

# **A Survey on Inductive Learning**

**A RESEARCH PROFICIENCY EXAM PRESENTED  
BY**

**JALAL MAHMUD  
DEPARTMENT OF COMPUTER SCIENCE  
STONY BROOK UNIVERSITY**

**Dec 21, 2005**

Abstract of the RPE  
**A Survey on Inductive Learning**  
by  
Jalal Mahmud  
Stony Brook University  
Dec 21, 2005

Inductive learning is the method of learning from observations. Inductive learning has important applications over a wide range of area including pattern recognition, language acquisition, bio-informatics and intelligent agent design. Because of such diverse applicability, inductive learning methods including automata learning, grammar induction, hidden markov model learning and symbolic statistical modeling have attracted the attention of researchers for several decades. Consequently, inductive learning continues to be a major focus of research. This report surveys some of the key results in automata learning, grammar induction, hidden markov model learning and symbolic statistical modeling. Different learning problems are critically compared to emphasize their usability for various applications. Learning from example sequences has been illustrated by implementing automata learning techniques to learn a process model to capture users browsing behavior in Web transactions. Experimental evaluation of the learned model is also reported. Possible future research is to incorporate grammar induction and symbolic statistical learning in rule based systems.

# Contents

<b>1 Introduction</b>	<b>1</b>
<b>2 Automata Learning</b>	<b>1</b>
2.1 Automata Preliminaries.....	1
2.2 Regular Grammar Induction and Automata Learning .....	4
2.3 Complexity Result for Automata Learning .....	4
2.4 Training Pattern for Automata Learning.....	5
2.5 Active and Passive Learning.....	5
<b>3. Non Probabilistic Automata Learning</b>	<b>5</b>
3.1 Learning with a Teacher: Active Learning .....	5
3.2 Learning without Teacher: Passive Learning .....	6
<b>4. Probabilistic Automata Learning</b>	<b>8</b>
4.1 Stochastic Automata Learning .....	8
4.2 Hidden Markov Model Learning .....	8
4.3 Bayesian Model Merging .....	10
<b>5. Symbolic Statistical Modeling</b>	<b>13</b>
5.1 Statistical Learning Preliminaries .....	13
5.2 Logic Programming Preliminaries .....	14
5.3 Parameter Learning of Logic Programs .....	15
5.4 Symbolic Statistical Modeling Language (PRISM) .....	15
5.5 Graphical EM Algorithm .....	15
5.6 Advantage of Symbolic Statistical Modeling .....	17
5.7 Future Directions of Symbolic Statistical Modeling .....	17
<b>6. Preliminary Work</b>	<b>17</b>
6.1 Process Model as Automaton .....	17
6.2 Learning from Web Transactions.....	19
6.3 Evaluation of the Model .....	20
<b>7. Discussion and Future Research</b>	<b>21</b>

# List of Figures

2.1	An Example Finite Automation .....	2
2.2	State Merging Example .....	3
2.3	Grammar Induction from Examples .....	4
2.4	A DFA which has structurally complete set $S^+ = \{b,aa,aaaa\}$ .....	5
3.1.	Search Space for DFA learning .....	6
3.2.	A Prefix Tree Automata and Merged Automata.....	7
4.1	A Stochastic Prefix Tree Automata (PTA) .....	8
4.2	An Example Hidden Markov Model .....	9
4.3	Model Merging for HMM .....	12
5.1.	Example of a Support Graph .....	16
6.1.	Learned Process Model for Web Transactions .....	18
6.2.	A Prefix Tree Automata and Learned DFA .....	19
6.3	Recall/Precision Characteristics for Learned Process Model .....	20
6.4	Failure Analysis of the Learned Model .....	21

# 1. Introduction

Inductive learning is a particular instance of machine learning. Its goal is to find general law from examples. This is a sub problem of theoretical computer science, artificial intelligence or pattern recognition. Grammatical inference is a special case of inductive learning which aims to construct models of some underlying system based on sets of positive and negative classifications. This problem can also be considered as a special instance of the problem of system identification where some target system is inferred based on input/output data. It is an important machine learning problem with several applications in pattern recognition, language acquisition, bioinformatics and intelligent agent design. Examples of grammar induction can be speech, chronological series, successive actions of a Web user, successive moves during a chess game. Grammatical inference or DFA induction from finite set of labeled examples has substantial application in several domains including syntactic pattern recognition, intelligent autonomous agents and language acquisition. We can also consider the case of a robot navigating in an unknown environment and attempting to construct an accurate map of that environment. From the example moves of the robot, it is possible to learn the rules for robot navigation. Research in instructional robots and conversational interfaces is based on the design of learning agents, which can be learned by inductive inference techniques. Above example applications motivate the study and comparative analysis of different aspects of automata induction and grammar inference. This work not only presents a detailed survey of learning automation, grammar and Hidden Markov Model (HMM) from examples, but it also presents the implementation as well as experimental evaluation of automata learning techniques for the development of a process model for Web transactions.

This survey also addresses the problem of parameter learning of logic programs. There are significant attempts to incorporate parameter learning into computer programs. The reason is twofold. First, it is desirable to add the ability of learning to computer programs, which is believed as a necessary step toward building intelligent systems. Second, it is possible to model complex phenomena such as gene inheritance, consumer behavior, natural language processing by adding learning ability of computer programs.

Following section discusses basics of automata learning and grammar induction. In section 3 and 4, non-probabilistic and probabilistic automata learning methods are discussed. Symbolic statistical modeling with a focus on parameter learning for logic program has been discussed in section 5. Preliminary work on the implementation of automata learning for development of a process model from Web transactional sequences is described in section 6. Finally, conclusion is drawn in section 7.

## 2. Automata Learning

### 2.1 Automata Preliminaries

#### **Finite Automaton.**

Finite automaton is an ordered 5-tuple  $M = (Q, \Sigma, \delta, q_0, F)$ , where

- $Q$  is a finite set of states.

- $\Sigma$  is a finite set of acceptable input symbols.
- $\delta$  is a transition mapping of a set  $Q \times \Sigma$  into a set  $P(Q)$ ;  $\delta$  defines the behavior of the driving device.  $\delta$  is often called a transition function.
- $q_0 \in Q$  is an initial state of the driving device.
- $F \subseteq Q$  is a set of final states.

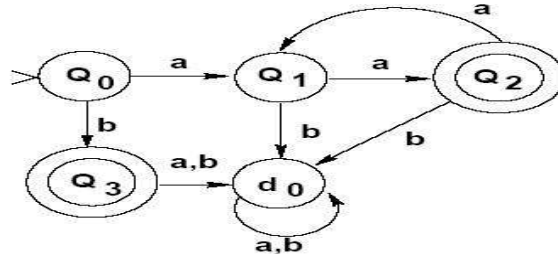


Fig 2.1: An Example Finite Automaton

### Deterministic Finite Automaton.

Automata is called *deterministic*, if  $\delta(q, a)$  contains no more than one state for any  $q$  and  $a$ .

### Non-Deterministic Finite Automaton.

Automata is called *non-deterministic* if  $\delta(q, a)$  contains more than one state for any  $q$  and  $a$ .

### Automaton Acceptance

A word is simply a finite sequence of symbols over some alphabet. Word  $w = a_1 \dots a_k$  over an alphabet  $\Sigma$  is accepted by the finite automaton  $M = (Q, \Sigma, \delta, q_0, F)$  if there exists a sequence  $q_1, q_2, \dots, q_n$  such that  $q_1 = q_0$ ,  $q_n \in F$  and  $\forall i \forall j: 1 \leq i < n, 1 \leq j < k \quad \delta(q_i, a_j) = q_{i+1}$ . A finite automaton *recognizes* a language  $L$  if the automaton accepts any word of the language  $L$ .

### State Equivalence

Two states  $q$  and  $q'$  in a DFA  $M = (Q, S, d, q_0, F)$  are said to be *equivalent* (or *indistinguishable*) if they generate the same strings over an alphabet  $\Sigma$ .

### Automaton Equivalence

Two finite automata,  $M_1 = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$  and  $M_2 = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$ , are called equivalent if they recognize the same language over an alphabet  $\Sigma$ .

### State Merging

In many applications of automata modeling, a critical problem in the use of deterministic finite state automata (DFA) is that the number of states is too large to implement many practical problems of a normal size. State merging is a classical idea for automata minimization [1,2]. Precondition of applying state merging operator is to have a method to construct an initial model from the data, a way to identify candidates for merging, merge states based on satisfaction of certain criteria and to have an error measure to compare goodness for the resultant merge. Each step of state merging generalizes the automaton,

because strings, which are not member of the training sets, may be generated as a result of state merging.

