Learning Stochastic Regular Grammars by Means of a State Merging Method *

Rafael C. Carrasco
Jose Oncina
Departamento de Tecnología Informática y Computación
Universidad de Alicante, E-03071 Alicante
E-mail: (carrasco, joncina)@dtic.ua.es

Abstract

We propose a new algorithm which allows for the identification of any stochastic deterministic regular language as well as the determination of the probabilities of the strings in the language. The algorithm builds the prefix tree acceptor from the sample set and merges systematically equivalent states. Experimentally, it proves very fast and the time needed grows only linearly with the size of the sample set.

1 Introduction

Identification of stochastic regular languages (SRL) represents a highly interesting question within the field of grammatical inference. Indeed, most of realistic situations involve examples provided by a random source. The assumption of stochastic behaviour has important consequences on the learning process. Gold[1] introduced the criterion of *identification in the limit* for successful learning of a language. He also proved that regular languages cannot be identified if only *text* (i.e., only strings in the language) is given, but they can be identified if a *complete presentation* (where all strings are classified as belonging or not to the language) is provided. However, Angluin[2] proved that a wide range of distribution classes, including the SRL, are identifiable from *positive samples* (text) with probability one.

With this aim, some attempts to find suitable learning procedures have already been done. Maryanski and Booth[3] used a chi-square test in order to filter regular grammars provided by heuristic methods. Although convergence to the true one was not guaranteed, acceptable grammars were always found. The approach of van der Mude and Walker[4] merges variables in a stochastic regular grammar, where Bayesian criteria are applied. In that paper[4], convergence to the true grammar was not proved and the algorithm showed too slow for application purposes.

*Work partially supported under grant TIC93-0633-C02-02 from CICYT (Programa Nacional de Tecnologías de la Información y de las Comunicaciones)

In the last years, neural network models were used in order to identify regular languages [5, 6, 7, 8] and they have been applied to the problem of stochastic samples[9]. However, these methods share the serious drawback that long computational times and vast sample sets are needed. Hidden Markov models are used by Stolcke and Omohundro[10] in order to maximize the probability of the sample, but they include *a priori* probabilities in order to penalize the size of the automaton.

On the other hand, an algorithm is available[11] which allows for the correct identification in the limit of any regular language if a complete presentation is given. Moreover, the time needed by this algorithm in order to output an hypothesis grows at most as s^3, being s the size of the sample. Experimentally, its behaviour is in average linear in s. In the present paper, we will follow the same guidelines and present an algorithm (`ALERGIA`) which builds the prefix tree acceptor (PTA) from the sample and evaluates at every node the relative probabilities of the transitions coming out from the node. Next, it tries to merge couples of nodes, following a well defined order (essentially, that of the levels in the PTA or lexicographic order). Merging is performed if the resulting automaton is —within statistical uncertainty— equivalent to the PTA. The process ends when further merging is not possible. We will introduce some definitions in section 2, and comment on the difficulties related to stochastic regular languages identification in section 3. A more detailed description of `ALERGIA` can be found in sections 4 and 5. Finally, results and discussion will be presented in section 6.

2 Preliminaries

Let $\mathcal{A}$ be a finite alphabet, $\mathcal{A}^*$ the set of all strings on $\mathcal{A}$ and λ the empty string such that for every symbol a in $\mathcal{A}$ satisfies $a\lambda = \lambda a = a$. If w and x are strings of symbols in $\mathcal{A}^*$ and $w = xa$, then we will also write $x = wa^{-1}$. A *stochastic finite automaton* (SFA), $A = (\mathcal{A}, Q, P, q_1)$, consists of an alphabet $\mathcal{A}$, a finite set of nodes $Q = \{q_1, q_2, \ldots q_n\}$, with q_1 the initial node, and a set P of probability matrices $p_{ij}(a)$ giving the probability of a transition from node q_i to node q_j led by the symbol a in the alphabet. If we call p_{if} the probability that the string ends at node q_i, the following constraint applies:

$$p_{if} + \sum_{q_j \in Q} \sum_{a \in \mathcal{A}} p_{ij}(a) = 1. \tag{1}$$

The probability $p(w)$ for the string w to be generated by A is defined by:

$$\begin{aligned} p(w) &= \sum_{q_j \in Q} p_{1j}(w)\, p_{jf} \\ p_{ij}(w) &= \sum_{q_k \in Q} \sum_{a \in \mathcal{A}} p_{ik}(wa^{-1}) p_{kj}(a) \end{aligned} \tag{2}$$

and the language generated by the automaton A is defined as:

$$L = \{w \in \mathcal{A}^* : p(w) \neq 0\} \tag{3}$$

Those languages generated by means of a SFA are called *stochastic regular languages.* In case the SFA contains no *useless nodes*[1], it generates a probability distribution for the

