1-17-1997

A Polynomial Time Incremental Algorithm for Regular Grammar Inference

Rajesh G. Parekh
Iowa State University

Codrin Nichitiu
Iowa State University

Vasant Honavar
Iowa State University

A Polynomial Time Incremental Algorithm for Regular Grammar Inference

Abstract
We present an efficient incremental algorithm for learning regular grammars from labeled examples and membership queries. This algorithm is an extension of Angluin's {\em ID} procedure to an incremental framework. The learning algorithm is intermittently provided with labeled examples and has access to a knowledgeable teacher capable of answering membership queries. Based on the observed examples and the teacher's responses to membership queries, the learner constructs a deterministic finite automaton (DFA) with which all examples observed thus far are consistent. When additional examples are observed, the learner modifies this DFA suitably to encompass the information provided by the new examples. In the limit this algorithm is guaranteed to converge to a minimum state DFA corresponding to the target regular grammar. We prove the convergence of this algorithm in the limit and analyze its time and space complexities.

Keywords
grammar inference, regular grammars, finite state automata, language learning, active learning, incremental algorithms, machine learning

Disciplines
Artificial Intelligence and Robotics | Theory and Algorithms

A Polynomial Time Incremental Algorithm for Regular Grammar Inference

TR #97-03

Rajesh Parekh, Codrin Nichitiu, and Vasant Honavar

January 17, 1997

ACM Computing Classification System Categories (1991):
I.2.6 [*Artificial Intelligence*] Learning — language acquisition, concept learning; F.1.1 [*Theory of Computation*] Models of Computation — Automata.

Keywords:
grammar inference, regular grammars, finite state automata, language learning, active learning, incremental algorithms, machine learning.

Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa Sate University
Ames, Iowa 50011-1040, USA

A Polynomial Time Incremental Algorithm for Regular Grammar Inference

Rajesh Parekh
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040, U.S.A.
parekh@cs.iastate.edu

Codrin Nichitiu[*]
Ecole Normale Superieure de Lyon
46 Allee d'Italie
69364 LYON Cedex 07, France.
codrin@ens-lyon.fr

Vasant Honavar[†]
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040, U.S.A.
honavar@cs.iastate.edu

January 17, 1997

[*]Codrin Nichitiu's visit to Iowa State University during the summer of 1996 was made possible by financial support from Ecole Normale Superieure de Lyon.

[†]Vasant Honavar's research was partially supported by the National Science Foundation (through grants IRI-9409580 and IRI-9643299) and the John Deere Foundation.

Abstract

We present an efficient incremental algorithm for learning regular grammars from labeled examples and membership queries. This algorithm is an extension of Angluin's *ID* procedure to an incremental framework. The learning algorithm is intermittently provided with labeled examples and has access to a knowledgeable teacher capable of answering membership queries. Based on the observed examples and the teacher's responses to membership queries, the learner constructs a deterministic finite automaton (DFA) with which all examples observed thus far are consistent. When additional examples are observed, the learner modifies this DFA suitably to encompass the information provided by the new examples. In the limit this algorithm is guaranteed to converge to a minimum state DFA corresponding to the target regular grammar. We prove the convergence of this algorithm in the limit and analyze its time and space complexities.

1 Introduction

Grammar Inference [Biermann & Feldman, 1972; Fu & Booth, 1975; Miclet & Quinqueton, 1986; Langley, 1995] is an important machine learning problem with several applications in pattern recognition, language acquisition, and intelligent agent design. It is defined as the process of learning an unknown grammar given a finite set of labeled examples. *Regular grammars* describe languages that can be generated (and recognized) by deterministic finite state automata (DFA). Since they represent a widely used subset of formal languages, considerable research has been focused on regular grammar inference (or equivalently, identification of the corresponding DFA). However, given a finite set of positive examples and a finite, possibly empty set of negative examples the problem of learning a minimum state DFA equivalent to the unknown target is *NP*-hard [Gold, 1978]. The learner's task is simplified, by requiring that the examples provided meet certain desired criteria (like *structural completeness* [Pao & Carr, 1978; Parekh & Honavar, 1996] or *characteristic sample* [Oncina & García, 1992]), or by providing the learner with access to sources of additional information, like a knowledgeable teacher who responds to queries generated by the learner, etc. The interested reader is referred to [Miclet & Quinqueton, 1986; Pitt, 1989; Langley, 1995; Parekh & Honavar, 1997] for recent surveys of different approaches to grammar inference.

