An Incremental Interactive Algorithm for Regular Grammar Inference *

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

We present 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 an unknown regular grammar G implicitly specifies a lattice (or version space) which represents a space of candidate grammars containing the unknown grammar G. This lattice can be searched efficiently using membership queries to identify the unknown grammar G using an implicit representation of the version space in the form of two sets $\mathcal{S}$ and $\mathcal{G}$ 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 at any given time. We present a provably correct incremental version of the algorithm in which a structurally complete set of positive samples is not necessarily available to the learner at the beginning of learning. The learner constructs a lattice of grammars based on the strings provided at the start and performs candidate elimination by posing *safe* membership queries. When additional examples become available, the learner incrementally updates the lattice and continues with candidate elimination. Eventually, when the set of positive samples provided by the teacher encompasses a structurally complete set for the unknown grammar, the algorithm terminates by identifying the unknown grammar G.

1 Introduction

Regular Grammar Inference [3] is 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. *Grammar Inference* is an important machine learning problem with applications of

*Vasant Honavar is grateful to National Science Foundation for its support of his research through grant NSF IRI-9409580. The authors would like to thank Professor Giora Slutzki for several helpful discussions and suggestions related to this work

practical significance in pattern recognition and language acquisition [5, 9, 12]. We present an algorithm for *regular grammar inference* in an *active learning* framework wherein the learner's task is to accurately infer the unknown grammar using the teacher supplied positive examples and answers to *membership* queries.

The paper is organized as follows: Section 1.1 introduces the necessary terminology and section 1.2 formally defines the grammar inference problem. Section 2 describes a non-incremental version of the grammar inference algorithm. Section 3 explains the incremental version of the algorithm. Finally, section 4 provides a summary, discussion of related work, and directions for future research.

1.1 Regular Grammars and Finite State Automata

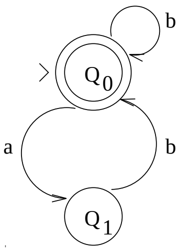

Figure 1: Finite State Automaton

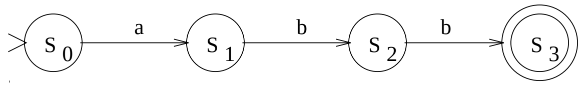

Figure 2: Canonical Automaton - M_0

A *grammar* is a finite set of rewrite (production) rules of the form $\alpha \longrightarrow \beta$ where α and β 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 $A \longrightarrow aB$ or $A \longrightarrow b$ 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. A *deterministic* FSA (DFSA), A, is a quintuple $A = (Q, \delta, \Sigma, q_0, F)$ where, Q is a finite set of states, Σ 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 δ is the transition function $Q \times \Sigma \longrightarrow Q$ that gives the next state of the automaton upon reading a particular symbol. Fig. 1 shows the state transition diagram for 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 (λ). $L(A) = \{\lambda, b, ab, bb, abb, \ldots\}$. The equivalent regular grammar is given by the following production rules: $S \longrightarrow \lambda$, $S \longrightarrow bS$, $S \longrightarrow aA$, $A \longrightarrow bS$ where S represents the start symbol. A *non-deterministic* FSA without ϵ-transitions (NFSA) is defined exactly like the DFSA except that the transition function δ: $Q \times \Sigma \longrightarrow 2^Q$ where 2^Q is the power set of Q. Intuitively, there may 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 an FSA and is defined by: $(Q_s, \delta_s, \Sigma, q_0, F_s)$ 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].

1.2 The Grammar Inference Problem

We present a mechanism for inference of a FSA (not necessarily deterministic) that is equivalent to the FSA corresponding to the unknown regular grammar (G). A teacher who is knowledgeable about the unknown grammar provides the learner with a set S^+ of positive examples. The learner maps S^+ to a hypothesis space of candidate FSA. The teacher then guides the learner in the search of the desired solution by answering *membership queries* (i.e. queries of the form "$y \in L(G)$?" where y is a string $\in \Sigma^*$.

Initially we describe a non-incremental version of the algorithm where S^+ is restricted to be a *structurally complete* set in order to facilitate a theoretical analysis of the problem domain. A structurally complete set of strings 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 by at least one string in the the set and for each accepting state of M_G there must be at least one string in the set which terminates in that accepting state. We then relax the requirement that the structurally complete set of positive examples must be available to the learner all at once and develop an incremental version of the algorithm. We refer to the unknown (or target) FSA as the FSA M_G such that $L(M_G) = L(G)$, the language of the unknown (target) grammar. In what follows, we use the terms unknown (target) grammar and unknown (target) FSA interchangeably.

2 Inference given a Structurally Complete S^+

The teacher provides a structurally complete set of positive samples S^+ which implicitly defines a lattice (or version space) Ω of candidate grammars or the initial hypothesis space that is guaranteed to contain the unknown grammar (G) [15, 16, 4]. The learner generates strings and queries the teacher about their membership in G. 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 grammars consistent with the data (sample strings, queries) processed so far. Thus, $\Theta = [\mathcal{S}, \mathcal{G}]$ provides a compact representation of the hypothesis space at all times. The teacher's response to a membership query results in pruning of the hypothesis space while ensuring that G is not eliminated in the process. The interaction between the learner and the teacher proceeds until a single grammar (or a set of grammars all of which are equivalent) equivalent to G remains in the hypothesis space.

2.1 Lattice of Grammars Specified by S^+

This section explains the construction of the lattice Ω given the set S^+. First, 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. Suppose the teacher provides a structurally complete set of strings $S^+ = \{abb\}$ from L(G). The corresponding MCA M_{S^+} is shown in Fig. 2. The lattice Ω of candidate grammars can be explicitly constructed by systematically merging the states of the MCA M_{S^+} to form partitions. Each such partition of states of M_{S^+} yields an element of Ω. The language corresponding to the FSA defined by a partition 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 MCA (Fig. 2) is depicted in Fig. 3. A MCA with m states yields an initial hypothesis space that contains: $E_m = \sum_{j=0}^{m-1} \binom{m-1}{j} E_j$ 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 *implicitly* using $\Theta = [\mathcal{S}, \mathcal{G}]$ as outlined earlier and explained in section 2.2.