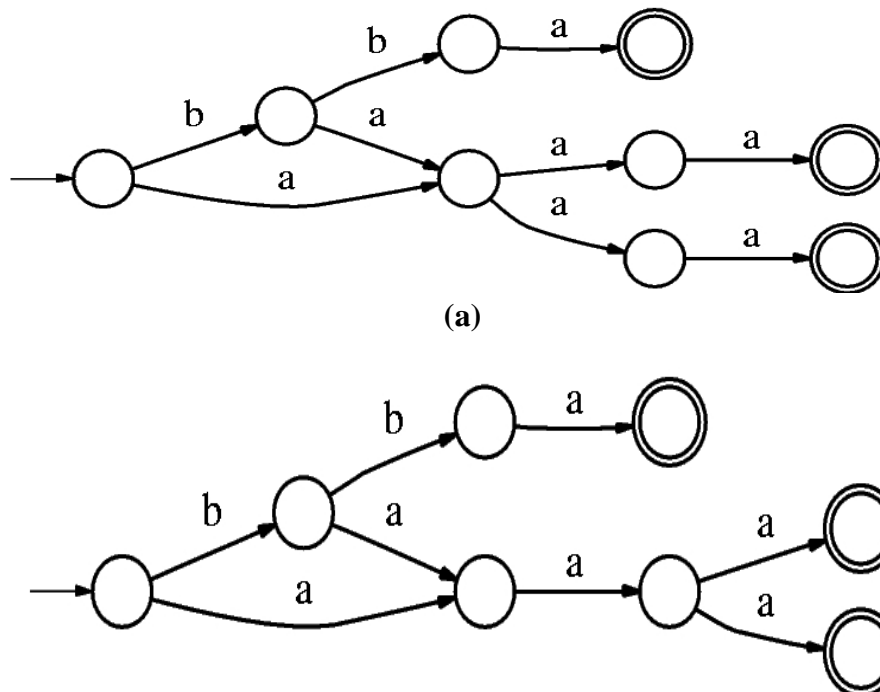


Fig 2.2 State Merging Example

### Automata Minimization

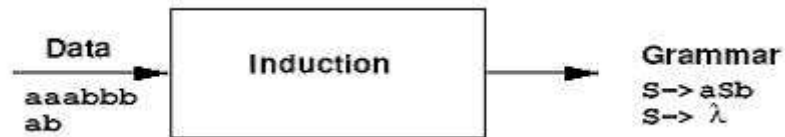
Classical automata theory [1,2,3] characterizes the unique finite automaton with minimal number of states that is equivalent to a given finite automaton. The Myhill-Nerode theorem [4,5] states that, among the many automata that accept a given language, there is a unique automaton that has a minimal number of states. This is called the *minimal* automaton of the language. It is well known how to efficiently minimize a deterministic finite-state automaton (DFA), in the sense of constructing another DFA that recognizes the same language as the original but with as few states as possible [1,3].

## 2.2 Regular Grammar Induction and Automata Learning

Grammar induction is a particular case of inductive learning. The general law is represented by a formal grammar or an equivalent machine. The problem of grammatical inference can be stated informally as follows. Given a finite set of strings along with a label indicating whether or not a string is a member of the language, find the rules which define the language. This is a popular machine learning problem that has wide applicability in both computational linguistics and related fields. For regular grammar inference, the task is to identify target grammar from finite set of positive examples and finite set of negative examples. Regular grammars can be equivalently represented by a set of production rules, a regular expression, a deterministic finite state automata (DFA) or a non-deterministic finite state automata (NFA).

Most work on the regular grammar inference has chosen DFA as the representation of the target regular grammar.

This is attributed to the following characteristics of a DFA: they are simple and easy to understand, there exists a unique minimum size DFA corresponding to the regular grammar, there exists polynomial time algorithm for several DFA operations.



**Fig 2.3 Grammar Induction from Examples**

Since regular grammars represent a widely used subset of formal language grammars, considerable research has focused on regular grammar inference (or equivalently, identification of the corresponding DFA). An understanding of the issues and problems encountered in learning regular languages (or equivalently, identification of the corresponding DFA) are therefore likely to provide insights into the problem of learning more general classes of languages.

### **2.3 Complexity-Result for DFA Learning**

Efficient learning of DFA is an important research problem. Goal is to learn the smallest automaton, which is compatible with the input. The problem of learning the minimum state DFA that is consistent with a given sample has been actively studied for over two decades. Exact learning of the target DFA from an arbitrary presentation of labeled examples is a NP-hard problem [6,7,8,21]. Gold showed that the problem of identifying the minimum state DFA consistent with a presentation  $S$  comprising of a finite non-empty set of positive examples  $S^+$  and possibly a finite nonempty set of negative examples  $S^-$  is NP-hard. Gold [21] also showed that it is not possible to exactly identify a language  $L$ , even in the limit of infinite data, given only positive examples. But it is possible to identify a stochastic regular language from only positive examples.

### **2.4 Training Pattern for Learning**

Once the sample data has been generated and labeled, inference is then conducted. It is assumed that the training data is collected using some knowledge of the target system to be inferred. The set of examples, known as the positive sample, is usually made of strings or sequences over a specific alphabet  $A$  a negative sample, i.e. a set of strings not belonging to the target language, can sometimes help the induction process. Efficient algorithms for DFA learning assumes that some additional information be provided to the learner. Pitt [24], Porat and Feldman [25], Dupont [22] and Lang et al. [15] assume a randomly-selected set of sample data. Parekh and Honavar [23] assume a structurally complete set. One necessary criterion for a structurally complete set of training data is that it covers every state transition of a DFA [22,23]. This requires that the algorithm which generates the training data knows something about the structure of the DFA, namely its state transitions. Angulin [6,7,8] assumes a live

complete set. If  $L$  is a regular set and  $A = (Q, q, F, \delta)$  is its canonical acceptor, then a live complete set for  $L$  is any finite set of strings  $P$  such that for every live state  $q$  of  $A$  there exist a string  $u \in P$  such that  $\delta(q, u) = q$ . In other words, live complete set contains a representative string for each live state of the target DFA. Oncina [9] assumes a characteristics set of samples. A characteristics set  $S = S^+ \cup S^-$  is such that  $S^+$  is structurally complete with respect to the target and  $S^-$  prevents merging of any two states that are not equivalent.

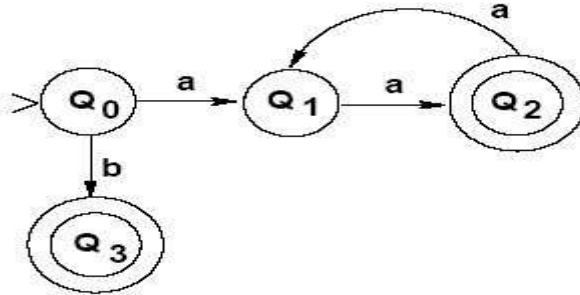


Fig 2.4 A DFA which has structurally complete set  $S^+ = \{b, aa, aaaa\}$

## 2.5 Active and Passive Learning

Regular grammatical inference methods that employ DFA as models can be divided into two broad classes: passive and active learning methods. In passive methods, a set of training data is supplied to the algorithm. In active learning approaches, the algorithm has some influence over which training data is labeled by the target DFA for model construction. Active learning approaches are typically iterative, in which membership queries are proposed periodically. In these iterative active approaches, the amount of training data available for inference grows over time, unlike passive approaches, in which a fixed set of training data are used for model construction. Active learning approaches are supervised but passive learning approaches are unsupervised.

## 3. Non Probabilistic Automata Learning

### 3.1 Learning with a Teacher: Active Learning

There are problem instances where only positive examples are available. Negative examples might not be available or are heavily biased or absurd examples. For such instances, solution is to learn with the help of a teacher.

#### Active Learning: $L^*$ Algorithm

Angulin [6,7,8] showed that DFA can be exactly learned in polynomial time with the help of a minimally adequate teacher **MAT (Minimally Adequate Teacher)** capable of answering membership and equivalence queries. Such a minimally adequate teacher also known as an oracle, against which the learner asks questions. This is also called learning with queries. Angulin showed that the learner must be able to answer membership queries (is  $x$  in the language) and equivalence queries (is  $M'$  equivalent to  $M$ , if not then give me a counterexample).

Angulin gives a polynomial-time algorithm [6] for learning deterministic finite state acceptors using membership and equivalence queries. The algorithm, known as  $L^*$  learns an unknown regular language and generates a minimal DFA that accepts the regular language. The algorithm infers the structure of the DFA by asking a teacher, who knows the unknown language, two types of questions: membership queries and equivalence queries. So this is supervised learning. On a membership query, the learner asks whether a string  $s$  is accepted by the unknown language, and the teacher answers true or false. On an equivalence query, the learner conjectures that the machine it has constructed is equivalent to the unknown language. The teacher replies that the conjecture is either correct or incorrect, and in the latter case gives a counter-example, which is a string accepted by one but not the other. In some instances, active learning assumes that the oracle knows something about the structure of the target DFA.