In many practical learning scenarios, a live complete set of examples may not be available at the outset. Instead, a sequence of examples is provided intermittently and the learner is required to construct an approximation of the target DFA based on the examples and the queries answered by the teacher. In such scenarios, an on-line or incremental model of learning that is guaranteed to eventually converge to the target DFA in the limit is of interest. Particularly, in the case of intelligent autonomous agents, incremental learning offers an attractive framework for characterizing the behavior of the agents [Carmel & Markovitch, 1996]. Against this background, we present a provably correct, polynomial time, incremental, interactive algorithm for regular grammar inference that learns from *labeled examples* and *membership queries*. The proposed algorithm (*IID*) extends Angluin's *ID* algorithm [Angluin, 1981] to an incremental setting.

2 Preliminaries

Let Σ be a finite set of symbols called the *alphabet*. Σ^* denotes the set of strings over the alphabet. α, β, γ will be used to denote strings in Σ^*. $|\alpha|$ denotes the length of the string α. λ is a special string called the *null* string and has length 0. Given a string $\alpha = \beta\gamma$, β is the *prefix* of α and γ is the *suffix* of α. Let $Pr(\alpha)$ denote the set of all prefixes of α. Given two sets S_1 and S_2, the *set difference* is denoted by $S_1 \backslash S_2$ and the *symmetric difference* is denoted by $S_1 \oplus S_2$.

A *Regular Grammar* (G) is a finite set of rewrite (production) 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*. 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. *Finite State Automata* (FSA) are recognizers for regular grammars. A *deterministic* FSA (DFA), 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. The extension of δ to handle input strings is denoted by δ^* and it maps $Q \times \Sigma^* \longrightarrow Q$. By definition, $\delta^*(q, \lambda) = q \; \forall q \in Q$ and $\delta^*(q, b\alpha) = \delta^*(\delta(q, b), \alpha)$. The set of all strings accepted by A is its language, $L(A)$. $L(A) = \{\alpha | \delta^*(q_0, \alpha) \in F\}$. Note that $L(A) = L(G)$ if the DFA A is an acceptor for the regular grammar G. Fig. 1 shows the state transition diagram for a sample FSA.

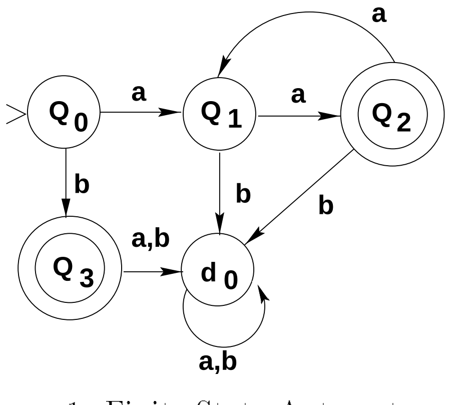

Figure 1: Finite State Automaton.

Given a regular language $L(G)$ there exists a DFA A with minimum number of states that recognizes $L(G)$. We call this minimum state acceptor for a regular language the *canonical DFA* and denote it as $A(L(G))$. Let N denote the number of states of $A(L(G))$.

Unless otherwise stated, we will now assume that A denotes the canonical DFA for a regular language $L(G)$.

Given a canonical DFA A for a regular language $L(G)$, a *labeled example* $(\alpha, c(\alpha))$ is a 2-tuple with $\alpha \in \Sigma^*$ and the *classification function* $c : \Sigma^* \longrightarrow \{+, -\}$ is defined as follows: $c(\alpha) = +$ if $\alpha \in L(G)$ and $c(\alpha) = -$ if $\alpha \notin L(G)$. A *membership query* is a query of the form "$\alpha \in L(G)$?". A teacher's response to the membership query is either *yes* or *no* depending on whether $c(\alpha) = +$ or $-$ respectively.

A state q of A is *live* if there exist strings α and β such that $\alpha\beta \in L(A)$, $\delta^*(q_0, \alpha) = q$, and $\delta^*(q, \beta) \in F$. A state that is not live is called *dead*. It can be verified that the canonical DFA for any regular language can have at most one dead state. We use d_0 to denote this dead state. Given A, a finite set of strings P is said to be *live-complete* if for every live state q of A there exists a string $\alpha \in P$ such that $\delta^*(q_0, \alpha) = q$. For example, $P = \{\lambda, a, b, aa\}$ is a live-complete set for the DFA in Fig. 1. A live-complete set represents all the live states of A. We will assume that the string λ is part of any live-complete set to represent the start state of A. The set $P' = P \cup \{d_0\}$ represents all states of A. To account for the state transitions, define a function $f : P' \times \Sigma \longrightarrow \Sigma^* \cup \{d_0\}$ as follows:

$$\begin{aligned} f(d_0, b) &= d_0 \\ f(\alpha, b) &= \alpha b \end{aligned}$$

Note that $f(\alpha, b)$ denotes the state reached on reading an input letter b from the state represented by the string α. Let $T' = P' \cup \{f(\alpha, b) | (\alpha, b) \in P' \times \Sigma\}$ and $T = T' \backslash \{d_0\}$.