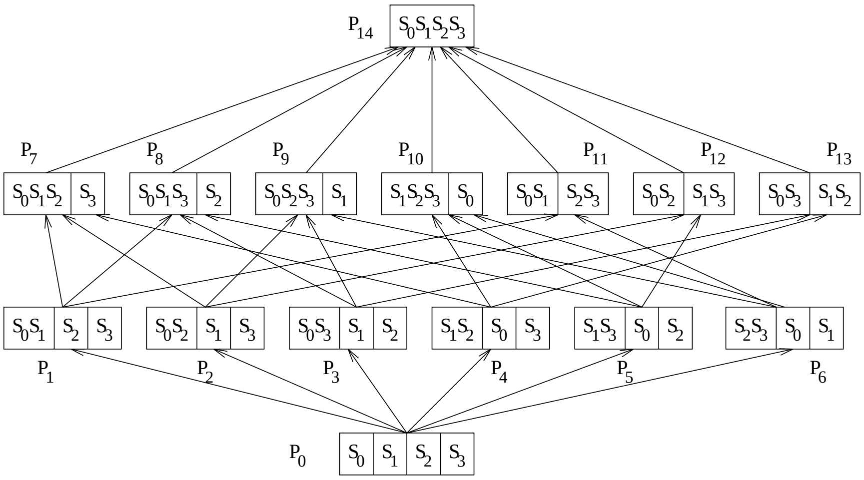

Figure 3: Lattice - Ω

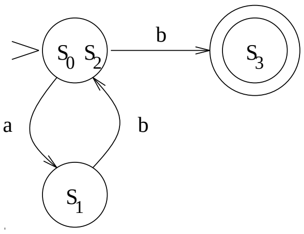

Figure 4: FSA corresponding to the partition P_2

Each element of the lattice (i.e., a partition of the states of S^+) corresponds to a FSA M constructed as follows: The states of M correspond to the cells of the partition. Each cell contains one or more states of M_{S^+} that are grouped together in the partition. For the automaton M, the start state is the cell which contains the start state 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, δ_M, is defined on the basis of the transitions within M_{S^+}. If several states of M_{S^+} are merged together in a cell c in a partition, then the transitions into each of those states become transitions into the state represented by c in M and the transitions out of each of these states become the transitions out of the state represented by c in M. Transitions between two states that get merged in cell c form self loops on the state resulting from the merger. The FSA corresponding to the partition P_2 (of Fig. 3) is shown in Fig. 4.

The lattice Ω of grammars (or equivalently FSA, 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 ($P_i \preceq P_j$). 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 covers 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 cells of the partitions under consideration. The learner exploits this property while pruning the hypothesis space as explained below.

2.2 Query-Aided Bi-Directional Search of the Lattice

As already noted, the hypothesis space of candidate grammars is represented implicitly by $\Theta = [\mathcal{S}, \mathcal{G}]$. $\mathcal{S}$ is the set of maximally *specific* elements and $\mathcal{G}$ is the set of maximally *general* elements of the hypothesis space that are consistent with all the data gathered by the learner at any time. Initially, $\mathcal{S} = \{\mathrm{P}_0\}$, the partition corresponding to the MCA $\mathrm{M}_{\mathrm{S}^+}$ and $\mathcal{G} = \{\mathrm{P}_{E_m - 1}\}$, the partition corresponding to the most general element of the lattice Ω i.e. the one with all states of the MCA merged together in a single cell. Note that E_m is the total number of partitions in the lattice Ω. During the search the learner constructs fully specified, deterministic finite state automata ($\mathrm{M}_i = \{\mathrm{S}, \delta_s, \Sigma, \mathrm{s}_0, \mathrm{A}\}$ and $\mathrm{M}_j = \{\mathrm{T}, \delta_t, \Sigma, \mathrm{t}_0, \mathrm{B}\}$) for partitions $\mathrm{P}_i \in \mathcal{S}$ and $\mathrm{P}_j \in \mathcal{G}$. M_i and M_j are compared for equivalence. If they are not equivalent then there exists a string y such that $\delta_s(\mathrm{s}_0, y) \in \mathrm{A}$ but $\delta_t(\mathrm{t}_0, y) \notin \mathrm{B}$ 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 $\mathrm{M}_i - \mathrm{M}_j = \{\mathrm{W}, \delta_w, \Sigma, \mathrm{w}_0, \mathrm{C}\}$ where, W = S $\times$ T, $\mathrm{w}_0 = (\mathrm{s}_0, \mathrm{t}_0)$, $\delta_w((\mathrm{s,t}), \sigma) = (\delta_s(\mathrm{s}, \sigma), \delta_t(\mathrm{t}, \sigma))$ for all $\sigma \in \Sigma$ and $\mathrm{C} = \{(\mathrm{s,t}) \mid \mathrm{s} \in \mathrm{A}$ and $\mathrm{t} \in \mathrm{T} - \mathrm{B}\}$ [7] forms the query "$y \in \mathrm{L(G)}$"? that is posed to the teacher. Based on the teacher's response Θ is pruned and elements of $\mathcal{S}$ and $\mathcal{G}$ become progressively *more general* and *more specific* respectively.

Since the lattice Ω 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 $\mathcal{S} = \{\mathrm{P}_0\}$ and $\mathcal{G} = \{\mathrm{P}_{E_m - 1}\}$.
2. While there exists an element $\mathrm{P}_i \in \mathcal{S}$ and $\mathrm{P}_j \in \mathcal{G}$ such that the corresponding FSA $\mathrm{M}_i \not\equiv \mathrm{M}_j$ pick the shortest string $y \in \mathrm{L}(\mathrm{M}_i - \mathrm{M}_j)$ or $\mathrm{L}(\mathrm{M}_j - \mathrm{M}_i)$ and pose the query $y \in \mathrm{L(G)}$? Based on the teacher's response modify Θ as described below.
3. Return the FSA corresponding to the partition to which the search of the lattice (Ω) converges.