**Drawback**

This algorithm requires an external agent (an oracle), that can answer equivalence queries. The oracle indicates whether the current model is equivalent to the target DFA. If it is not, then it returns new training data that belongs to the target language but does not belong to the language encoded by the candidate model. Once again, this assumes that the oracle knows something about the structure of the target DFA.

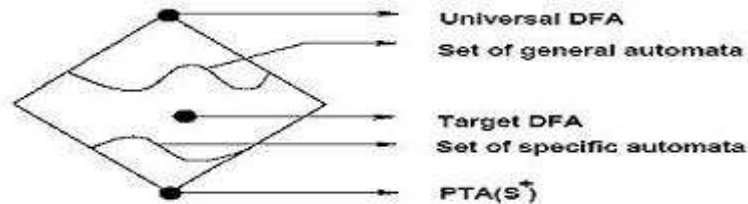
**Grammar Learning by Neural Network**

In the last years neural network models [39] were used in order to identify regular languages and they have been applied to stochastic samples. Rules defining the learned grammar was extracted from a neural network in the form of a DFA. These methods of regular language identification share the serious drawback of having long computational time. Moreover training neural network requires a large number of training examples. Neural network is essentially a Black Box that cannot explain the inherent learning process. Lack of understanding of the rules that underlie neural network performance has limited their use in some application domains.

**3.2 Learning without Teacher**

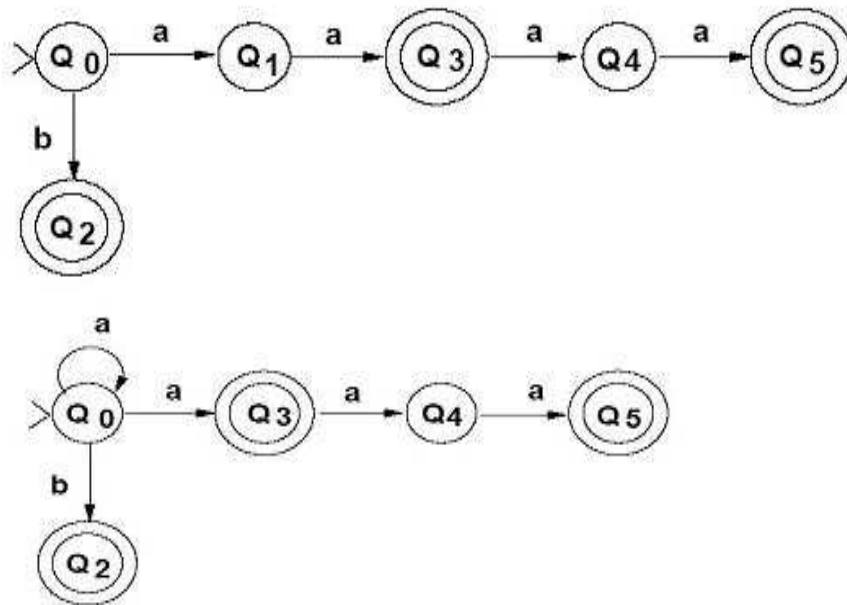
**Non-Stochastic DFA Induction: RPNI**

The *regular positive and negative inference* (RPNI) algorithm [13] is a polynomial time algorithm for identification of a DFA consistent with a given set  $S = S^+ \cup S^-$ . Here  $S^+$



**Fig 3.1. Search Space for DFA learning**

denotes the set of positive examples and  $S^-$  denotes the set of negative examples. A is consistent with a *sample*  $S = S^+ \cup S^-$  if it accepts all positive examples and rejects all negative examples. RPNI algorithm is guaranteed to identify a DFA that is consistent with the set of positive examples and the set of negative examples. Further if  $S$  happens to be a superset of a characteristic set for the target DFA then the algorithm is guaranteed to return a canonical representation of the target DFA. A labeled sample is provided as input to the algorithm. It constructs a prefix tree acceptor  $PTA(S^+)$  automata and numbers its states in the standard order.  $PTA(S^+)$  is a DFA that contains a path from the start state to an accepting state for each string in  $S^+$  modulo common prefixes. The strings of  $PTA(S^+)$  are sorted in lexicographic order and each state is numbered by the position of its corresponding string in the sorted list. Then it performs an ordered search in the space of partitions of the set of states of  $PTA$ , under the control of the set of negative examples. The algorithm iterates to find a more general hypothesis by systematically merging the states of the  $PTA$  in order. The derived automaton obtained by merging two states is tested for consistency with the set  $S^-$ . If the derived automaton accepts any string from  $S^-$  then the merge is rejected and the remaining states are considered for merging in order. Otherwise, the derived automaton that is consistent with  $S^-$  is treated as the new hypothesis and the state merging continues with the states of the new hypothesis. The algorithm terminates when no further state merges results in a derived automaton that is consistent with  $S^-$ . The last consistent hypothesis is then returned as the learned DFA.



**Fig 3.2. A Prefix Tree Automata and Merged Automata**

## 4. Probabilistic Automata Learning

### 4.1 Stochastic Automata Learning Algorithm

Carrasco and Oncina [9] proposed ALERGIA, a stochastic automata learning algorithm that learns a regular language given the strings belonging to the language. No information regarding the strings not belonging to the language is needed. It requires stochastic samples of the input strings (That means samples consisting of positive examples that appear repeatedly). Nevertheless, repeated occurrence of positive data compensates for lack of negative examples. This algorithm develops a prefix tree acceptor (PTA) from the sample and evaluates at every node the relative probabilities of the transitions coming out from that node. They merge the nodes of prefix tree acceptor following a well-defined order. During the merging process, lexicographic orders of the nodes are preserved. Merging is performed if the resultant automaton is within some statistical uncertainty. When further merging is not possible, the process ends. For merging states they have a notion of equivalence. Two states are equivalent if they have same number of outgoing arcs, same outgoing arc frequency on each alphabet symbol and equivalent successors.

#### Advantages

Experimentally the algorithm needs very short time and comparatively small samples in order to identify the regular set. It is suitable for recognition tasks where noisy samples or random samples are common. The algorithm has been shown to be very fast even for a large sample set. Its performance is comparable to that of a recurrent neural network.

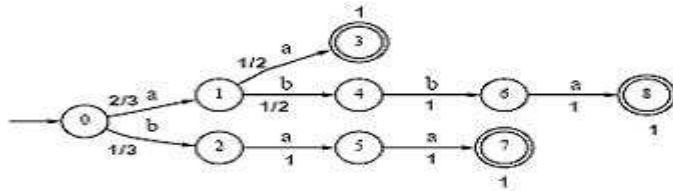


Fig 4.1 A Stochastic Prefix Tree Automata (PTA)

#### Drawbacks

Two kinds of errors are possible: rejection of compatibility between two equivalent nodes and acceptance of compatibility between two non-equivalent nodes. ALERGIA tends to merge too many states, even at a high confidence limit, leading to an over-general grammar. The resulting automaton frequently has loops in it, corresponds to regular expressions like a  $(b^*)c$ . Real data is seldom described by such repeated structures.

### 4.2 HMM Learning

#### Hidden Markov Model

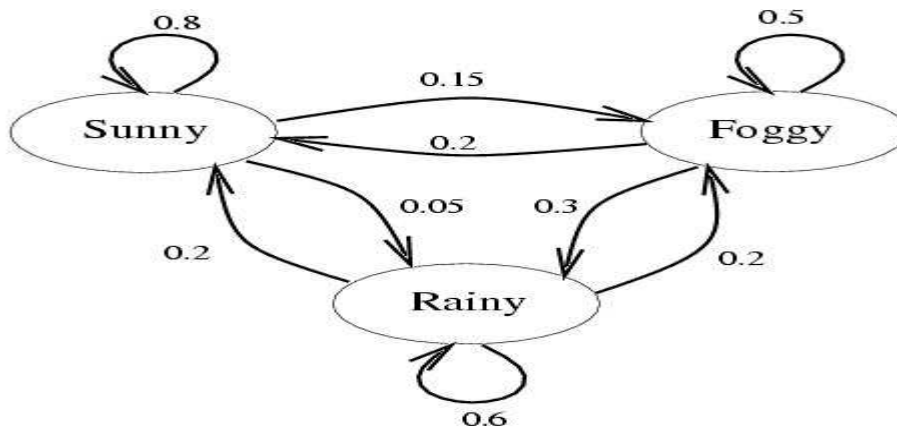
An HMM [14] consists of the following components:

- *Alphabet:*  $\Sigma = \{ b_1, b_2, \dots, b_M \}$
- *Set of States:*  $Q = \{ 1, \dots, K \}$
- *Transition Probabilities:*  $a_{ij}$  for all states  $i$  and  $j$   
In general,  $a_{i1} + \dots + a_{iK} = 1$ , for all states  $j = 1 \dots K$
- *Start Probabilities:*  $a_{0j}$  for all states  $j$   
These are the probability of starting with a particular state  $j$ .  
In general,  $a_{01} + \dots + a_{0K} = 1$ , for all states  $j = 1 \dots K$
- *Emission Probabilities:* For each state  $\pi$ ,  $e_i(b) = P(x_i = b | \pi_i = k)$   
These are the probability of emitting a certain letter while in a certain state.  
 $e_i(b_1) + \dots + e_i(b_M) = 1$ , for all states  $i = 1 \dots K$

In a HMM, probability of a sequence depends on the underlying state sequence. When the state sequence is known: (states=  $s_1, s_2 \dots s_n$ )

$$P(S) = P(a_1 | s_1) P(s_2 | s_1) P(a_2 | s_2) P(s_3 | s_2) \dots P(s_i | s_{i-1}) P(a_i | s_i)$$

When the state sequence is not known, usual approach is to apply Viterbi Algorithm [14 ] to get the best state sequence corresponding to the observation sequence. State sequence with maximum a prior probability is considered as best state sequence by Viterbi algorithm [14]. Such state sequence is assumed to the true state sequence and corresponding probability is computed according to the above equation. When the best state sequence (MAP sequence) is not wanted, every possible state sequence corresponding to an observation sequence is computed and probability of the observation sequence in that model is computed by taking the sum over all possible paths. For efficient computation, the forward and backward algorithm useful to compute the above formula in polynomial time [14].



**Fig 4.2 An Example Hidden Markov Model**

### **The Learning or Estimation Problem for HMM.**

The learning problem is how to adjust the HMM parameters, so that the given set of observations (training set) is represented by the model in the best way for the intended application. The quantity to be optimized during the learning process differs from application to application.

### Parameter Learning

**Baum-Welch algorithm** is used to find the unknown parameters of a HMM. It is also known as the **forward-backward algorithm**. The Baum-Welch algorithm is an EM (expectation-maximization) algorithm. It can compute maximum likelihood estimates and posterior mode estimates for the parameters (transition and emission probabilities) of a HMM, when given only emissions as training data. The algorithm has two steps: (1) calculating the forward probability and the backward probability for each HMM state; (2) on the basis of this, determining the frequency of the transition-emission pair values and dividing it by the probability of the entire string. This amounts to calculating the expected count of the particular transition-emission pair.

### Structure Learning

**Baum-Welch** algorithm can also be used to learn the structure of the HMM. Baum Welch estimation method assumes a certain topology and adjusts the parameters so as to maximize the likelihood of the model on the given data. If the structure is only minimally specified, then this method can find HMM structures by setting a subset of the parameters to zero (or close enough to zero so that pruning is justified).

## 4.3 Bayesian Model Merging

Model merging has been proposed as an efficient, robust, and cognitively plausible method for building probabilistic models in a variety of cognitive domains [10,11,12]. Stolke and Omuhondru described Bayesian model merging approach [10,11]. The strategy facilitates learning both the structure and the parameter of the model from given set of examples. The process begins with a maximum likelihood model that directly encodes the data. The method can be characterized as follows:

**Data incorporation:** Given a body of data  $X$ , an initial model  $M$  is built by explicitly accommodating each data point individually such that  $M$  maximizes the likelihood  $P(X|M)$ . The size of the initial model grows with the amount of data, and will usually not exhibit significant generalization.

**Structure merging:** A sequence of new models are built, obtaining  $M_{i+1}$  from  $M_i$  by applying a *generalization* or *merging* operator. The merging operation is dependent on the type of model at hand. But it generally has the property that data points previously 'explained' by a separate model substructures come to be accounted for by a single, shared structure. The merging process thereby gradually moves from a simple, instance-based model toward one that expresses structural generalizations about the data. To guide the search for suitable merging operations a criterion is needed that trades off the goodness of fit the data against the desire for 'simpler,' and therefore more general models. As a formalization of this tradeoff, they use the *posterior probability* of the model given the data. Posterior is proportional to the product of the prior probability and likelihood. The likelihood is defined by the model semantics, whereas the prior has to be chosen to express the bias, or prior expectation, as to what the likely models are. This choice is domain-dependent. A search strategy is needed to find models with high (maximal, if possible) posterior probability. A simple approach here is **Best-first search**. Starting with the initial model (which maximizes the likelihood, but usually has a very low prior probability), it explores all possible merging steps, and successively chooses the one (greedy search) or ones (beam search) that give the

greatest immediate increase in posterior. Merging is terminated when no further increase is possible.

### **HMM Structure Induction by Bayesian Model Merging**

An initial HMM is constructed as a disjunction of all the observed samples. Each sample is represented by dedicated HMM states such that entire model generates all and only the observed strings. This is similar to the PTA constructed by the ALERGIA algorithm for learning SFA. The initial probabilities are assigned as follows. The probability of entering a path according to each string from the start state is uniformly distributed. (if there are  $k$  paths corresponding to  $k$  strings in  $S^+$ , then probability of entering each path is  $1/k$ ). Within each path probability of observing a particular symbol at each state and probability of taking the outgoing arc from the state are both set to 1. The merging steps combine individual HMM states and gives the combined states emission and transition probabilities which are weighted average of the corresponding distributions for the states which have been merged. Successive merging of states generalizes the model. The merging operation which is repeatedly applied to a pair of HMM states, preserves the ability to generate all the samples observed by the initial model. But after merging of states, unobserved strings may also be generated by the HMM. That means the merging operation actually generalizes the sample data. Bayesian posterior probability is used to decide which states to merge and when to stop.

Given the current model with  $n$  states, Stolcke and Omuhondru generated a set of  $n^2$  candidate new models. Each candidate contains a single merge of a pair of states from the current model. The likelihood of each candidate model is computed using the forward algorithm. When generalization is done, the likelihood decreases since now the model will accept more strings than the ones in the data set. The candidate model with the least decrease in likelihood is selected. The intuition is to generalize as little as possible at each step. If at a step, the likelihood of all candidate models decreases beyond a threshold then the merging is stopped. At each step of model merging, Stolcke and Omohundro generated every possible pair of candidate merging and calculated corresponding model structure and posterior probability of training data for that model. The number of potential merges calculated in each step is a potential factor for their algorithm. Sometimes domain specific constraints are useful for reducing number of potential merges, but usually is  $O(|Q|^2)$ , where  $|Q|$  is the cardinality of the states of the model.  $|Q|$  grows linearly with number of samples observed, which makes such merging approach practically impossible. There is a trade off between model likelihood and a bias towards simpler model. For expressing preference among alternate models, it is required to have a notion of prior probability distribution for each model.

### **Advantages**

For most applications, it is not possible to specify HMM by hand. Instead HMM needs to be at least partly estimated from available sample data. All the previous HMM induction method specifies only learning parameter of the model once the structure is fixed. The Bayesian model merging approach can adjust the model topology with data. Structure induction methodology by model merging experimentally outperformed traditional methodology like Baum Welch. It produces models, which are more general and/or compact.

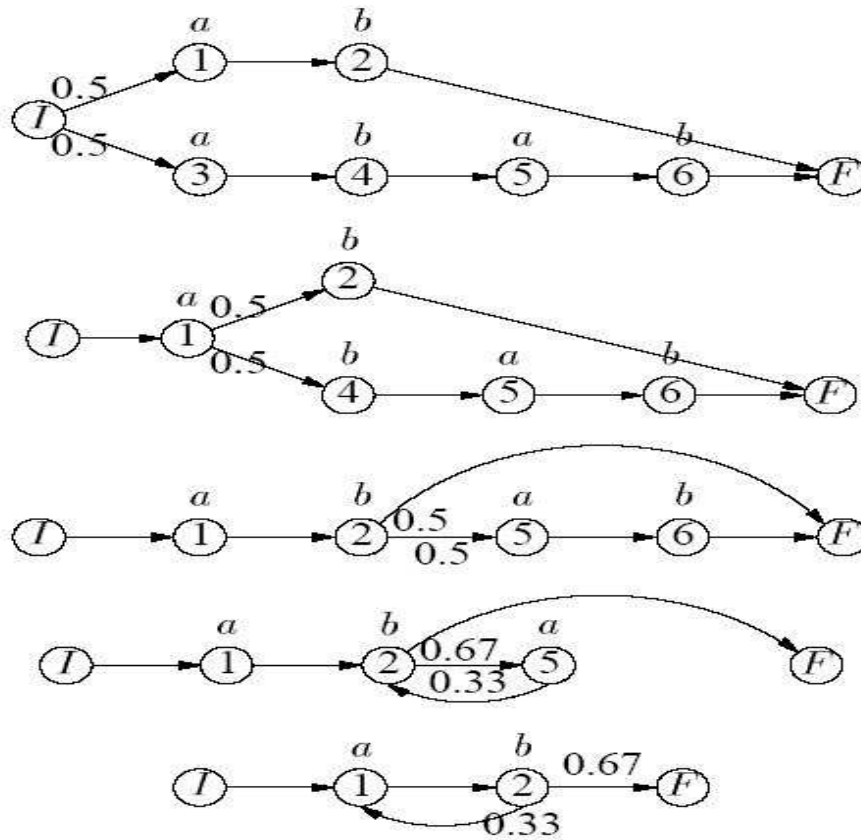


Fig 4.3. Model Merging for HMM

**Drawbacks**

Their heuristic is not guaranteed to find the appropriate sequence of merges and may result in a model which is not equivalent to the generating model. One of the major shortcomings of the model merging method is its inability to back off from a merging step, that is an overgeneralization for the new data. Bayesian model merging approach requires the use of priors. Sometimes calculation of prior is difficult. Experimentation with large range of priors as well as application specific ones is time consuming.