3 The *ID* Algorithm

The *ID* algorithm for inference of regular grammars, its correctness proof, and complexity analysis are covered in detail in [Angluin, 1981]. Since the proposed incremental algorithm for grammar inference *IID*, extends *ID* to an incremental setting, to keep the discussion that follows self-contained, we review *ID* briefly in this section.

Given a canonical DFA A for a regular language and a live-complete set P for A, *ID* constructs a partition of the set T' (constructed from P as described above) such that elements of T' representing the same state of A are grouped together in the same block of the partition. In the process a set of strings V is constructed such that no two distinct states of A have the same behavior on all strings in V. When the set V has i elements, define function $E_i : T' \longrightarrow 2^V$ as follows:

$$\begin{aligned} E_i(d_0) &= \phi \\ E_i(\alpha) &= \{v_j | v_j \in V, 0 \le j \le i, \alpha v_j \in L(A)\} \end{aligned}$$

Algorithm

Input: A live complete set P and a teacher to answer membership queries.

Output: A description of the canonical DFA for the target regular grammar.

```
begin
  1)  $i = 0$, $v_i = \lambda$, $V = \{\lambda\}$, $T = P \cup \{f(\alpha, b) \mid (\alpha, b) \in P \times \Sigma\}$ and $T' = T \cup \{d_0\}$
  2)  – $E(d_0) = \phi$
      – $\forall \alpha \in T$, pose the membership query "$\alpha \in L(A)$?".
          if the teacher's response is yes
          then $E_0(\alpha) = \{\lambda\}$
          else $E_0(\alpha) = \phi$
  3)  while $\exists \alpha, \beta \in P$ and $b \in \Sigma$ such that
              $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \neq E_i(f(\beta, b))$
      do
          – Let $\gamma \in E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$
          – Let $v_{i+1} = b\gamma$
          – Set $V = V \cup \{v_{i+1}\}$ and $i = i + 1$
          – $\forall \alpha \in T$, pose the membership query "$\alpha v_i \in L(A)$?".
              if the teacher's response is yes
              then $E_i(\alpha) = E_{i-1}(\alpha) \cup \{v_i\}$
              else $E_i(\alpha) = E_{i-1}(\alpha)$
      end while
  4)  Output $M$ computed as follows and STOP.
      – The states of $M$ are the sets $E_i(\alpha)$, where $\alpha \in T$
      – The initial state $q_0$ is the set $E_i(\lambda)$
      – The accepting states are the sets $E_i(\alpha)$ where $\alpha \in T$ and $\lambda \in E_i(\alpha)$
      – if $E_i(\alpha) = \phi$
        then add self loops on the state $E_i(\alpha)$ for all $b \in \Sigma$
        else $\forall \alpha \in P$ and $\forall b \in \Sigma$, set the transition $\delta(E_i(\alpha), b) = E_i(f(\alpha, b))$
end
```

The following example demonstrates the working of the ID algorithm. Consider the DFA in Fig. 1 for which $P = \{\lambda, a, b, aa\}$ is a live-complete set. $T' = \{d_0, \lambda, a, b, aa, ab, ba, bb, aaa, aab\}$. Table 1 shows the computation of $E_i(\alpha)$ for strings $\alpha \in T'$.
Note that the DFA returned by the procedure is exactly the DFA in Fig. 1. Angluin [Angluin, 1981] has shown that number of membership queries needed is $\mathcal{O}(|\Sigma| \cdot N \cdot |P|)$. Thus, *ID* runs in time polynomial in $|\Sigma|$, N, and $|P|$.

