ).

### 3.1 Incremental Lattice Update

Given the current representation of the lattice  $\Theta_k = [\mathcal{S}_k, \mathcal{G}_k]$  and a new positive example  $s$  the lattice update operation  $\oplus$  defines the new lattice  $\Theta_{k+1} = \Theta_k \oplus s$  represented by  $[\mathcal{S}_{k+1}, \mathcal{G}_{k+1}]$ . If  $s$  is accepted by the FSA corresponding to every partition in  $\mathcal{S}_k$  then  $\Theta_{k+1} = \Theta_k$ ; otherwise  $\mathcal{S}_k$  and  $\mathcal{G}_k$  are modified as follows.

Assume that  $M_s$  (with  $n$  states) is the MCA (with the corresponding partition  $P_s$ ) that accepts only the string  $s$ . For each partition  $P_y \in \mathcal{S}_k$ , a new partition  $P'_y \in \mathcal{S}_{k+1}$ , is constructed by fusing the blocks corresponding to the start states of  $M_y$  and  $M_s$  together into a single block and appending the other blocks of  $P_y$  and  $P_s$  as distinct blocks of  $P'_y$ . Thus,  $L(M'_y) = L(M_y) \cup L(M_s)$ . If  $P_y$  contains  $m$  blocks and  $P_s$  contains  $n$  blocks then  $P'_y$  contains  $m + n - 1$  blocks. For each partition of  $P_z \in \mathcal{G}_k$  a set of partitions  $\{P'_z\} \in \mathcal{G}_{k+1}$  (where the number of blocks in each  $P'_z$  is exactly the same as the number of blocks in  $P_z$ ) is constructed by fusing the blocks containing the start states of  $M_z$  and  $M_s$  together and then distributing the remaining  $n - 1$  blocks of  $P_s$  with the  $m$  blocks of  $P_z$  in  $\mathcal{D}_m^{n-1}$  ways<sup>2</sup>.

### Algorithm:

1. The teacher provides a bound,  $N$ , on the number of states in the target FSA and a set of positive examples  $S_0^+$ . Using  $S_0^+$  construct  $M_{S_0^+}$  and  $\Omega_0$  represented by  $\Theta_0 = [S_0, \mathcal{G}_0]$ . If the longest string in  $S_0^+$  is of length at most  $2N - 1$  then set  $k = 0$  and go to step 2; otherwise  $S_0^+$  is structurally complete with respect to  $M_G$ . Use the non-incremental version of the algorithm to infer the target grammar.
2. While there exists an element  $P_i \in \mathcal{S}_k$  and  $P_j \in \mathcal{G}_k$  such that the corresponding FSA  $M_i \not\equiv M_j$ , pick the shortest string  $y \in L(M_i - M_j)$  or  $L(M_j - M_i)$  and pose the query  $y \in L(G)$ ? If the query is safe then modify  $\Theta_k$  according to the candidate elimination algorithm. Ignore the query otherwise. If further safe eliminations in  $\Theta_k$  are not possible then await additional positive examples.
3. When the teacher provides a string  $s$  of length  $|s|$  do one of the following
  - (a) If  $|s| \leq 2N - 1$  then update  $\Theta_k$ , set  $k = k + 1$  and go to step 2.
  - (b) If  $|s| > 2N - 1$  then since structural completeness is achieved, resume the search for the target FSA in  $\Theta_k$  treating all queries to be safe. Return the solution to which the candidate elimination procedure converges.

For a description of the correctness of the incremental version see [16].

### Example

We use the incremental algorithm to infer the FSA in Fig. 1. Consider that the teacher provides a bound on the number of states  $N = 2$  (from which the learner infers that positive samples of length 4 or greater will not be necessary to form a structurally complete set of examples) and the example  $b$ . Thus,  $S_0^+ = \{b\}$ . The lattice  $\Omega_0$  is shown in Fig. 5. The result of the first query is shown below.

---

<sup>2</sup>  $\mathcal{D}_B^E = \sum_{i=0}^E \binom{E}{i} \mathcal{D}_{B-1}^{E-i}$  is the number of ways of distributing  $E$  distinct elements among  $B$  distinct boxes.  $\mathcal{D}_B^0 = 1$  and  $\mathcal{D}_1^E = 1$ .