[1] A node q_i is useless if there are no strings $x, y \in \mathcal{A}^*$ such that $\sum_j p_{1i}(x) p_{ij}(y) p_{jf} \neq 0$.

strings in $\mathcal{A}^*$:

$$\sum_{w \in \mathcal{A}^*} p(w) = 1. \tag{4}$$

Finally, two SRL are said to be *equivalent* if they provide identical probability distributions over $\mathcal{A}^*$. Note that it is not enough that two languages L_1 and L_2 include the same strings for them to be equivalent, also the probability of every string must be equal:

$$L_1 \equiv L_2 \Longleftrightarrow p_1(w) = p_2(w)\ \forall w \in \mathcal{A}^*. \tag{5}$$

In this work we will limit ourselves to *deterministic stochastic finite automata* (DSFA). This means that for every node $q_i \in Q$ and symbol $a \in \mathcal{A}$, there exists at most one node such that $p_{ij}(a) \neq 0$. In such cases, a transition function $k = \delta(i, a)$ can be defined. This function gives the final node q_k for the transition starting at q_i and driven by symbol a. The probability of this single transition will be denoted by $p_i(a)$. In contrast to non-stochastic automata, determinism is an effective restriction. Indeed, it is not generally possible to find a stochastic DFA equivalent to a given non-deterministic SFA.

3 Identifying Regular Languages

A complete sample S consistent with L consists of two subsets: S_+ with strings in L (*positive examples*) and S_- with strings not in L (*negative examples*). If only S_+ is presented, then S is a *positive sample* or *text*. The algorithm *identifies in the limit* L if adding new examples to S may only produce a finite number of changes of hypothesis. Negative examples play a relevant role, since they may be necessary in order to reject a language L' whose only difference with L lays on $L' - L$ (and such languages exist because an order which respects inclusion is not defined).

However, samples of SRL consist only of positive examples which appear repeatedly, according to the probability distribution expressed in eq. (2). Nevertheless, the statistical regularity is able to compensate the lack of negative data. As proved in ref.[2], many recursively numerable sets of distributions —in particular SRL when probabilities are restricted to rationals— are identifiable with probability one, again by means of enumerative algorithms.

Enumerative methods are experimentally unfeasible and the search of fast and reliable algorithms for identification becomes a challenging task.

The aim of this work is to find an algorithm which identifies in the limit stochastic regular languages and whose complexity does not grow exponentially with the size of S. Our approach will be based on the one proposed in ref.[11] for the identification of (non-stochastic) regular languages. For this reason, we will briefly describe it in the following.

Given a language L, the minimum DFA generating L is called the *canonical acceptor* $M(L)$. On the other hand, if S is a finite complete sample of L the prefix tree acceptor T of S is defined as the minimum automaton accepting only the (finite) set of strings S. For instance, the canonical acceptor of even valued binary strings is plotted in fig. 1 together with the prefix tree acceptor for the sample $S = \{\lambda, 00, 10, 110\}$

If π is a partition of the set Q of nodes of T, $\pi(T)$ is the automaton obtained by merging the nodes in the same block of the partition. For instance, in fig. 1, the canonical acceptor may be obtained from the prefix tree acceptor by merging states labelled with

numbers in the same block of the partition $\tilde{\pi} = (\{1,2,4,6,7\},\{3,5\})$. Indeed, this is a particular example of a general well known fact: given a large enough sample S, there exists a partition $\tilde{\pi}$ of the nodes in T, such that $\tilde{\pi}(T)$ coincides with $M(L)$, the canonical acceptor. Therefore, the problem of identifying L is reduced to the simpler one of obtaining the partition $\tilde{\pi}$.