4 *IID* – An Incremental Extension of *ID*

We present the algorithm *IID* which extends *ID* to an incremental setting. Our learning model assumes the availability of a source of labeled examples and a teacher capable of answering membership queries posed by the learner. However, unlike in the case of *ID*,

i	0	1	2	3
v_i	λ	b	a	aa
$E(d_0)$	ϕ	ϕ	ϕ	ϕ
$E(\lambda)$	ϕ	$\{b\}$	$\{b\}$	$\{b, aa\}$
$E(a)$	ϕ	ϕ	$\{a\}$	$\{a\}$
$E(b)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$
$E(aa)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda, aa\}$
$E(ab)$	ϕ	ϕ	ϕ	ϕ
$E(ba)$	ϕ	ϕ	ϕ	ϕ
$E(bb)$	ϕ	ϕ	ϕ	ϕ
$E(aaa)$	ϕ	ϕ	$\{a\}$	$\{a\}$
$E(aab)$	ϕ	ϕ	ϕ	ϕ

Table 1: Execution of ID.

a live-complete set of examples is not provided to the learner at the outset. Instead, the learner is required to gradually refine its model of the target DFA as new examples become available. Let M_t denote the DFA that corresponds to the learner's current model after observing t examples. Initially, M_0 is a *null* automaton with only one state (the dead state) and it rejects every string in Σ^*. Clearly, every negative example encountered by the learner at this point is consistent with M_0. Without loss of generality we assume that the first example seen by the learner is a positive example. When the first positive example, α, is observed M_0 is modified to accept the positive example. With each additional observed example, α, it is determined whether α is consistent with M_t. i.e., $c(\alpha) = +$ and $\alpha \in L(A)$ or $c(\alpha) = -$ and $\alpha \notin L(A)$ in which case $M_{t+1} = M_t$. Otherwise M_t is suitably modified such that α is consistent with the resulting DFA, M_{t+1}. A detailed description of the algorithm follows:

Algorithm

Input: A stream of labeled examples and a teacher to answer membership queries.
Output: A DFA M_t with which all t examples observed by the learner are consistent.

```
begin

    1)  $i = 0$, $k = 0$, $t = 0$, $P_k = \phi$, $T_k = \phi$, $V = \phi$.
        Initialize $M_t$ to the null DFA.

    2)  Wait for a positive example $(\alpha, +)$
        – $P_0 = Pr(\alpha)$ and $P_0^{'} = P_0 \cup \{d_0\}$
        – $T_0 = P_0 \cup \{f(\alpha, b) | (\alpha, b) \in P_0 \times \Sigma\}$ and $T_0^{'} = T_0 \cup \{d_0\}$
        – $v_0 = \lambda$ and $V = \{v_0\}$
```

```
      – $E_0(d_0) = \phi$
      – $\forall \alpha \in T_0$, pose the membership query "$\alpha \in L(A)$?".
              **if** the teacher's response is *yes*
              **then** $E_0(\alpha) = \{\lambda\}$
              **else** $E_0(\alpha) = \phi$

3)    **while** $\exists \alpha, \beta \in P_k$ and $b \in \Sigma$ such that
              $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \neq E_i(f(\beta, b))$
      **do**
          – Let $\gamma \in E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$
          – Let $v_{i+1} = b\gamma$
          – Set $V = V \cup \{v_{i+1}\}$ and $i = i + 1$
          – $\forall \alpha \in T_k$, pose the membership query "$\alpha v_i \in L(A)$?".
                  **if** the teacher's response is *yes*
                  **then** $E_i(\alpha) = E_{i-1}(\alpha) \cup \{v_i\}$
                  **else** $E_i(\alpha) = E_{i-1}(\alpha)$
      **end while**

4)    Set $t = t + 1$ and construct $M_t$ as follows:
      – The states of $M_t$ are the sets $E_i(\alpha)$, where $\alpha \in T_k$
      – The initial state $q_0$ is the set $E_i(\lambda)$
      – The accepting states are the sets $E_i(\alpha)$ where $\alpha \in T_k$ and $\lambda \in E_i(\alpha)$
      – **if** $E_i(\alpha) = \phi$
        **then** add self loops on the state $E_i(\alpha)$ for all $b \in \Sigma$
        **else** $\forall \alpha \in P$ and $\forall b \in \Sigma$, set the transition $\delta(E_i(\alpha), b) = E_i(f(\alpha, b))$

5)    Wait for a new example $(\alpha, c(\alpha))$
      **if** $\alpha$ is consistent with $M_t$
      **then**
          – $M_{t+1} = M_t$
          – $t = t + 1$
          – **goto** step 5
      **else**
          – $P_{k+1} = P_k \cup Pr(\alpha)$ and $P'_{k+1} = P_{k+1} \cup \{d_0\}$
          – $T_{k+1} = T_k \cup \{f(\alpha, b) | (\alpha, b) \in \{P_{k+1} \backslash P_k\} \times \Sigma\}$ and $T'_{k+1} = T_{k+1} \cup \{d_0\}$
          – $\forall \alpha \in T_{k+1} \backslash T_k$, fill in the entries for $E_i(\alpha)$ by posing membership queries:
                  $E_i(\alpha) = \{v_j | 0 \leq j \leq i, \alpha v_j \in L(A)\}$
          – $k = k + 1$
          – **goto** step 3
**end**
```

4.1 Example

We show a sample run of the incremental algorithm for the DFA of Fig. 1.