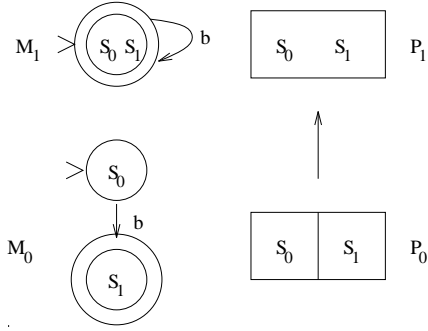


Fig. 5. Lattice  $\Omega_0$

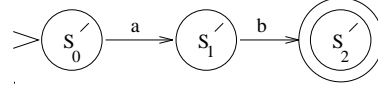


Fig. 6. MCA for  $s = ab$

$\Theta$	$M_i \equiv M_j ?$	Query $y$	Modified $\Theta$
$1 \mathcal{S}_0 = \{P_0\}; \mathcal{G}_0 = \{P_1\}$	$M_0 \not\equiv M_1$	$\lambda \in L(G)$ (SAFE)	$\mathcal{S}_0 = \{P_1\}; \mathcal{G}_0 = \{P_1\}$

The learner then waits for more positive examples. Consider that  $ab$  is provided. The MCA for  $s = ab$  is shown in Fig. 6.  $S_1^+ = \{b, ab\}$ .  $ab$  is not accepted by the FSA corresponding to the partition  $P_1 \in \mathcal{S}_0$ . The lattice update operation (shown in Fig. 7) results in the lattice  $\Omega_1$  (Fig. 8) that is implicitly represented by  $\Theta_1 = [\mathcal{S}_1, \mathcal{G}_1]$ .

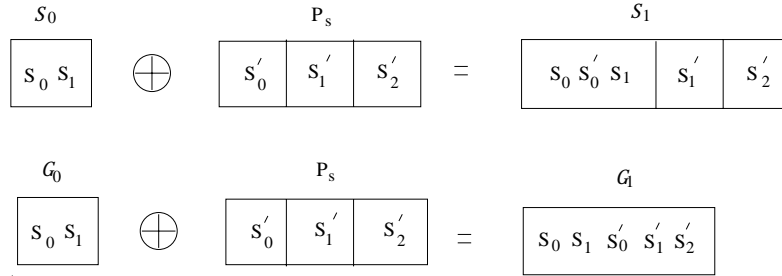


Fig. 7. Lattice Update Operation  $\oplus$

The processing of the next two queries is summarized below.

$\Theta$	$M_i \equiv M_j ?$	Query $y$	Modified $\Theta$
$2 \mathcal{S}_1 = \{P_0\}; \mathcal{G}_1 = \{P_4\}$	$M_0 \not\equiv M_4$	$a \notin L(G)$ (SAFE)	$\mathcal{S}_1 = \{P_0\}; \mathcal{G}_1 = \{P_2\}$
$3 \mathcal{S}_1 = \{P_0\}; \mathcal{G}_1 = \{P_2\}$	$M_0 \not\equiv M_2$	$abb \in L(G)$ (UNSAFE)	$\mathcal{S}_1 = \{P_0\}; \mathcal{G}_1 = \{P_2\}$

At this point no safe query can be posed so the learner is forced to wait for more examples. Note that  $S_1^+$  is structurally complete with respect to the target FSA in Fig. 1. The teacher can now provide  $abab$  which is not necessary for structural completeness (since its length is greater than  $2N - 1 = 3$ ). The learner infers that  $S_1^+$  is structurally complete with respect to  $M_G$  and the lattice is not updated. After processing a safe query the algorithm converges to the target.

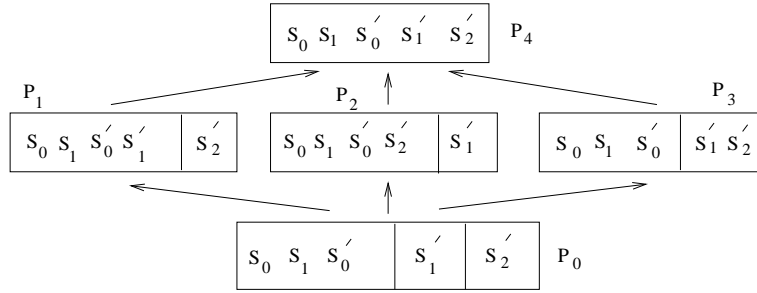


Fig. 8. Lattice  $\Omega_1$

	$\Theta$	$M_i \equiv M_j ?$	Query $y$	Modified $\Theta$
4	$\mathcal{S}_1 = \{P_0\}; \mathcal{G}_1 = \{P_2\}$	$M_0 \not\equiv M_2$	$abb \in L(G)$ (SAFE)	$\mathcal{S}_1 = \{P_2\}; \mathcal{G}_1 = \{P_2\}$