Candidate Elimination:

1. If y is a positive example
   (a) Remove any partition $\mathrm{P}_k \in \mathcal{G}$ for which the corresponding FSA M_k rejects y.
   (b) Minimally generalize any partition $\mathrm{P}_l \in \mathcal{S}$ if M_l does not accept y till the FSA corresponding to the generalizations 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 partition $\mathrm{P}_k \in \mathcal{S}$ for which the corresponding FSA M_k accepts y.
   (b) Minimally specialize any partition $\mathrm{P}_l \in \mathcal{G}$ if M_l accepts y till the FSA corresponding to the specializations do not accept 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 $\mathcal{G}$.

Elimination of candidate solutions proceeds as explained above until all the remaining partitions in $\mathcal{S}$ and $\mathcal{G}$ correspond to equivalent FSA. The proof of correctness (see Appendix A.1) shows that each such FSA is guaranteed to be equivalent to the unknown target FSA corresponding to the target language L(G). The following example illustrates the working of the algorithm given $S^+ = \{abb\}$ 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. Alternatively, this may be guided by a suitable heuristic.)

Example

Step	Θ	$M_i \equiv M_j$?	Query y	Modified Θ
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	$\mathcal{S} = \{P_3\}$; $\mathcal{G} = \{P_9, P_{13}\}$	$M_3 \not\equiv M_9$	$b \in L(G)$	$\mathcal{S} = \{P_9\}$; $\mathcal{G} = \{P_9\}$

Thus, the algorithm terminates with a solution M_9 which is the target FSA.

3 An Incremental Algorithm for Grammar Inference

In inductive learning it is often the case that the entire training data is not available to the learner at the start. This motivates the need for incremental algorithms that enable the learner to develop suitable hypotheses based on the available data available and, when presented with additional training data, update the hypotheses suitably without having to reprocess previous data (samples, queries).

We explore an incremental algorithm that relaxes the requirement that the structurally complete set of positive examples must be provided by the teacher at the start. The teacher provides a set S_0^+ of positive samples that is not necessarily structurally complete. Based on this the learner implicitly constructs a lattice of candidate grammars (as explained in section 2.1) and then performs candidate elimination by posing what are characterized as *safe queries* (see below for details). When no further eliminations are possible the learner waits for the teacher to provide additional examples. The additional positive examples provided by the teacher are used to update the existing lattice as necessary. This interleaving of *lattice expansion* and *safe candidate elimination* continues until the positive examples provided by the teacher represent a structurally complete set of examples with respect to the target grammar. The learner then uses the bi-directional search algorithm (section 2.2) with all queries regarded as safe queries to converge to the target. The safe nature of the queries used for candidate elimination ensures that the desired solution is never eliminated from the lattice (provided the teacher presents the necessary positive examples in increasing order of length).

3.1 Structural Completeness in the Limit

Consider a set S^+ that is structurally complete with respect to the target FSA M_{G^*}. The set of positive examples provided by the teacher S_0^+ represents a structurally complete set for the subautomaton ($M_{G_0^*}$) defined by those states and transitions of the unknown target (M_{G^*}) that are covered by the examples. From the correctness proof of the non-incremental version of the algorithm (see Appendix A.1) we know that the lattice constructed from the examples in S_0^+ and represented implicitly by sets $\mathcal{S}_0$ and $\mathcal{G}_0$ is guaranteed to contain $M_{G_0^*}$. Assume that after possibly a few eliminations the teacher provides the learner with another positive sample s. If s exercises some transition of M_{G^*} or terminates in some accepting state of M_{G^*} that is not covered by any string in S_0^+ the set $S_1^+ = S_0^+ \cup \{s\}$ is a structurally complete set for the subautomaton $M_{G_1^*}$, of M_{G^*} and $L(M_{G_0^*}) \subseteq L(M_{G_1^*}) \subseteq L(M_{G^*})$. It will be shown in section 3.3 that the updated lattice represented by the sets $\mathcal{S}_1$ and $\mathcal{G}_1$ will contain $M_{G_1^*}$.

Eventually, after the teacher has provided an adequate number of positive examples the set $S_n^+ \supseteq S^+$ and structural completeness is achieved. The learner must either be told or must infer when structural completeness is achieved with respect to the target FSA. If the target FSA has N states then it is clear that positive samples of length greater that $2N-1$ are not necessary to form a structurally complete set of examples. This follows from the fact that in order for a string to traverse a particular arc, $\delta(p, a) = q$, of a FSA with N states, at most $N-1$ arcs will need to be traversed to reach state p from the start states and similarly at most $N-1$ arcs will be needed to reach an accepting state from state q. We require the teacher to provide positive examples in increasing order by length since otherwise, it is not possible to guarantee convergence to the unknown target grammar (a proof of this claim is given in the Appendix A.4). Given the knowledge of N and the fact that the teacher provides examples in increasing order by length, the learner can infer when structural completeness is achieved (i.e. when the teacher provides an example that is of length greater than $2N-1$). Note that the teacher need not provide every positive sample of length at most $2N-1$. It suffices to ensure that no sample that is necessary for structural completeness (i.e., a sample that exercises some transition or terminates in some accepting state which would not otherwise be covered by any other positive example of length at most $2N-1$) is missed.