The language L_0 accepted by the PTA is always finite and coincides with the positive part of the sample set: $L_0 = S_+$. Any partition π leads to an enlarged accepted language $L_\pi \supset L_0$. Therefore, we are looking for a partition not including negative examples from S_-:

$$L_\pi \cap S_- = \emptyset. \tag{6}$$

The number of possible partitions in T grows exponentially with its size $t = |T|$, but a reasonable way to look for $\tilde{\pi}$ can be found. First, let us define an order of the nodes of T. The numerable set $\mathcal{A}^*$ —and therefore any language contained in $\mathcal{A}^*$— may be lexicographically ordered[2]. This allows one to define a similar order for the nodes of the automaton, as each node may be assigned the (lexicographically) first string leading from q_1 to that node. We will assume that the subindex i in q_i corresponds to this lexicographic order, as in fig. 1. Now, in order to find $\tilde{\pi}$, proceed as follows: merge (only if (6) still holds) nodes q_i and q_j, varying the subindex j from 2 to t and then, for every j, changing i from 1 to $j-1$. In this way at most $\frac{1}{2}t(t-1)$ comparisons are done while convergence to the canonical acceptor is guaranteed. For a formal proof, see [11]. Convergence is achieved whenever S_- is large enough to reject any wrong merge and then, $M(L)$ is produced as output. Thus, once S_+ and S_- are large enough, the hypothesis automaton cannot be changed by adding new examples to S and identification is reached.

A similar procedure for stochastic languages, preserving the properties of identification and polinomial time complexity, is desirable. This is the subject of the next section.

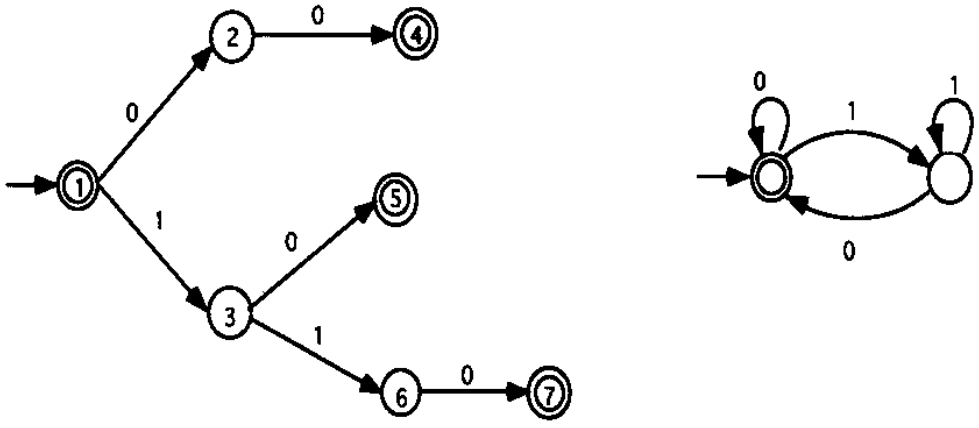

Figure 1: A prefix tree and the canonical acceptor for the regular set $(0+1)^*0$. Final states are doubled circled. Numbering of nodes follows the lexicographic order

[2]Sorted by length and then by alphabetical order within every length.

```
algorithm ALERGIA
input:
        S :  sample set of strings
        α :  1-confidence level
output:
        stochastic DFA
begin
        A = stochastic Prefix Tree Acceptor from S
        do (for j = successor( first node(A)) to lastnode(A))
           do(for i = firstnode(A) to j)
             if compatible(i,j)
               merge(A,i,j)
               determinize(A)
               exit (i-loop)
             end if
           end for
        end for
        return A
end algorithm
```

Figure 2: Algorithm ALERGIA.

4 The Algorithm

In the case of SRL, there are no negative examples in S, but the probability of appearance of every string follows a well defined distribution. Our algorithm ALERGIA takes advantage of this feature and performs merging of states when the resulting automaton is compatible with the observed frequencies of the strings in S.