If the teacher had given examples like  $bb$  or  $bab$  when no more candidates from  $\Omega_1$  could be eliminated the lattice would not have been updated as  $M_0$  accepts both those examples. If however, the teacher had provided an example like  $abb$  (instead of  $abab$ ) that is potentially necessary for structural completeness the lattice would have had to be updated.

## 4 Summary and Discussion

Grammar inference is an important machine learning problem with several practical applications in speech recognition, computational biology, language acquisition, syntactic pattern classification, and cryptography. We have presented provably correct non-incremental and incremental versions of an algorithm for regular grammar inference. We have adopted the idea of mapping the structurally complete set of examples to an ordered lattice from the grammar inference algorithm proposed by Pao and Carr [14]. Their algorithm is not incremental and requires explicit enumeration of the entire lattice. The hypothesis space  $\Omega$  defined by the set  $S^+$  is too large to be represented explicitly or to be searched exhaustively. Our algorithm uses a compact representation of the hypothesis space in terms of  $\mathcal{S}$  and  $\mathcal{G}$ . The operations on the version space sets take time polynomial in the size of the  $\mathcal{S}$  and  $\mathcal{G}$  sets. The efficiency of our algorithm thus relies on the fact that the size of these sets at any time is not unreasonably large. The proposed algorithm uses an efficient bidirectional search strategy inspired by Mitchell's [13] version space algorithm which enables elimination of large parts of the hypothesis space based on a single query and also to make unambiguous inferences even when the algorithm has not converged. Given a current representation of the lattice in the incremental version if an example is accepted by all FSA in  $\mathcal{S}_k$  then clearly, the example is positive. When the structurally complete sample set has been acquired, an example that is not accepted by all FSA in  $\mathcal{G}_k$  can be classified as negative. The idea of incremental lattice update was inspired by Hirsh's work on *Incremental Version-Space Merging* [8]. The set  $\mathcal{G}$  in our algorithm, which represents the set of most general FSA of the lattice that do not

accept any negative strings identified by the queries during the inference process is analogous to the *border set* described by Dupont *et al* [4].

Angluin [1] has proposed an algorithm (*ID*) to infer the target grammar from a *live complete set* of examples (which can be constructed from a structurally complete set) using a polynomial number of membership queries. A trivial upper bound on the number of queries needed in our procedure is exponential in the number of states of  $M_{S^+}$ . We do not yet have an expected case analysis for the number of queries. Our approach offers an alternative to the *ID* procedure when a structurally complete set of samples is available. A direct extension of the *ID* procedure to the incremental version has not been studied.

Porat and Feldman [17] have proposed an algorithm that uses a complete ordered sample and membership queries and is guaranteed to converge in the limit. They maintain a single working hypothesis that gets modified after the presentation of each sample. Their algorithm uses a sub-routine that generates all strings in the ordered sample to check the consistency of the modified hypothesis. Since the complete ordered sample up to a particular length  $m$  is exponential in  $m$ , a trivial time complexity bound on their procedure is exponential. One potential advantage we see in maintaining a space of hypotheses (as is the case in our method) is the ability to make unambiguous inferences even when the algorithm has not converged to the target.

VanLehn and Ball [19] have proposed a version-space approach to learning context-free grammars from a set of positive and negative examples that returns a set of grammars consistent with the given sample set. Their algorithm is also based on a lattice of partitions and involves lattice updating to accommodate more evidence in the form of examples. The learner is required to store all the examples seen earlier for future reference and the version-space is represented by a triple  $[S^+, S^-, \mathcal{G}]$  where  $S^+$  and  $S^-$  represent sets of positive and negative examples respectively and  $\mathcal{G}$  is the set of generalizations. By restricting our approach to inference of regular grammars our version-space is finite and compactly represented by  $[S, \mathcal{G}]$ . Our algorithm does not store the previous examples and is guaranteed to converge to the desired target instead of a set of candidate solutions as is the case for VanLehn and Ball's method.