3.2 Safe Queries

The teacher will answer membership queries with respect to G. At step k in the inference process the set S_k^+ will be structurally complete with respect the the subautomaton $M_{G_k^*}$. With $L(M_{G_k^*}) \subseteq L(M_{G^*})$, 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^*}. Thus query strings that are labeled positive by the teacher are potentially unsafe for elimination. Since the teacher provides strings in increasing order by length, it is clear that query strings that are labeled positive and are longer than the length of the longest positive example in S_k^+ are potentially unsafe for candidate elimination, and hence are simply ignored by the learner. At some $k = n$, the set S_k^+ is structurally complete with respect to M_{G^*} and from this point on all queries can be treated as safe queries for candidate elimination.

3.3 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 to give $\mathcal{S}_{k+1}$ and $\mathcal{G}_{k+1}$ respectively. Intuitively, the lattice update procedure can be described as follows.

Assume that M_s (with n states) is the MCA (with the corresponding partition P_s) that accepts only 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 cells corresponding to the start states of M_y and M_s together in to a single cell and appending the other cells of P_y and P_s as distinct cells. Thus, $L(M_y^{'}) = L(M_y) \cup L(M_s)$ If P_y contains m cells and P_s contains n cells then $P_y^{'}$ contains $m + n - 1$ cells. 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 cells in each $P_z^{'}$ is exactly the same as the number of cells in P_z) is constructed by fusing the cells containing the start states of M_z and M_s together and then distributing the contents of the remaining cells of P_s among the cells of P_z in all possible combinations.

Owing to space constraints we omit a formal description of the lattice update procedure. This appears in the appendix A.2.

3.4 The Incremental Algorithm

The incremental grammar inference algorithm is described as follows:

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 the MCA $M_{S_0^+}$ and a representation of the lattice Ω_0 by $\Theta_0 = [\mathcal{S}_0, \mathcal{G}_0]$. 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 unknown 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 $y \in M_i - M_j$ and pose the query $y \in L(G)$? If the query is safe then modify Θ_k according to the version space strategy described in section 2.2. Ignore the query otherwise. If further safe eliminations in Θ_k are not possible then wait for the teacher to provide more positive examples.

3. When the teacher provides another positive example do one of the following

   (a) If the length of the positive example is up to $2N - 1$ then update the lattice to Θ_{k+1} as described in section 3.3, set $k = k + 1$ and go to step 3.

   (b) If the length of the positive example is greater than $2N - 1$ then, since structural completeness is achieved, resume the search for the target FSA (as described in section 2.2) in Θ_k treating all queries to be safe. Return the solution to which the candidate elimination procedure converges.

The correctness proof of this algorithm appears in the Appendix (A.3).

Example

In this example we demonstrate the application of the incremental grammar inference algorithm by inferring the FSA in Fig. 1. Consider that initially the teacher provides a bound on the number of states $N = 2$ (from which the learner infers that positive samples of length 4 will be unnecessary for structural completeness) and the example b. Thus, $S_0^+ = \{b\}$. The learner constructs the lattice Ω_0 shown in Fig. 5. The first query is processed as shown below.

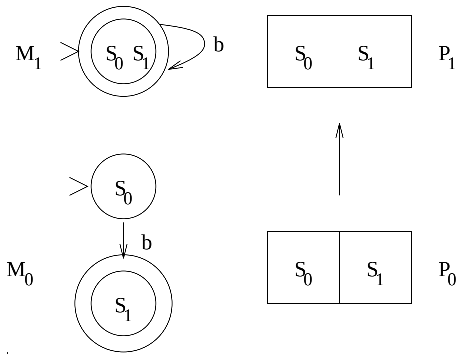

Figure 5: Lattice Ω_0

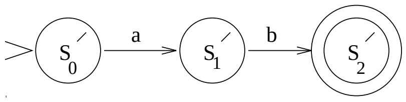

Figure 6: Canonical automaton for $s = ab$

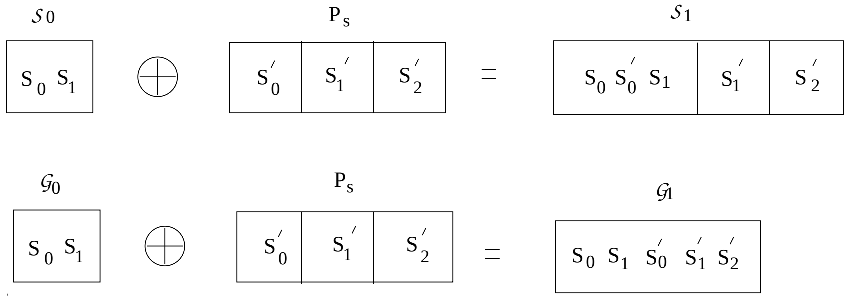

Figure 7: Lattice Update Operation $\oplus$

	Θ	$M_i \equiv M_j$?	Query y	Modified Θ
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 the teacher to provide more positive examples. Consider that the 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 as shown in Fig. 7 (see section 3.3) results in the lattice Ω_1 (Fig. 8) that is implicitly represented by $\Theta_1 = [\mathcal{S}_1, \mathcal{G}_1]$. The processing of the next two queries is summarized below.

	Θ	$M_i \equiv M_j$?	Query y	Modified Θ
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 until the teacher provides more examples. Note that the set 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.

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

After processing a safe query the algorithm correctly converges to the target. If the teacher had given examples like bb or bab after the point where no more candidates from Ω_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.