The algorithm first builds the prefix tree T from S and evaluates at every node the relative frequencies of the outgoing arcs, incorporating this information in T (see fig. 9). We will write as n_i the (experimental) number of strings arriving at node q_i, $f_i(a)$ the number of strings following arc $\delta_i(a)$, and $f_i(\#)$ the number of strings ending at node q_i. The quotients $f_i(a)/n_i$ and $f_i(\#)/n_i$ estimate the probabilities $p_i(a)$ and p_{if} respectively.

Later, the algorithm compares couples of nodes (q_i, q_j), varying j from 2 to t and i from 1 to $j-1$. Two nodes in the same block of the partition $\tilde{\pi}$ are said to be *equivalent* ($q_i \equiv q_j$). As T is built, equivalent nodes have equal outgoing transition probabilities for every symbol $a \in \mathcal{A}$ and the destination nodes must be equivalent too:

$$q_i \equiv q_j \Longrightarrow \forall a \in \mathcal{A} \begin{cases} p_i(a) = p_j(a) \\ \delta_a(i) \equiv \delta_a(j). \end{cases} \tag{7}$$

This provides a criterion in order to reject equivalence of nodes. However, experimental data are subjected to statistical fluctuations and equivalence must be accepted within a confidence range. In such case, the nodes will be called *compatible.*

A confidence range for a Bernoulli variable with probability p and observed frequency

f out of n tries is given by the Hoeffding bound[14]:

$$\left| p - \frac{f}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}} \quad \text{with probability larger than } (1-\alpha). \tag{8}$$

ALERGIA will reject equivalence if two estimated probabilities differ in an amount larger than the sum of confidence ranges (see fig. 3). In this way, the probability of a wrong rejection is kept below 2α because at least one of the estimations must lay out of its confidence range for them to be considered different. The check is done for the termination frequencies $f_i(\#)$ and for the frequencies of the outgoing arcs $f_i(a)$. Thus, $|\mathcal{A}| + 1$ comparisons are done at every node, being $|\mathcal{A}|$ the size of the alphabet. If two nodes are found to be similar, all destination nodes are checked recursively, as shown in fig. 4 (algorithm `compatible`).

algorithm `different`
input:
`n,n':` `number of strings arriving at each node`
`f,f':number of strings ending/following a given arc`
output:
`boolean`
begin
return $\left| \frac{f}{n} - \frac{f'}{n'} \right| > \sqrt{\frac{1}{2} \log \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}} \right)$
end algorithm

Figure 3: Algorithm `different` checks similitude of observed frequencies.

Recursion in algorithm `compatible` involves only a finite number of calls. Indeed, due to the order followed within the merging process, when (q_i, q_j) are compared, q_j is always the root of a subtree of the PTA and therefore, the language it generates is finite. In other words, there is no loop in the q_j-subtree and the recursion takes always a finite time. Another important point concerns indeterminism. When q_i and q_j are found to be compatible, q_i and q_j are merged and the resulting automaton could in principle be indeterministic. In practice, this is not the case because the successors responsible for the indetermination are merged too. The reason for this comes from the recursive character of `compatible`: the respective a-successors of two compatible nodes are also compatible. This feature also ensures that the defined (lexicographic) order of the nodes is preserved during the merging process, as the automaton remains deterministic. Finally, every time a merge is performed, the frequencies f_i and the numbers n_i are recalculated, consistently with the fact that more information is available at each node. A schematic representation of ALERGIA is shown in fig. 2. A detailed example showing how the whole process works can be found in the Appendix.

```
algorithm compatible
input:
      i, j : nodes
output:
      boolean
begin
      if different(n_i, f_i(#), n_j, f_j(#))
        return false
      endif
      do(∀a ∈ A)
        if different(n_i, f_i(a), n_j, f_j(a))
          return false
        end if
        if not compatible(δ(i, a), δ(j, a))
          return false
        end if
      end do
      return true
end algorithm
```

Figure 4: Algorithm `compatible` checks $q_i \equiv q_j$.

5 Convergence of the Algorithm

In this section we will discuss the convergence of the algorithm. To this respect, the key point is the probability of finding the correct partition $\tilde{\pi}$ leading to the canonical acceptor. There are two different kinds of error where the algorithm could fail when looking for $\tilde{\pi}$:

1. (type α) rejection of compatibility between two equivalent nodes,
2. (type β) merge of two non-equivalent nodes.