**Stochastic Context-Free Grammar Learning by Bayesian Model Merging**

Based on the model merging approach to HMM induction, Stolke and Omuhondru have extended the algorithm to apply to stochastic context-free grammars (SCFGs), the probabilistic generalization of CFGs.

**Data incorporation.** To incorporate a new sample string into a SCFG a top-level production (for the start non-terminal) is added that covers the sample precisely. One non-terminal for each observed terminal is also created.

**Merging.** The obvious analog of merging HMM states is the merging of non-terminals in a SCFG. This is indeed one of the strategies used to generalize a given SCFG by generating a grammar that generates more than its predecessor, while reducing the size of the grammar.

### **Comparison of Bayesian Approach with ALERGIA**

ALERGIA uses a depth first merging of states and runs in time polynomial in the size of  $S^+$ . It is guaranteed to converge to the target deterministic automata in the limit. On the other hand Bayesian model merging procedure considers all possible state merge at each step before merging two states and is thus computationally more expensive. Additionally there is a need to appropriately chosen priors in the Bayesian model merging procedure. Experimental results shows that relatively uninformed priors performs well. Although there is no guarantee to converge to the target HMM, the Bayesian model merging procedure has been successfully applied in various practical applications. Further Bayesian model merging procedure has been used to learn class based n grams and stochastic context free grammars which are more general in terms of their descriptive capabilities than the deterministic finite automata returned by ALERGIA.

## **5. Symbolic Statistical Learning**

### **5.1 Statistical Learning Preliminaries**

#### **Maximum Likelihood (ML) criterion**

In ML we try to maximize the probability of a given sequence of observations  $O$  with respect to the parameters of the model  $M$ . This probability is the total likelihood of the observations and can be expressed mathematically as,  $L = P(O|M)$  However there is no known way to analytically solve for the model, which maximize the quantity . But using an iterative procedure we can choose model parameters such that it is locally maximized.

#### **EM Algorithm**

In statistical computing, an **expectation-maximization (EM) algorithm** is an algorithm for finding maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved latent variables. EM alternates between performing an expectation (E) step, which computes the expected value of the latent variables, and a maximization (M) step, which computes the maximum likelihood estimates of the parameters given the data and setting the latent variables to their expectation.

Let the observed variables be known as  $y$  and the latent variables as  $z$ . Together,  $y$  and  $z$  form the complete data. Assume that  $p$  is a joint model of the complete data with parameters  $\theta$ :

$P(y, z | \theta)$ . An EM algorithm will then iteratively improve an initial estimate  $\theta_0$  and construct new estimates  $\theta_1$  through  $\theta_N$ . An individual re-estimation step that derives  $\theta_{n+1}$  from  $\theta_n$  takes the following form:

$$\theta_{n+1} = \arg \max_{\theta} \sum_z p(z | y, \theta_n) \log p(y, z | \theta)$$

In other words,  $\theta_{n+1}$  is the value that maximizes (M) the expectation (E) of the complete data log-likelihood with respect to the conditional distribution of the latent data under the previous parameter value. This expectation is usually denoted as  $Q(\theta)$ :

$$Q(\theta) = \sum_z p(z | y, \theta_n) \log p(y, z | \theta)$$

What is calculated in the first step are the fixed, data-dependent parameters of the function  $Q$ . Once the parameters of  $Q$  are known, it is fully determined and is maximized in the second (M) step of an EM algorithm. It can be shown that an EM iteration does not decrease the observed data likelihood function, and that the only stationary points of the iteration are the stationary points of the observed data likelihood function. In practice, this means that an EM algorithm will converge to a local maximum of the observed data likelihood function. EM is particularly useful when maximum likelihood estimation of a complete data model is easy. "Expectation-maximization" is a description of a class of related algorithms, not a specific algorithm; The Baum-Welch algorithm is an example of an EM algorithm applied to hidden Markov models.

## 5.2 Logic Programming Preliminaries

In logic programming, a program DB is a set of definite clauses and the execution is search for an SLD refutation of a given goal G. The top-down interpreter recursively selects the next goal and unfolds it into sub-goals using a non-deterministically chosen clause. The computed result by the SLD refutation is an answer substitution (variable binding). Usually there are more than one refutation for G, and the search space for all refutations is described by an SLD tree which may be infinite depending on the program and the goal. More often than not, applications require all solutions. In natural language processing for instance, a parser must be able to find all possible parse trees for a given sentence, as every one of them is syntactically correct. Similarly in statistical abduction, we need to examine all explanations to determine the most likely one. All solutions are obtained by searching the entire SLD tree, and there is a choice of the search strategy. In Prolog, the standard logic programming language, backtracking is used to search for all solutions in conjunction with a fixed search order for goals (textually from left-to-right) and clauses (textually top-to-bottom) due to the ease and simplicity of implementation. The problem with backtracking is that it forgets everything until up to the previous choice point, and hence it is quite likely to prove the same goal again and again, resulting in exponential search time. One answer to avoid this problem is to store computed results and reuse them whenever necessary. OLDT is such an instance of memoizing scheme. Reuse of proved sub-goals in OLDT search often drastically reduces search time for all solutions, especially when refutations of the top goal include many common subrefutations. Take as an example a logic program coding of a HMM. For a given string  $s$ , there exist exponentially many transition paths that output  $s$ . OLDT search applied to the program however only takes time linear in the length of  $s$  to find all of them unlike exponential time by Prolog's backtracking search. It is required at programming level a tabulation mechanism for structure sharing of partial explanations between subgoals.

### 5.3 Parameter Learning of Logic Programs

Parameter learning is common in various fields from neural networks to reinforcement learning to statistics. It is used to tune up systems for their best performance, be they classifiers for statistical models. The work done by Sato and Kameya [26,27] for parameter learning of logic programs has been studied. Sato and Kameya focuses on statistical parameter learning of parameterized logic programs, i.e. definite clause programs containing probabilistic facts with a parameterized distribution. They assumed that facts (unit clauses) in a program are probabilistically true and have a parameterized distribution. Other clauses, non-unit definite clauses, are always true as they encode laws such as if one has a pair of blood type genes a and b, one's blood type is AB. To learn parameters in a program, they applied ML (maximum likelihood) estimation to observed data. Parameter learning in 2 phases: Search for all explanations for the observations. (Redundancy is eliminated by tabulating partial explanations using OLDT search.) This phase returns a support graph representing the discovered explanations. Run the graphical EM algorithm on the support graph, and learn the parameters of the distribution associated with the program.

The main contribution of Sato and Kameya for parameter learning of logic program are the following:

- Distribution semantics for parameterized logic programs.
- The graphical EM algorithm (combined with tabulated search), a general yet efficient EM algorithm that runs on support graphs
- Comparable experimental performance with traditional learning approaches for statistical models.