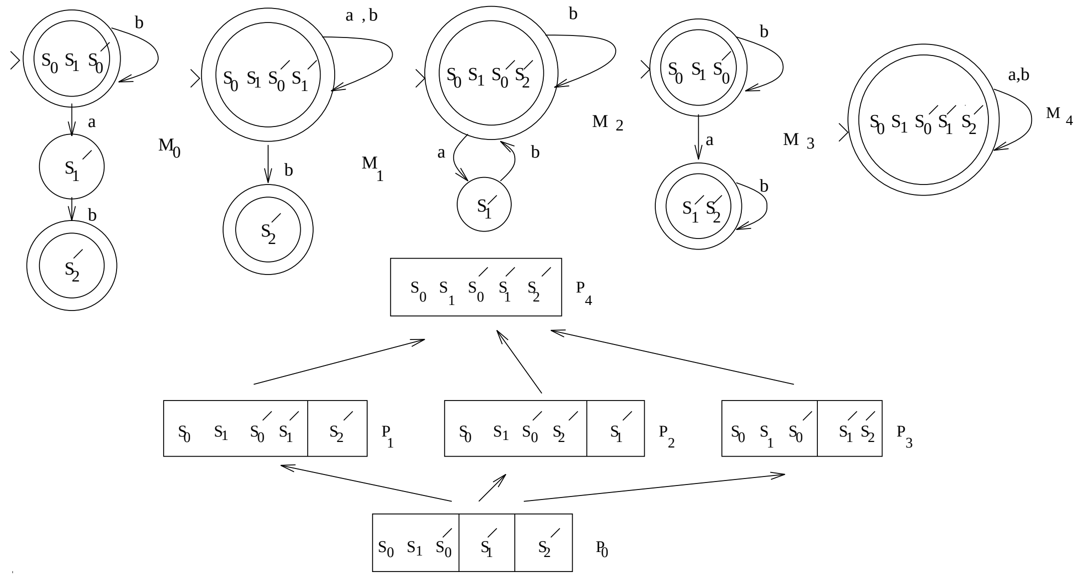

Figure 8: Lattice Ω_1

4 Summary and Discussion

Grammar inference is an important machine learning problem with several practical applications including speech recognition, computational biology (e.g., DNA sequence recognition), language acquisition, syntactic pattern classification, and cryptography. We have presented provably correct incremental and non-incremental algorithms for inference of regular grammars. Both algorithms are guaranteed to converge to the unknown target. This work is 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. Their algorithm is not incremental and requires explicit enumeration of the entire lattice. The hypothesis space Ω 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 proposed algorithm uses an efficient bidirectional search strategy inspired by Mitchell's [13] version space algorithm. This enables the learner to eliminate large parts of the hypothesis space based on a single informative query and also to make unambiguous inferences even though the algorithm has not converged to a single solution. 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].

Angluin [1] has proposed an algorithm *ID* to infer the target grammar from a *live complete set* of examples using a polynomial number of membership queries. The live complete set can be constructed given a structurally complete sample. A trivial upper bound on the number of membership queries needed in our procedure is exponential in the size of the MCA constructed for the structurally complete set 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 our algorithm has not converged to the desired solution.

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. Some advantages of Mitchell's version-space are lost in the sense that their approach requires the learner to store all the examples seen earlier for future reference. Their version-space is represented by a triple $[\mathrm{S}^+, \mathrm{S}^-, \mathcal{G}]$ 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 $[\mathcal{S}, \mathcal{G}]$. Our algorithm does not store the previous examples and is guaranteed to converge to the desired target solution instead of a set of candidate solutions as is the case for VanLehn and Ball's method.

Several other versions of grammar inference problem have been studied extensively in the literature. 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. Natarajan [14] has described an adaptation of Angluin's L^* procedure to the PAC learning framework. 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 using positive and negative samples. Lankhorst [11] has presented a genetic algorithms based approach for learning context free grammars.

Experimental analysis is currently underway to study the expected case time and space complexity bounds for our method. Promising directions for further exploration include: mechanisms for generating 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; and empirical estimates of the expected case time and space complexity of the proposed grammar algorithm and its extensions.

References

[1] Angluin, D. A Note on the Number of Queries Needed to Identify Regular Languages. *Information and control*, 51. 1981. pp 76-87.

[2] Angluin, D. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75. 1987. pp 87-106.

[3] Biermann, A.W. and Feldman, J.A. A Survey of Results in Grammatical Inference. In Watanabe S. (editor), *Frontiers of Pattern Recognition.* Academic Press. 1972. pp. 31-54.

[4] Dupont, P., Miclet, L., and Vidal, E. What is the Search Space of the Regular Inference?. In *Proceedings of the Second International Colloquium on Grammatical Inference, ICGI-94*, Alicante, Spain, September 1994. pp. 25-37.

[5] Fu. K.S. Syntactic Pattern Recognition and Applications. *Prentice-Hall, Inc., N.J.* 1982.

[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 1991.

[7] Harrison, M. Introduction to Switching and Automata Theory. *McGraw-Hill, NY* 1965.

[8] Hirsh, H. Incremental Version-Space Merging: A General Framework for Concept Learning. *Kluwer Academic Publishers.* 1990.

[9] Honavar, V.G. Toward Learning Systems That Integrate Different Strategies and Representations. In: Artificial Intelligence and Neural Networks: Steps toward Principled Integration. Honavar, V. & Uhr, L. (editors) *New York: Academic Press*, 1994.

[10] Hopcroft, J., and Ullman, J. Introduction to Automata Theory, Languages, and Computation. *Addison-Wesley Publishing Company, Inc.* 1979.

[11] Lankhorst, M. A Genetic Algorithm for Induction of Nondeterministic Pushdown Automata. *University of Groningen, Computer Science Report CS-R 9502*, The Netherlands. 1995.

[12] Miclet, L. and Quinqueton J. Learning from Examples in Sequences and Grammatical Inference. In Ferrate, G., Pavlidis, T., Sanfeliu, A., and Bunke H. (editors) Syntactic and Structural Pattern Recognition. *NATO ASI Series* Vol. F45, 1986. pp. 153-171.

[13] Mitchell, T. Generalization as search. *Artificial Intelligence*, 18. 1982. pp 203-226.

[14] Natarajan, B. K., Machine Learning: A Theoretical Approach. *Morgan Kaufmann* San Mateo, California. 1992.

[15] Pao, T.W.L., and Carr, J.W.III. A solution of the Syntactic Induction-Inference Problem for Regular Languages. *Computer Languages*, Vol. 3 1978, pp. 53-64.