The learner starts with a model M_0 equivalent to the *null* DFA accepting no strings. Suppose the learner encounters the example $(b, +)$. This causes the learner to take the following actions:

- $P_0 = \{\lambda, b\}$ and $P_0' = \{d_0, \lambda, b\}$
- $T_0 = \{\lambda, a, b, ba, bb\}$ and $T_0' = \{d_0, \lambda, a, b, ba, bb\}$
- The computation of the functions E_i is shown in Table 2.

i	0	1
v_i	λ	b
$E(d_0)$	ϕ	ϕ
$E(\lambda)$	ϕ	$\{b\}$
$E(a)$	ϕ	ϕ
$E(b)$	$\{\lambda\}$	$\{\lambda\}$
$E(ba)$	ϕ	ϕ
$E(bb)$	ϕ	ϕ

Table 2: Execution of the incremental version of *ID* ($k = 0$).

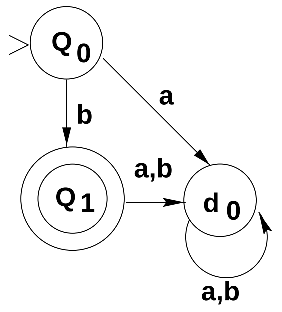

Figure 2: The intermediate DFA M_1 in the incremental procedure.

At this point the learner constructs a model (M_1) of the target DFA (Fig. 2).
Suppose the next example observed by the learner is $(a, -)$. Since, M_1 correctly rejects a, $M_2 = M_1$ and the learner waits for additional examples.

Assume that the learner observes $(aa, +)$ next. Since $aa \notin L(M_2)$ the learner takes the following steps to update M_2:

- $P_1 = \{\lambda, a, b, aa\}$ and $P_1' = \{d_0, \lambda, a, b, aa\}$

- $T_1 = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab\}$ and $T_1' = \{d_0, \lambda, a, b, aa, ab, ba, bb, aaa, aab\}$
- The function E_1 is extended to cover the new elements belonging to $T_1 \backslash T_0$. The resulting computation of the various E_i's is depicted in Table 3.

i	1	2	3
v_i	b	a	aa
$E(d_0)$	ϕ	ϕ	ϕ
$E(\lambda)$	$\{b\}$	$\{b\}$	$\{b, aa\}$
$E(a)$	ϕ	$\{a\}$	$\{a\}$
$E(b)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$
$E(ba)$	ϕ	ϕ	ϕ
$E(bb)$	ϕ	ϕ	ϕ
$E(aa)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda, aa\}$
$E(ab)$	ϕ	ϕ	ϕ
$E(aaa)$	ϕ	$\{a\}$	$\{a\}$
$E(aab)$	ϕ	ϕ	ϕ

Table 3: Execution of the incremental version of *ID* ($k = 1$).

This yield's a revised model (M_3) of the target DFA. It is easy to see that this is exactly the DFA we are trying to learn (Fig. 1). Note also that the set P_1 is live-complete with respect to the target DFA.

4.2 Correctness Proof

Theorem 1:
IID converges correctly to the canonical representation of the target DFA when the set P_k encompasses a live complete set (P_l) for the target.

Proof:
Consider an execution of *ID* given the live-complete set P_l. First we demonstrate that the execution of *ID* can be made to track that of *IID* in that the set V generated by both is the same and hence the values for $E_i(\alpha)$ $\forall \alpha \in T_l$ are the same for both executions. We prove this claim by induction.

Base Case:
Both *ID* and *IID* start with $v_0 = \lambda$.

At $k = 0$, *IID* has the set P_0 available to it. $P_0 \subseteq P_l$. Clearly, for all strings $\alpha, \beta \in P_0$ such that $E_0(\alpha) = E_0(\beta)$ but $E_0(f(\alpha, b)) \neq E_0(f(\beta, b))$ in the case of *IID* it is also the case that the same strings $\alpha, \beta \in P_l$ for *ID* such that $E_0(\alpha) = E_0(\beta)$ but $E_0(f(\alpha, b)) \neq E_0(f(\beta, b))$.

Assume that one such pair α, β is selected by both *ID* and *IID*. The string $\gamma \in E_0(f(\alpha, b)) \oplus E_0(f(\beta, b))$ can only be λ. Thus, the string $v_1 = b\gamma$ is the same for both executions.