### 5.4 Symbolic Statistical Modeling Language (PRISM)

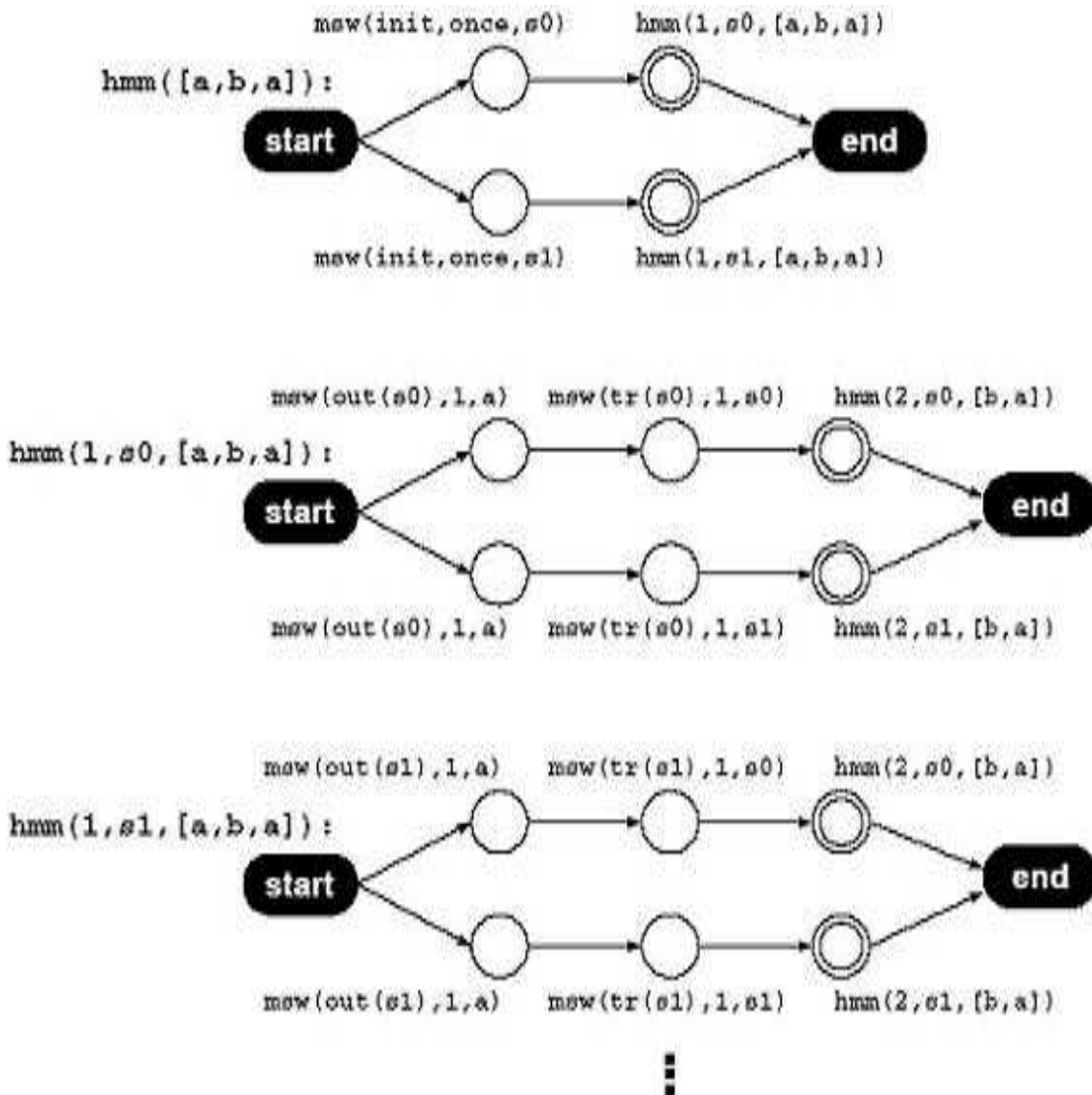
Sato and Kameya developed a symbolic statistical modeling language, PRISM. The language is intended for modeling complex symbolic-statistical phenomena such as discourse interpretation in natural language processing, stochastic NLP, consumer behavior and gene inheritance interacting with social rules. PRISM provides three modes of execution. The sampling execution corresponds to a random sampling drawn from the distribution defined by the modeling part. The second one computes the probability of a given atom. The third one returns the support set for a given goal. These execution modes are available through built-in predicates.

### 5.5 Graphical EM

Sato and Kameya proposed a new EM algorithm, the graphical EM algorithm, that runs for a class of parameterized logic programs representing sequential decision processes where each decision is exclusive and independent. Graphical EM is derived from distribution semantics [28]. It runs on a new data structure called support graphs describing the logical relationship between observations and their explanations, and learns parameters by computing inside and outside probability generalized for logic programs.

**Data Structure Used**

Graphical EM uses explanation graph and support graph as the underlying data structure. Explanation graph for G is obtainable by searching for all explanations of G using tabling. Tabling remembers successful goals and reuses them to avoid re computation of the same goal. Probabilities are computed using explanation graphs which are a compact representation of statistical-logical dependency among events. In an explanation graph, sub-graphs are partially ordered and shared by super-graphs. Sharing of sub-graphs causes sharing of computations by dynamic programming. Thus efficient computation is achievable. A support graph consists of totally ordered disconnected sub-graphs. Sub-graph labeled comprises start node, end node and explanation graphs with table nodes, switch nodes. Data sharing is achieved through the distinct table nodes referring to the same sub-graph.



**Fig 5.1. Example of a Support Graph**

### **Assumptions**

Graphical EM algorithm assumes: uniqueness condition (one cause yields one effect), finite support condition (finite number of explanations for an observation), acyclic support condition (explanations must not be cyclic), exclusiveness condition (explanations must be mutually exclusive) and independence condition (events in an explanation must be independent). Graphical EM achieves the same time complexity as BW and IO when OLDT search is used in explanation graph construction. OLDT search strategy applied to graphical EM algorithm gives a better EM algorithm, called fast EM.

## **5.6 Advantage of Symbolic Statistical Modeling**

The complexity analysis shows that when combined with OLDT search for all explanations for observations, the graphical EM algorithm has the same time complexity as existing EM algorithms, i.e. the Baum-Welch algorithm for HMMs and the Inside-Outside algorithm for PCFG. Learning experiments with PCFGs indicated that the graphical EM algorithm can significantly outperform the Inside-Outside algorithm. Also distribution semantics proposed in the paper enabled to derive an EM algorithm for the parameter learning of logic programs for the first time.

## **5.7 Future Directions for Symbolic Statistical Modeling**

Learning ability can be further improved by introducing priors instead of ML estimation to cope with data sparseness. Current parameter estimation is based on estimation only from positive examples. Use of negative examples for learning parameter of logic programs can be an interesting area for research. Also it is worth to see the result of relaxing some of the applicability conditions. Symbolic statistical modeling can be incorporated in other research fields.

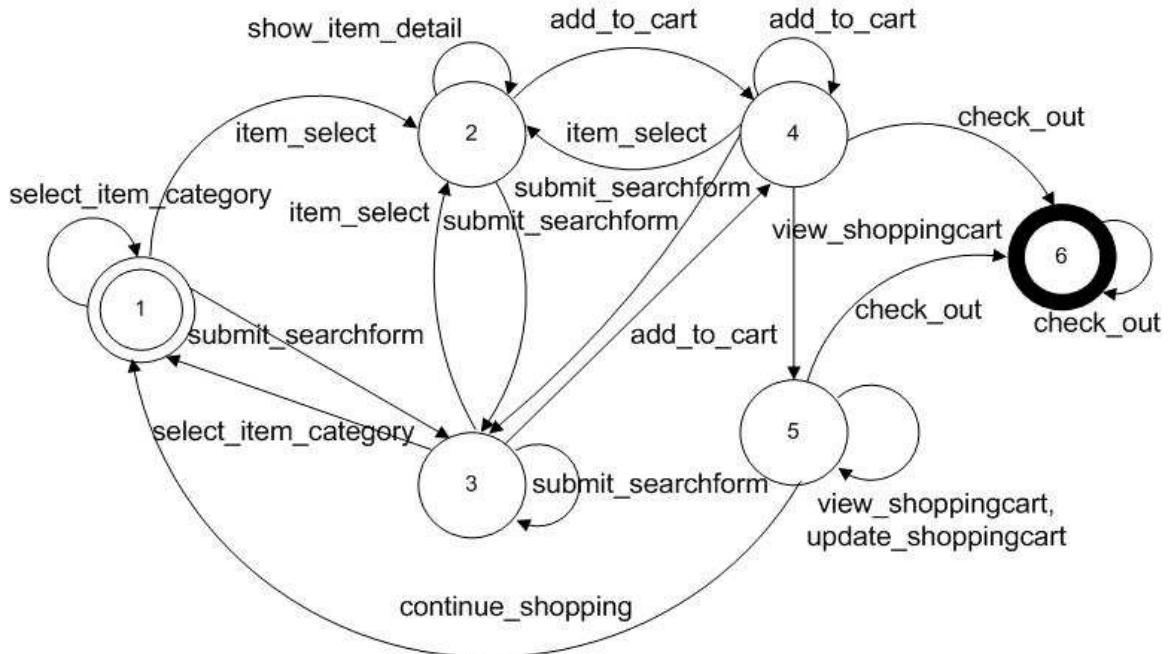
## **6. Preliminary Work**

Preliminary work on application of automata learning has been done. An automata-based process model to capture users Web transactional behavior was learned. The model was realized by coupling techniques from content analysis of Web documents, automata learning and statistical classification. The process model and associated techniques have been incorporated into Guide-O [37], a prototype system that facilitates online transactions using speech/keyboard interface (Guide-O-Speech), or with limited-display size handhelds (Guide-O-Mobile).