Angluin [2] has proposed a polynomial time algorithm ( $L^*$ ), which allows the learner to infer the target grammar by posing both membership and equivalence queries. The  $L^*$  procedure can be adapted to the *PAC* learning framework to learn from membership queries and examples alone. Rivest and Schapire [18] have suggested a *diversity* based mechanism dealing with *homing sequences*. Giles *et al* [6] use recurrent neural networks to learn FSA from positive and negative samples. Lankhorst [11] has presented a genetic algorithms based approach for learning context free grammars.

The experimental estimation of expected case time and space complexity of the proposed algorithm, generation of informative queries so as to speed up learning, sampling strategies for obtaining a structurally complete set with a high probability using a relatively small number of random samples, and extension of the proposed approach to regular *tree* and *attributed* grammars are under investigation.

## References

1. Angluin, D. A Note on the Number of Queries Needed to Identify Regular Languages. *Information and control*, 51. '81. pp 76-87.
2. Angluin, D. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75. '87. pp 87-106.
3. Biermann, A., and Feldman, J. A Survey of Results in Grammatical Inference. In Watanabe S. (ed), *Frontiers of Pattern Recognition*. Academic Press. '72. pp. 31-54.
4. Dupont, P., Miclet, L., and Vidal, E. What is the Search Space of the Regular Inference?. In *Proceedings of the ICGI-94*, Alicante, Spain, Sept. '94. pp. 25-37.
5. Fu, K. Syntactic Pattern Recognition and Applications. *Prentice-Hall, N.J.* '82.
6. Giles, C., Chen, D., Miller, H., Sun, G., and Lee, Y. Second-order Recurrent Neural Networks for Grammatical Inference. In *Proceedings of the International Joint Conference on Neural Networks 91*, vol. 2, pp. 273-281, July '91.
7. Harrison, M. Introduction to Switching and Automata Theory. *McGraw-Hill*, '65.
8. Hirsh, H. Incremental Version-Space Merging: A General Framework for Concept Learning. *Kluwer Academic Publishers*, '90.
9. Honavar, V. Toward Learning Systems That Integrate Different Strategies and Representations. In: Artificial Intelligence and Neural Networks: Steps toward Principled Integration. Honavar, V. & Uhr, L. (eds) *New York: Academic Press*, '94.
10. Hopcroft, J., and Ullman, J. Introduction to Automata Theory, Languages, and Computation. *Addison-Wesley*, '79.
11. Lankhorst, M. A Genetic Algorithm for Induction of Nondeterministic Pushdown Automata. *University of Groningen, Computer Science Report CS-R 9502*, The Netherlands. '95.
12. Miclet, L. and Quinqueton J. Learning from Examples in Sequences and Grammatical Inference. In Ferrate, G., *et al* (eds) Syntactic and Structural Pattern Recognition. *NATO ASI Series Vol. F45*, '86. pp. 153-171.
13. Mitchell, T. Generalization as search. *Artificial Intelligence*, 18. '82. pp 203-226.
14. Pao, T., and Carr, J. A solution of the Syntactic Induction-Inference Problem for Regular Languages. *Computer Languages*, Vol. 3, '78, pp. 53-64.
15. Parekh R., and Honavar, V. An Efficient Interactive Algorithm for Regular Language Learning. *Computer Science TR95-02*, Iowa State University, '95. (Preliminary version appeared in *Proceedings of the 5th UNB AI Symposium*, Fredericton, Canada, '93).
16. Parekh, R., and Honavar, V. An Incremental Interactive Algorithm for Regular Grammar Inference. *Computer Science TR96-03*, Iowa State University, '96.
17. Porat S., and Feldman J. Learning Automata from Ordered Examples. *Machine Learning*, 7, pp 109-138. '91.
18. Schapire, R., The Design and Analysis of Efficient Learning Algorithms. *MIT Press*, '92.
19. VanLehn, K. and Ball, W. A Version Space Approach to Learning Context-Free Grammars. *Machine Learning 2*, '87. pp 39-74.