Induction Hypothesis:
Assume that after observing some t examples, at some value of k $(0 \leq k < l)$, when $P_k \subseteq P_l$ is available to *IID*, the sequence of strings $v_0, v_1, \ldots, v_i$ and the values $E_i(\alpha)\ \forall \alpha \in T_k$ are the same for both executions.

Induction Proof:
We now show that the same string v_{i+1} is the generated by both *ID* and *IID*. Following the reasoning presented in the base case, and the induction hypothesis we can state that for all strings $\alpha, \beta \in P_k$ such that $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \neq E_i(f(\beta, b))$ in the case of *IID* it is also the case that the same strings $\alpha, \beta \in P_l$ for *ID* such that $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \neq E_i(f(\beta, b))$.

Assume that one such pair α, β is selected by both executions. Then $E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$ is the identical for both. Thus given that the same string $\gamma \in E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$ is selected, the string $v_{i+1} = b\gamma$ is identical for both executions.

Thus, we have shown that after observing some t examples when the live-complete set P_l is available to *IID* the values of $E_i(\alpha)\ \forall \alpha \in T_l$ are exactly the same as the corresponding values of $E_i(\alpha)$ for *ID*.

[Angluin, 1981] has shown that the DFA M returned by *ID* is isomorphic to the canonical DFA A for the target whenever *ID* is provided with a live-complete set. From above we know that at $k = l$ the current model (M_t) of the target automaton maintained by *IID* is identical to one that arrived at by *ID*. Thus, we have proved that the incremental version correctly converges to a canonical representation of the target DFA. $\square$

Theorem 2:
At any time during the execution of the learning algorithm, all the (t) examples observed by the learner are consistent with the current representation of the target (M_t).

Proof:
Suppose M_t is modified after observing an example α not consistent with it. Upon observing α, P_k is modified to include α and all its prefixes. Since T_{k+1} contains α and we are assuming that the teacher correctly answers membership queries, we know that the function E_j from which M_{t+1} is constructed is such that $E_j(\alpha)$ contains λ if $c(\alpha) = +$ or does not contain λ if $c(\alpha) = -$. Thus, α is consistent with M_{t+1}.

We now show that all strings α consistent with M_t are also consistent with M_{t+1}.

By construction, the states of M_t are represented by $E_i(\beta)$ $\forall\beta \in T_k$, and for any state q of M_t that is represented by $E_i(\beta)$, $\delta^*(q_0, \beta) = q$. Thus for any string α such that $\delta^*(q_0, \alpha) = q$, it is clear that there is a corresponding string $\beta \in T_k$ such that $\delta^*(q_0, \alpha) = \delta^*(q_0, \beta)$. Further, if α is consistent with M_t, we know that $\lambda \in E_i(\beta)$ if $c(\alpha) = +$ or $\lambda \notin E_i(\beta)$ if $c(\alpha) = -$. We prove that for all strings α that are consistent with M_t, there exists a string $\gamma \in T_{k+1}$ such that $\delta^*(q_0, \alpha) = \delta^*(q_0, \gamma)$ (in M_{t+1}) and for some $j \geq i$, $\lambda \in E_j(\gamma)$ if $c(\alpha) = +$ or $\lambda \notin E_j(\gamma)$ if $c(\alpha) = -$.

Consider the *ID* algorithm. The set T is partitioned such that all elements representing the same state are grouped together in the same block of the partition. The initial partition (corresponding to $v_0 = \lambda$) separates the elements of T into accepting and non-accepting states. Further refinements of the partition correspond to the strings v_i that tell apart elements that do not represent the same state. These refinements involve splitting individual blocks of the current partition. The new blocks (obtained by splitting an existing block) differ from each other only in terms of the string v_i. Since $v_0 = \lambda$ and $v_i \neq \lambda$ ($\forall i > 0$), we see that refinements of a block containing the string λ will still contain λ and refinements of a block not containing the string λ will not have λ as part of them now.