### **6.1 Process Model as Automaton**

Formally, the process model is defined as follows: Let  $C = \{c_0, c_1, \dots\}$  be a set of concepts, and  $I(c)$  denote the set  $c$  of concept instances. Let  $Q = \{q_0, q_1, \dots\}$  be a set of states. With every state  $q_i$  we associate a set  $S_i \subseteq C$  of concepts. Let  $O = \{o_0, o_1, \dots\}$  be a set of operations. An operation  $o_i$  can take parameters. A transition  $\delta$  is a function  $Q \times O \rightarrow Q$ , and a concept operation  $\rho$  is also a function  $C \rightarrow O$ . Operations label transitions, i.e. if  $\delta(q_i, o) =$

$q_j$  then  $o$  is the label on this transition. An operation  $o = \delta(c)$  is enabled in state  $q_i$  whenever the user selects an instance of  $c \in S_i$  and when it is enabled a transition is made from  $q_i$  to state  $q_j = \delta(q_i, o)$ .



**Fig 6.1. Learned Process Model for Web Transactions**

Figure 6.1 illustrates a process model. The concepts associated with state 1 are “Item Taxonomy”, “Item List”, and “Search Form”. This means that if these concept instances are present in the Web page given to 1 as its input then they will be extracted and presented to the user. The user can select any of these three concepts. We say that, the user chooses “Item Taxonomy” concept whenever she selects a particular category of item in the taxonomy and upon selection the corresponding operation *select item category* is invoked. This amounts to fetching a new Web page corresponding to the selected category and a transition is made to state 1. When the user selects the “Search Form” concept she is required to supply the form input upon which the *submit searchform* operation is invoked. This amounts to submitting the form with the user-supplied form input. A Web page consisting of the search results is generated and a transition is made to 3. Lastly a user can select an item from a list of items in the “Item List” concept. This will result in a Web page describing the item selected and the transition labeled *item select* is made to state 2. The state transitions of other states can be similarly described. In the figure, state 6 is deemed as the final state.

## 6.2 Learning from Web Transactions

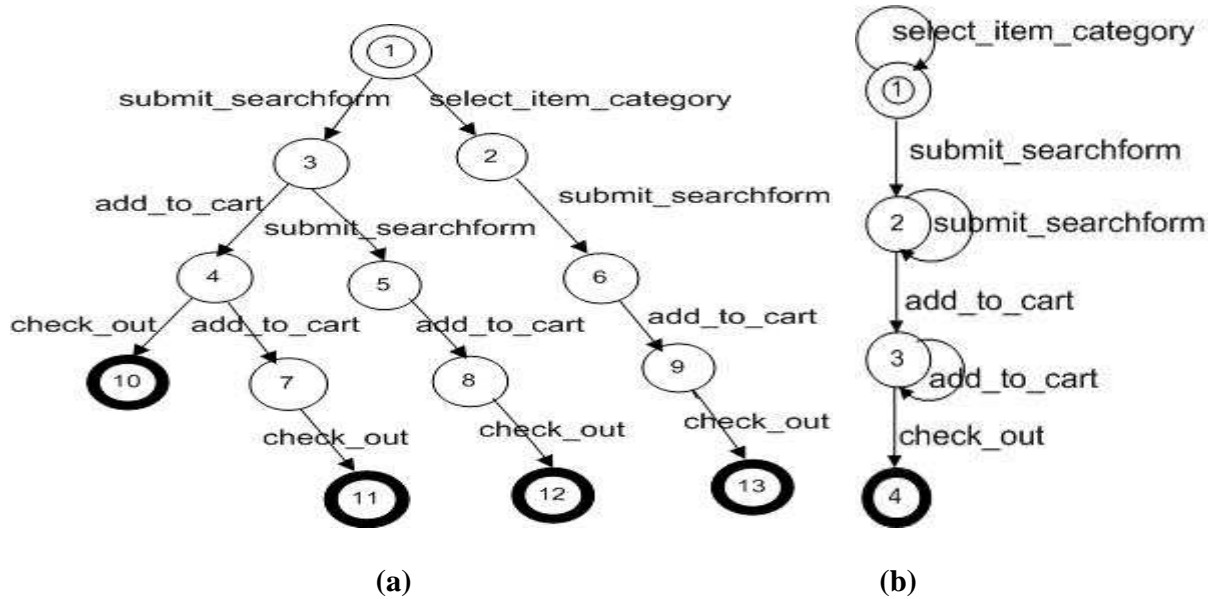


Fig 6.2. A Prefix Tree Automata and Learned DFA

For a process model, training sequences are logs of user activities during web transactions. Each training sequence we used for learning the process model consists of strings whose elements are operations on concepts; *e.g.*  $\langle submit\_searchform, item\_select, add\_to\_cart, check\_out \rangle$ . Each of the training sequence is labeled as either positive (successful transactional sequence) or negative (unsuccessful transactional sequence). From the training transactional sequences, we constructed the set  $S^+$  (by taking all the transactional sequences labeled as positive). We also constructed the set  $S^-$  (by taking all the transactional sequences labeled as negative). We first construct a prefix tree automata (PTA) as shown in 6.2(a) using only the examples in the positive set  $S^+$ . In 6.2(a), the sequence of operations along each root-to-leaf path constitutes a string in  $S^+$ . For this example the negative set  $S^-$  consists of the strings:  $\{\langle check\_out \rangle, \langle submit\_searchform, add\_to\_cart \rangle, \langle submit\_searchform, check\_out \rangle, \langle submit\_searchform, select\_item\_category, add\_to\_cart, check\_out \rangle\}$ . The prefix of every string in  $S^+$  is associated with a unique state in the prefix tree. The prefixes are ordered and each state in the prefix tree automata is numbered by the position of its corresponding prefix string in this lexicographic order. Next we generalize the prefix tree automata by state merging. We choose state pairs  $(i, j)$ ,  $i < j$  as candidates for merging. The candidate pair  $i, j$  is merged if it results in a consistent automata. For example merging the pair (1,2) is consistent whereas (3,4) is not merged as the resulting automata will accept the string  $\langle submit\_searchform, check\_out \rangle$  in  $S^-$ . Figure 6.2 (b) shows the final DFA returned by the algorithm which is consistent with  $S^+$  and  $S^-$ . Since we do at most  $Q^2$  state merging, where  $Q$  is the cardinality of  $S^+$ , the time complexity is polynomial. In the next subsection, experimental evaluation of the performance of the learned process model is presented.

## 6.3 Evaluation of the Model

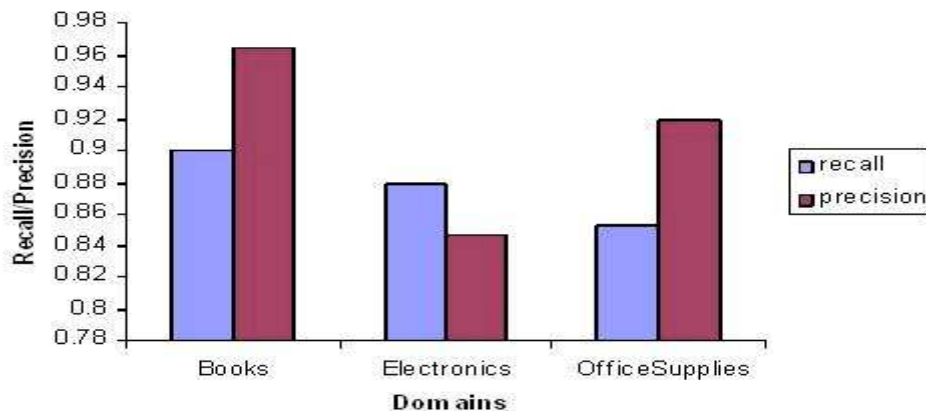
### Experimental Setup

We choose three different domains – books, electronics and office supplies. We collected approximately 200 traces from 30 web sites over those 3 domains. These were labeled sequences whose elements are concept operations. A number of CS graduate students were enlisted for this purpose. Each subject was manually trained to identify concept instances from web pages. To assist in labeling web pages with concept instances and operations, a user interface was provided to each subject during trace collection. Specifically each student was told to do around 5 to 6 transactions with a Web browser and the sequences were generated by monitoring their browsing activities. They labeled each transactional sequence as either successful transaction (which became a positive example) or unsuccessful transaction (which became a negative example).