[16] Parekh R.G., and Honavar V.G. An Efficient Interactive Algorithm for Regular Language Learning. *Computer Science TR95-02*, Iowa State University, Ames. 1995. (Preliminary version appeared in *Proceedings of the 5th UNB AI Symposium*, Fredericton, Canada, 1993).

[17] Porat S., and Feldman J. A., Learning Automata from Ordered Examples. *Machine Learning*, 7, pp 109-138. 1991.

[18] Schapire, R., The Design and Analysis of Efficient Learning Algorithms. *MIT Press*, 1992.

[19] VanLehn, K. and Ball, W. A Version Space Approach to Learning Context-Free Grammars. *Machine Learning 2*, 1987. pp 39-74.

A Appendix

A.1 Correctness of the Non-Incremental Grammar Inference Algorithm

The proof of correctness of the non-incremental version of the grammar inference algorithm follows from the following two theorems:
Theorem 1:[1] The target grammar G lies in the lattice Ω defined by a structurally complete set of strings from G.
Proof:
Let $Z_i = \{S, \delta_s, \Sigma, s_0, A\}$ and $Z_j = \{T, \delta_t, \Sigma, t_0, B\}$ 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 $\delta_t(t_y, \gamma) = t_j$ where $(\gamma \in \Sigma)$.

Lemma 2: 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 ψ 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.

$$x \in \text{path}(s_0, s_i) \Longleftrightarrow x \in \text{path}(\psi(s_0), \psi(s_i)) \text{ for all } s_i \text{ of } Z_i \tag{1}$$

We prove $x \in \text{path}(s_0, s_i) \Longrightarrow x \in \text{path}(\psi(s_0), \psi(s_i))$ for all s_i of Z_i by mathematical induction on the length of string x (i.e. $\mid x \mid$).
Base Case:
If $\mid x \mid = 1$ 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 $\mid x \mid <= n$.
If $\mid x \mid = n$, then $x \in \text{path}(s_0, s_{pn}) \Longrightarrow x \in \text{path}(\psi(s_0), \psi(s_{pn}))$ for a state s_{pn} of Z_i.

[1]The proof of theorem 1 is originally due to Pao and Carr [15] and has been reworked in [16]. It was also independently proven by Miclet. This version of the proof appears in [4]

Induction Proof:
Now we show that the lemma is true for $|x| = n + 1$. Let $x = a_1a_2a_3 \ldots a_{n+1}$. 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_{pn}. Thus, $a_1a_2a_3 \ldots a_n$ is a path from $\psi(s_0)$ to $\psi(s_{pn})$. Now $s_0 \sim t_0$ and there is only one state $t_{qn} \sim s_{pn}$. From this we conclude that $a_1a_2a_3 \ldots a_n$ is a path from t_0 to t_{qn}. 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_{qn} 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 construct the MCA M_{S^+}. Let M_G be the FSA corresponding to G. We establish the following two lemmas.

Lemma 3: 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 $s_{p1}, s_{p2}, \ldots, s_{pn}$ 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 $2 \leq j \leq n$. Since x is contained in L(G) there are states $t_{r1}, t_{r2}, \ldots, t_{rn}$ in M_G such that a_1 is an edge from t_0 to a state t_{r1}, 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 $1 \leq j \leq n$. 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 4: 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_{rk} to t_{rl} there exist states s_{pk} and s_{pl} 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 $s_i \sim t_j$. □

Constructing the solution automaton M_{G^*}
Merge each state $s_{p1}, s_{p2}, \ldots, s_{pk}$ of the automaton M_{S^+} such that each $s_{pn} \sim t_j$ where n = 1,2, ...,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 3 we know that each s_{pn} (n = 1,2, ...) is merged into exactly one s_i. From lemma 4 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_{pG} of M_{S^+} is merged into s_F and s_{pG} is an accepting state for M_{S^+}. Thus, using lemma 2 lemma 3, lemma 4, and the above method of constructing the solution automaton we have proved that $M_{G^*} \equiv M_G$. □

Theorem 5: 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_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, $\mathcal{S}$ contains only one partition — that corresponding to the MCA M_{S^+} and $\mathcal{G}$ contains only one partition — that corresponding to the most general FSA in the lattice. Therefore, the hypothesis space $\Theta = [\mathcal{S}, \mathcal{G}]$ implicitly includes the entire lattice Ω. Theorem 1 guarantees that a partition corresponding to the target grammar lies within Ω, and hence in the hypothesis space Θ. Clearly, the invariance condition holds if we set P_y to be the partition corresponding to M_{S^+} and P_z to be the most general partition in Ω.
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 y. If the query string y is a positive example:

1. Any partition $P_k \in \mathcal{G}$ for which the corresponding FSA M_k does not accept y is removed. Clearly, no such P_k could be P_z or else M_{G*} would also not accept the positive example 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 MSE 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. □

A.2 Lattice Update Operation $\oplus$

Consider the MCA $M_s = (Q_s, \delta_s, \Sigma, q_{0_s}, F_s)$ that only accepts the string s. The corresponding partition P_s contains n cells $P_{s1}, \ldots P_{sn}$ where $n = \mid Q_s \mid$. Let the automaton $M_y = (Q_y, \delta_y, \Sigma, q_{0_y}, F_y)$ correspond to a partition P_y in $\mathcal{S}_k$ and the automaton $M_z = (Q_z, \delta_z, \Sigma, q_{0_z}, F_z)$ correspond to a partition P_z in $\mathcal{G}_k$. Further assume that the first cell i.e. P_{s1}, P_{y1}, and P_{z1} of the partitions P_s, P_y, and P_z respectively contains the start state of the corresponding FSA in each case. For each partition P_y in $\mathcal{S}_k$ a new partition $P_y^{'} \in \mathcal{S}_{k+1}$ is constructed. If P_y contains m cells then the new partition $(P_y^{'})$ contains $m + n - 1$ cells.