The situation is similar for *IID*. Consider a string $(\alpha, c(\alpha))$ that is consistent with M_t. This means that $\exists\beta \in T_k$ such that $\delta^*(q_0, \alpha) = \delta^*(q_0, \beta)$ (in M_t) and $\lambda \in E_i(\beta)$ if $c(\alpha) = +$ or $\lambda \notin E_i(\beta)$ if $c(\alpha) = -$. After seeing an example that is not consistent with M_t, the learning algorithm modifies T_k to T_{k+1} and computes E_i for the strings belonging to $T_{k+1} \backslash T_k$. The extended E_i represents a partition of T_{k+1} that has the same number of blocks as the partition of T_k but with each block possibly having more elements now (since $T_k \subseteq T_{k+1}$). Assume that strings v_{i+1}, v_{i+2}, ..., v_j are generated and the functions E_{i+1}, E_{i+2}, $\ldots$, E_j are computed thereby progressively refining the partition of T_{k+1} represented by E_i. The block represented by $E_i(\beta)$ might be split and the state $\delta^*(q_0, \alpha)$ will now correspond to one of these newly created blocks (say the one represented by $E_j(\gamma)$). By following the argument for the *ID* we know that all the blocks created by splitting the block represented $E_i(\beta)$ will contain (or not contain) λ depending on whether $E_i(\beta)$ contained (or did not contain) λ. We have thus proved that there exists a string $\gamma \in T_{k+1}$ such that $\delta^*(q_0, \alpha) = \delta^*(q_0, \gamma)$ (in M_{t+1}) and for some $j \geq i$, $\lambda \in E_j(\gamma)$ if $c(\alpha) = +$ or $\lambda \notin E_j(\gamma)$ if $c(\alpha) = -$. i.e., all strings α that were consistent with M_t are also consistent with M_{t+1}. $\square$

4.3 Complexity Analysis

Assume that at some $k = l$ the set P_l encompasses a live-complete for the target DFA A. We have shown in the correctness proof of the incremental algorithm that at this point the algorithm's current representation of the target would be isomorphic to A.

The size of T_l is at most $|\Sigma| \cdot |P_l| + 1$. Also, the size of V is no more than N (the number of states of A). Thus the total number of membership queries posed by the learner is $\mathcal{O}(|\Sigma| \cdot |P_l| \cdot N)$. Searching for a pair of strings α, β to distinguish two states in the current representation of the target takes time that is $\mathcal{O}(T_l^2)$. Thus, the incremental algorithm runs in time polynomial N, $|\Sigma|$, and $|P_l|$.

Since the size of T_l is at most $|\Sigma| \cdot |P_l| + 1$, the size of V is no more than N, and the function E_i is replaced by the function E_{i+1} the space complexity of the algorithm is $\mathcal{O}(|\Sigma| \cdot |P_l| \cdot N)$.

5 Discussion

Efficient and provably convergent algorithms for grammar inference find applications in a broad range of problems in artificial intelligence, pattern recognition, and intelligent agent architectures. The proposed incremental algorithm for regular grammar inference, IID, is guaranteed to converge to an unknown target grammar and has polynomial time and space complexities. This extends previous work of Angluin [Angluin, 1981] to an incremental setting.

In related work, Parekh and Honavar have proposed a provably correct algorithm for regular grammar inference based on the *version space* approach for searching the candidate space of finite state automata [Parekh & Honavar, 1996]. Their approach searches a lattice of FSA generated by successive state mergings of a *prefix tree automaton* (PTA) which is essentially a lookup table for a set of positive examples of the target grammar. The lattice is represented compactly as a version space and is guaranteed to contain the target FSA (provided the set of positive examples is *structurally complete*). A candidate elimination algorithm which updates the version space based on the teacher's responses to membership queries is guaranteed to eventually converge to the target grammar. In the incremental framework they assume that the positive examples (needed to construct a *structurally complete* set) are provided intermittently. The algorithm augments the version space as needed in response to these new positive examples. Assuming that the examples are provided in increasing order by length, convergence to the target FSA is guaranteed when a structurally complete set of positive examples has been processed. Though provably correct, this algorithm has practical limitations because the size of the lattice grows exponentially with the number of states of the PTA.

Dupont has proposed an incremental version of the *RPNI* algorithm [Oncina & García, 1992] for regular grammar inference [Dupont, 1996]. This algorithm is also based on the idea of a lattice of partitions of the states of a PTA for a set of positive examples. It uses information from a set of negative examples to guide the ordered search through the lattice and is guaranteed to converge to the target DFA when the set of examples seen by the learner include a *characteristic sample* for the target automaton as a subset. The algorithm runs in time that is polynomial in the sum of lengths of the positive and negative examples. However, it requires storage of all the examples seen by the learner to ensure that each time the representation of the target is modified, it stays consistent with all examples seen previously.

Porat and Feldman [Porat & Feldman, 1991] have proposed an incremental algorithm for inference of regular grammars from a *complete ordered sample*. The algorithm is an iterative strategy that works with the current encoding of the examples seen thus far and modifies it suitably if this encoding is not consistent with the next labeled example. Their algorithm is also guaranteed to converge in the limit provided the examples appear in strict lexicographic order. The algorithm works with only a finite working storage which is an advantage over the incremental extension of the *RPNI* algorithm. Besides requiring an ordered presentation of the examples, this algorithm also requires a consistency check with all the previous examples when the current representation of the target is modified by the algorithm.