### Performance Metrics for Evaluation

We evaluated our model learning scheme quantitatively. We used the traces collected by the subjects to measure the accuracy of our model learning. Traces were divided in two parts: training and testing. For testing the performance of the learned process model, 80 transactional sequences were used. Learning accuracy of the process model in terms of recall and precision has been measured. Recall for a process model is the ratio of the number of successful transactions accepted by the model over total number of such transactions. For precision, denominator is the total number of accepted transactions (either labeled as successful or unsuccessful).

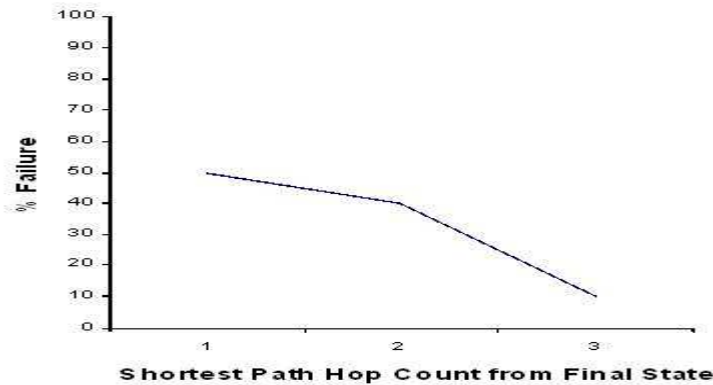
### Learning Accuracy for Different Domains



**Fig 6.3. Recall/Precision Characteristics of Learned Process Model**

As the above figure displays, recall precision characteristics were 90% and 96% for books domain, 88% and 84% for electronics domain, 84% and 92% for office supplies domain.

### **Failure Analysis**



**Fig 6.4. Failure Analysis of the Learned Model**

Failure analysis was performed from the experimental results. Test traces that were not accepted by our model were denoted as failure traces. We identified the state, which is reached until a failure is accounted and measured the hop count from that state to the accepting state. Analysis of failure curve indicates that almost half of the failure traces ends one hop away from the final state of the model. And only 10% failure ends three or more hop away from final state. This result emphasizes the power of our model that can direct the user as close as to the final state even for failure scenario.

## **7. Discussion and Future Research**

This work addresses the comparative analysis, problems and solutions to different aspects of automata learning, grammar induction and symbolic statistical modeling. This study is motivated by partly to better understand the process of inductive inference and partly by the numerous applications of such learning. Application of automata learning is described to learn a process model from example transactional sequences. Experimental evaluation is also reported. Possible future directions are application of probabilistic automata learning for modeling of proteins, application to text categorization or text mining. A future area of research is to learn deductive spreadsheet (DSS) [38] expressions from given examples. Thus grammar induction can be efficiently applied to a rule based system. The study on symbolic statistical learning for parameterized logic program also motivates to incorporate similar statistical modeling into XSB system, which is a tabled logic-programming environment [34,35,36]. Incorporation of learning ability into XSB will be beneficial to the users interested in Program Analysis, Model-checking and Data Mining. Finally, addition of parameter learning ability from DSS expressions is a possible future research.

## 8. References

- [1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
- [2] Hopcroft, John E., & Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass.: Addison-Wesley.
- [3] Papadimitriou, C. (1994), *Computational Complexity*. Addison-Wesley, Reading, MA. Rozenberg, G. & A. Salomaa, eds. (1997), *Handbook of Formal Languages*, 3 vols. Springer, Berlin.
- [4] A. Nerode, "Linear automaton transformations." *Proceedings of the American Mathematical Society*, **9**, 1958, 541 - 544.
- [5] J. Myhill, "Finite automata and the representation of events." *WADD TR-57-624*, Wright Patterson AFB, Ohio, 1957.
- [6] D. Angulin. Learning regular sets from queries and counterexamples. In *Information and Computation*, volume 75(2), pages 87-106, November 1987.
- [7] D. Angulin. On the complexity of minimum inference of regular sets. *Information and control*, 39(3):337-350, 1978.
- [8] D. Angulin, Inductive inference of formal languages from positive data. *Information and control*, 45(2):117-135, 1980.
- [9] Carrasco, Rafael C., & Jos E Oncina, 1994. Learning stochastic regular grammars by means of a state merging method.
- [10] A. Stolcke & S. Omohundro (1994), Best-first Model Merging for Hidden Markov Model Induction. Technical Report TR-94-003, ICSI, Berkeley, CA.
- [11] A. Stolcke & S. Omohundro (1994), Inducing Probabilistic Grammars by Bayesian Model Merging. In *Grammatical Inference and Applications*, R. C. Carrasco & J. Oncina, editors, Springer, pp. 106-118.
- [12] A. Stolcke & S. Omohundro (1992), Hidden Markov Model Induction by Bayesian Model Merging. In *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan & C. L. Giles, editors, Morgan Kaufman, pp. 11-18.
- [13] J. Oncina and P. Garc. Inferring regular languages in polynomial update time. World Scientific Publishing, 1991. *Pattern Recognition and Image Analysis*.
- [14] Rabiner, L.R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257-286
- [15] Lang, K 1992, Random DFA\_s can be approximately learned from sparse uniform sample. Pages 45-52 of *Proceedings of the 5th ACM workshop on Computational Learning Theory*
- [16] Fu, K. 1982, *Syntactic Pattern Recognition and Applications*. Prentice Hall, NJ.
- [17] 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.

- [18] Feldman, J. A., Lakoff, G., Stolcke, A., Weber, S. H. (1990) *Miniature Language Acquisition : A Touchstone for Cognitive Science*. Tech. rept. TR-90-009 International Computer Science, Institute, Berkeley, CA.
- [19] R. Rivest and R. Schapire. Diversity-based inference of finite automata. In *Proc. 28th IEEE Symposium on Foundations of Computer Science*, pages 78–87. IEEE Computer Society Press, 1987.
- [20] R. Rivest and R. Schapire. A new approach to unsupervised learning in deterministic environments. In *Proc. of the 4th International Workshop on Machine Learning*, pages 364–375. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1987.
- [21] Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and Control* 37,302–320.
- [22] P. Dupont. Incremental regular inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147*, pages 222–237, 1996.
- [23] R. G. Parekh and V. G. Honavar. An incremental interactive approach for regular grammar inference. In *Proceedings of the Third ICGI-96 (Lecture Notes in Artificial Intelligence 1147)*, pages 238–250. Springer Verlag, 1996.
- [24] L. Pitt. Inductive inference, DFAs and computational complexity. In *Proceedings of the International Workshop on Analogical and Inductive Inference (Lecture Notes in Artificial Intelligence 397)*, pages 18–44. Springer Verlag, 1989.
- [25] S. Porat and J. Feldman. Learning automata from ordered examples. *Machine Learning*, 7:109–138, 1991.
- [26] Sato, T. and Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. *JAIR*, Vol.15, pp.391–454, 2001.
- [27] Kameya, Y. and Sato, T.: Efficient EM learning with tabulation for parameterized logic programs. *L2000, LNAI*, Vol.1861, pp.269–294, 2000.
- [28] Sato, T.: A statistical learning method for logic programs with distribution semantics, *ICLP95, Tokyo*, pp.715–729, 1995.
- [29] J. Bongard and H. Lipson. Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 169–176. IEEE Computer Society, 2004a.
- [30] J. Bongard and H. Lipson. Automating system identification using co-evolution. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.
- [31] S. M. Lucas and T. J. Reynolds. Learning deterministic finite automata with a smart state labelling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27: 1063–1074, 2005.
- [32] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference (Lecture Notes in Artificial Intelligence 1433)*, pages 1–12. Springer-Verlag, 1992.
- [33] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, 4:603–632, 2003.

- [34] Konstantinos F. Sagonas, Terrance Swift, David Scott Warren: The XSB Programming System. Workshop on Programming with Logic Databases (Informal Proceedings), ILPS 1993.
- [35] Konstantinos F. Sagonas, Terrance Swift, David Scott Warren: XSB as a Deductive Database. SIGMOD Conference 1994
- [36] I. V. Ramakrishnan, Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David Scott Warren: Efficient Tabling Mechanisms for Logic Programs. ICLP 1995: 697-711
- [37] Zan Sun, Jalal Mahmud, Saikat Mukherjee and I.V. Ramakrishnan, "Model-directed Web Transactions under Constrained Modalities", submitted to International World Wide Web Conference WWW'2006.
- [38] C.R Ramakrishnan, I.V. Ramakrishnan and David S. Warren: "XcelLog: A User - Centered Deductive Spreadsheet System", Technical Report.
- [39] Christian W. Omlin and C. Lee Giles. Extraction of Rules from Discrete-time Recurrent Neural Networks. Neural Networks, Vol. 9, No. 1, pp. 41-52, 1996