1. The cell $P_{y1}^{'}$ is obtained by fusing together the cells P_{y1} and P_{s1}.

2. The cells $P'_{y2} \ldots P'_{ym}$ are simply the cells $P_{y2} \ldots P_{ym}$ and the cells $P'_{ym+1} \ldots P'_{ym+n-1}$ are cells $P_{s2} \ldots P_{sn}$.

The FSA $M'_y = (Q'_y, \delta'_y, \Sigma, q'_{0_y}, F'_y)$ is defined as follows: $Q'_y = \{P'_{yi} \mid P'_{yi}$ is a cell in $P'_y\}$; $\delta'_y(P'_{yi}, a) = P'_{yj}$ $(i, j = 1 \ldots m+n-1)$ if $\exists p \in P'_{yi}$ and $\exists q \in P'_{yj}$ with $p, q \in Q_y$ and $\delta_y(p, a) = q$ or $p, q \in Q_s$ and $\delta_s(p, a) = q$; $q'_{0_y} = P'_{y1}$; $F_y{}' = \{P'_{yi} \mid P'_{yi}$ is a cell in P'_y and $\exists q \in P'_{yi}$ such that $q \in F_y$ or $F_s\}$.

For each partition $P_z \in \mathcal{G}_k$ a set of new partitions $\{P'_z\}$ is constructed. The union of sets $\{P'_z\}$ constructed from the different $P_z \in \mathcal{G}_k$ constitutes the set $\mathcal{G}_{k+1}$. If P_z contains m cells then each partition (P'_z) in the set of new partitions, $\{P'_z\}$, contains m cells.

1. The cell P'_{z1} of P'_z is obtained by fusing together the cells P_{z1} and P_{s1}.

2. The contents of cells $P_{z2} \ldots P_{zm}$ of P_z are copied to the cells $P'_{z2} \ldots P'_{zm}$ respectively.

3. Finally the cells $P_{s2} \ldots P_{sn}$ are distributed among the cells $P'_{z1} \ldots P'_{zm}$ of P'_z.

The FSA $M'_z = (Q'_z, \delta'_z, \Sigma, q'_{0_z}, F'_z)$ is defined as follows: $Q'_z = \{P'_{zi} \mid P'_{zi}$ is a cell in $P'_z\}$; $\delta'_z(P'_{zi}, a) = P'_{zj}$ $(i, j = 1 \ldots m)$ if $\exists p \in P'_{zi}$ and $\exists q \in P'_{zj}$ with $p, q \in Q_z$ and $\delta_z(p, a) = q$ or $p, q \in Q_s$ and $\delta_s(p, a) = q$; $q'_{0_z} = P'_{z1}$; $F_z{}' = \{P'_{zi} \mid P'_{zi}$ is a cell in P'_z and $\exists q \in P'_{zi}$ such that $q \in F_z$ or $F_s\}$. Fig. 9 depicts an example of the lattice update process.

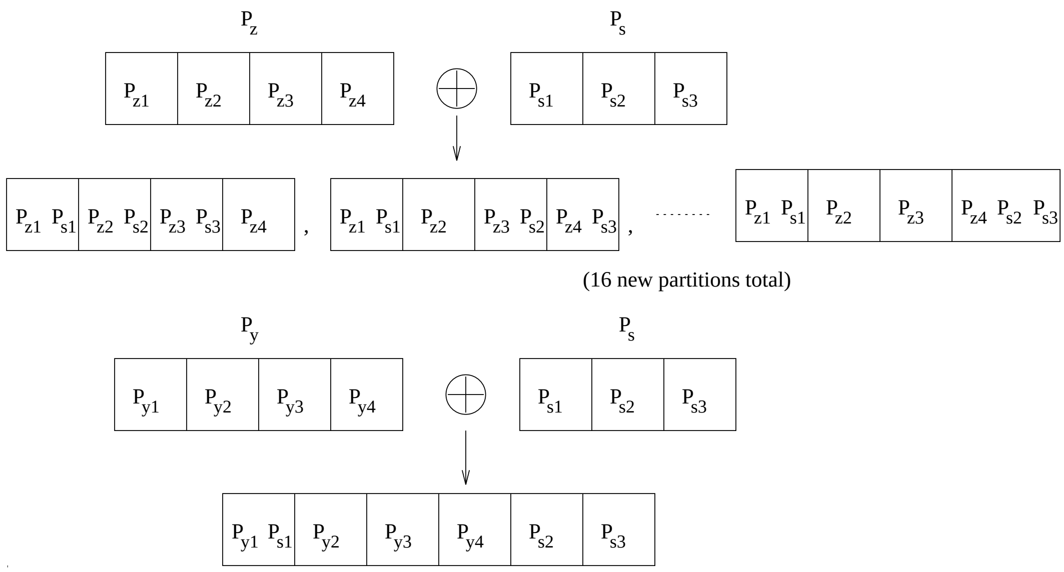

Figure 9: Example of Lattice Updating operation $\oplus$

A.3 Correctness of the Incremental Algorithm

The correctness of the incremental version of the grammar inference algorithm can be argued as follows. Since the sample set S_0^+ is structurally complete with respect to the subautomaton $M_{G_0^*}$ theorem 1 ensures that $M_{G_0^*}$ is contained in the lattice space defined by $\Theta_0 = [\mathcal{S}_0, \mathcal{G}_0]$. Safe queries are more restrictive than the queries encountered in the non-incremental version of the algorithm. Thus using theorem 5 we can ensure that at all times the following invariance condition is satisfied:

$$\exists\ \mathrm{P}_z \in \mathcal{G}_0 \text{ and } \exists\ \mathrm{P}_y \in \mathcal{S}_0 \text{ such that } \mathrm{P}_y \preceq \mathrm{P}_{M_{G_0^*}} \preceq \mathrm{P}_z$$