Our framework for incrementally learning a regular grammar from labeled examples does not require storage of all the examples. Only those examples that are inconsistent with the current representation of the target are required to be stored (implicitly) by the learner. The algorithm does not require any specific ordering of the labeled examples. Furthermore, the incremental modification of learner's representation of the target DFA is guaranteed to be consistent with all the examples processed by the learner at any stage during learning and no explicit consistency check is needed. The reader should note that like the *ID* algorithm, this incremental version also avails of a knowledgeable teacher capable of answering membership queries.

Regular grammars can be used to capture the behavior of *intelligent agents* like robots navigating in a finite world. Incremental regular grammar inference can provide a framework for these agents to learn from experience in an unfamiliar environment. Given its efficiency and guaranteed consistency with all examples seen thus far, our algorithm can provide an effective tool for agent learning, especially in an interactive setting.

The learner's reliance on the teacher to provide accurate responses to membership queries poses a potential limitation in applications where a reliable teacher is not available. We are exploring the possibility of learning in an environment where the learner does not have access to a teacher. The algorithms due to Dupont [Dupont, 1996] and Porat & Feldman [Porat & Feldman, 1991] operate in this framework. Some open problems include whether the limitations of these algorithms (e.g., need for storage of all the previously seen examples and complete lexicographic ordering of examples) can be overcome without sacrificing efficiency and guaranteed convergence to the target. In this context, it is of interest to explore alternative models of learning that: relax the convergence criterion (for example, allow approximate learning of the target within a given error bound); provide for some additional hints to the learning algorithm (like a bound on the number of states of the target DFA); include a helpful teacher that carefully guides the learner (perhaps by providing *simple* examples first).

References

Angluin, D. (1981). A Note on the Number of Queries Needed to Inentify Regular

Languages. *Information and Control*, **51**, 76–87.

Biermann, A., & Feldman, J. (1972). A Survey of Results in Grammatical Inference. *Pages 31–54 of:* Watanabe, S. (ed), *Frontiers of Pattern Recognition.* Academic Press.

Carmel, D., & Markovitch, S. (1996). Learning Models of Intelligent Agents. *Pages 62–67 of: Proceedings of the AAAI-96 (vol. 1).* AAAI Press/MIT Press.

Dupont, P. (1996). Incremental Regular Inference. *Pages 222–237 of:* Miclet, L., & Higuera, C. (eds), *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147.* Montpellier, France: Springer.

Fu, K. S., & Booth, T. L. (1975). Grammatical Inference: Introduction and Survey (part 1). *IEEE Transactions on Systems, Man and Cybernetics*, **5**, 85–111.

Gold, E. M. (1978). Complexity of Automaton Identification from Given Data. *Information and Control*, **37**(3), 302–320.

Langley, P. (1995). *Elements of Machine Learning.* Palo Alto, CA: Morgan Kauffman.

Miclet, L., & Quinqueton, J. (1986). Learning from Examples in Sequences and Grammatical Inference. *Pages 153–171 of:* Ferrate, G. *et al* (ed), *Syntactic and Structural Pattern Recognition.* NATO ASI Series Vol. F45.

Oncina, J., & García, P. (1992). Inferring Regular Languages in Polynomial Update Time. *Pages 49–61 of:* Pérez, N. *et al* (ed), *Pattern Recognition and Image Analysis.* World Scientific.

Pao, T., & Carr, J. (1978). A Solution of the Syntactic Induction-Inference Problem for Regular Languages. *Computer Languages*, **3**, 53–64.

Parekh, R. G., & Honavar, V. G. (1996). An Incremental Interactive Algorithm for Regular Grammar Inference. *Pages 238–250 of:* Miclet, L., & Higuera, C. (eds), *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147.* Montpellier, France: Springer.

Parekh, R. G., & Honavar, V. G. (1997). Automata Induction, Grammar Inference, and Language Acquisition. *In:* Moisl, H. *et al* (ed), *Handbook of Natural Language Processing.* Marcel Dekker. To appear.

Pitt, L. (1989). Inductive Inference, DFAs and Computational Complexity. *Pages 18–44 of: Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence 397.* Springer-Verlag.

Porat, S., & Feldman, J. (1991). Learning Automata from Ordered Examples. *Machine Learning*, **7**, 109–138.