<u>**Claim 6**</u>: Consider that there exists partition $\mathrm{P}_z \in \mathcal{G}_k$ and $\mathrm{P}_y \in \mathcal{S}_k$ in the current representation of the lattice $\Theta_k = [\mathcal{S}_k, \mathcal{G}_k]$ such that $\mathrm{P}_y \preceq \mathrm{P}_{M_{G_k^*}} \preceq \mathrm{P}_z$ where $\mathrm{M}_{G_k^*}$ is subautomaton of M_{G^*} and the set of samples S_k^+ is structurally complete with respect to $\mathrm{M}_{G_k^*}$. Consider that a string s is provided such that $\mathrm{S}_{k+1}^+ = \mathrm{S}_k^+ \cup \{s\}$ is structurally complete with respect to the subautomaton $\mathrm{M}_{G_{k+1}^*}$ of the target M_{G^*}. There exists a partition $\mathrm{P}_{M_{G_{k+1}^*}}$ (corresponding to $\mathrm{M}_{G_{k+1}^*}$) in the updated lattice represented by $\Theta_{k+1} = [\mathcal{S}_{k+1}, \mathcal{G}_{k+1}]$ and there exist partitions $\mathrm{P}_z^{'} \in \mathcal{G}_{k+1}$ and $\mathrm{P}_y^{'} \in \mathcal{S}_{k+1}$ in Θ_{k+1} such that $\mathrm{P}_y^{'} \preceq \mathrm{P}_{M_{G_{k+1}^*}} \preceq \mathrm{P}_z^{'}$.
The proof follows directly from the method used to construct the modified lattice during the lattice update operation.

Since the subautomaton $\mathrm{M}_{G_{k+1}^*}$ is contained in the space of candidate grammars Ω_{k+1} and the invariance is preserved after the operation $\oplus$, we conclude that for some $k = n$ when the set S_n^+ is structurally complete with respect to M_{G^*}, then M_{G^*} is contained in the space Ω_k and the invariance is preserved. From this point on all queries are safe and by application of theorem 2 we conclude that the procedure will converge to the desired target. □

A.4 Length order restriction in the incremental algorithm

<u>**Claim 7**</u>: The incremental version of the grammar inference algorithm could possibly converge to an incorrect solution if the teacher does not provide positive examples in increasing order by length.

<u>**Proof**</u>: We present an example to show that convergence to the desired solution is not guaranteed when the teacher fails to provide examples in increasing order by length.

Consider that we wish to infer the FSA shown in Fig. 10. The structurally complete set S^+ for this FSA is $\{\lambda, a, aa\}$. Note that strings in S^+ together exercise every transition and visit every accepting state of the target FSA at least once. Assume that the teacher provides positive examples $\{\lambda, aa\}$. Thus, $\mathrm{S}_0^+ = \{\lambda, aa\}$. The lattice constructed from the canonical automaton for S_0^+ is shown in Fig. 11. It is represented implicitly by $\Theta_0 = [\mathcal{S}_0, \mathcal{G}_0]$.

Step	Θ	$\mathrm{M}_i \equiv \mathrm{M}_j$?	Query y	Modified Θ
1	$\mathcal{S}_0 = \{\mathrm{P}_0\}; \mathcal{G}_0 = \{\mathrm{P}_4\}$	$\mathrm{M}_0 \not\equiv \mathrm{M}_3$	$a \in \mathrm{L(G)}$? (SAFE)	$\mathcal{S}_0 = \{\mathrm{P}_1, \mathrm{P}_3\}; \mathcal{G}_0 = \{\mathrm{P}_4\}$

After processing the query $y = a$ the partitions in $\mathcal{S}_0$ and $\mathcal{G}_0$ represent FSA that are equivalent to each other. No further elimination is possible. Suppose, next the teacher provides the string $s = a$. This string is accepted by both FSA in $\mathcal{S}_0$ so the lattice is not modified and $\Theta_1 = \Theta_0$. Clearly the set $\mathrm{S}_1^+ = \mathrm{S}_0^+ \cup \{a\}$ is structurally complete with respect to the target FSA. Since no more lattice elements can be eliminated, the procedure terminates with equivalent FSA whose language is a^*. Clearly, this is not the language of the target FSA. □

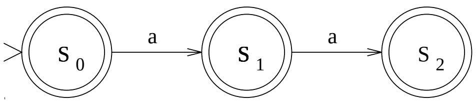

Figure 10: Target FSA

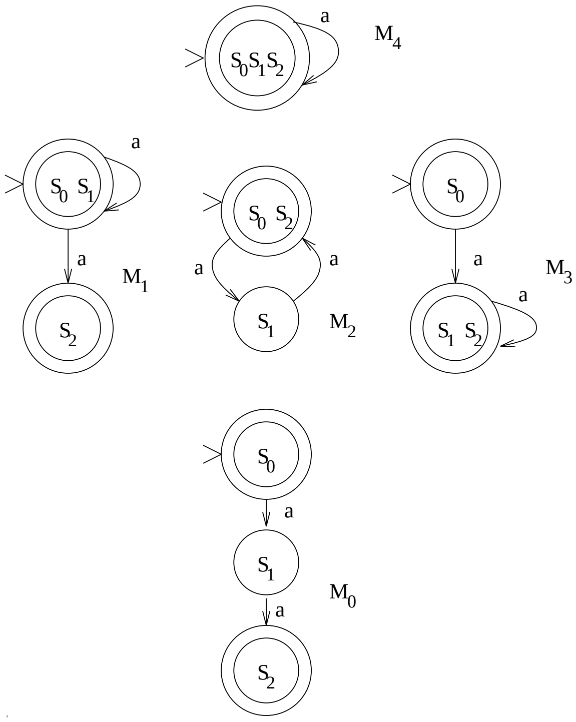

Figure 11: Latice Ω_0 constructed from $S_0^+ = \{\lambda, aa\}$