Assume that S has a size $s = |S|$ large enough in the sense of section 3, while T —the prefix tree acceptor of S— is of size $t = |T|$ and the target (canonical) acceptor M has size $m = |M|$. Starting from T the algorithm should perform $t - m$ merges in order to successfully output M. Therefore, the global probability α_g for the first kind of error is bounded by the product $2\alpha(|\mathcal{A}| + 1)t$, being α the parameter used in **different** (fig. 3). If one wants to keep α_g negligibly small as the size of the sample becomes large, one may take $\alpha = kt^{-1}$ with k a small constant. The value of k has a smooth influence on the results, as eq. (8) depends on $\log k$.

On the other hand, two non-equivalent nodes can (incorrectly) be found to be compatible if the difference of the observed frequencies is smaller than the confidence range. Once errors of type α are negligible, the resulting automaton must be a partition of M. In particular, the partition will be the trivial one and **ALERGIA** will output M if for every couple of blocks B_i and B_j of the partition $\tilde{\pi}$ of T there exist two incompatible nodes $q_i \in B_i$ and $q_j \in B_j$. Therefore, an upper bound for β_g is given by the probability that

an error occurs when comparing representatives of each block. For this purpose, we may just select the first (in lexicographic order) node of the block as its representative. In this way, at most $\frac{1}{2}m(m-1)(|\mathcal{A}|+1)$ evaluations are needed in order to give an upper limit for β_g, each of those may contribute to β_g but all of them decrease with the size s. Therefore, β_g tends to zero as s grows. Recall that the error range behaves like:

$$\epsilon = \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}}+\frac{1}{\sqrt{n'}}\right) \tag{9}$$

where n and n', the number of strings arriving at each node, grow linearly with s. Therefore, ϵ tends to zero as s grows, even if, as proposed here, α changes with t, as t cannot grow faster than s.

It is not difficult to write an upper bound for β_g. When comparing $\hat{f}_1 = f_1/n_1$ and $\hat{f}_2 = f_2/n_2$, the expected value and variance of the difference $\delta\hat{f} = \hat{f}_1 - \hat{f}_2$ are:

$$\begin{aligned} \mathrm{E}(\delta\hat{f}) &= \delta p = p_1 - p_2 \qquad (10)\\ \mathrm{Var}(\delta\hat{f}) &= \mathrm{Var}(\hat{f}_1) + \mathrm{Var}(\hat{f}_2) = \\ &= \frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2} \leq \frac{1}{4n_1} + \frac{1}{4n_2}. \qquad (11) \end{aligned}$$

On the other hand, the probability $\beta = p(|\delta f| < \epsilon)$ that the observed difference is compatible with zero is smaller than $p(|\delta f - \delta p| > |\delta p| - \epsilon)$. Thus, using Chebychev's inequality[15], $\beta \leq B$ with:

$$B = \begin{cases} (|\delta p| - \epsilon)^{-2}\mathrm{Var}(\delta f) & \text{if } \epsilon < |\delta p| \text{ and } \mathrm{Var}(\delta f) < (\delta p - \epsilon)^2 \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

where B vanishes with s, because $\mathrm{Var}(\delta f)$ tends to zero (and so does ϵ). This bound B has to be evaluated less than $\frac{1}{2}m(m-1)(\mathcal{A}+1)$ times, and in all of them it tends to zero as the sample grows. Therefore, β_g vanishes in the limit of large sample sets.

It is also possible to choose a different functional dependence of α on t so that α_g also vanishes. However, this slows down convergence and the samples needed become larger. On the other hand, even if α_g is not very small, one obtains automata which are more complicated but equivalent (in the sense that the language is correctly identified) to the canonical one.

6 Results and Discussion

The performance of the algorithm has been tested with a variety of grammars. For each grammar, different samples were generated by the canonical stochastic automaton of the grammar and given as input for `ALERGIA`. For instance, the Reber grammar[16] of fig. 5 has been used in order to compare `ALERGIA` with previous works on neural networks which used this grammar as check[9].

In fig. 6 we plot the average number of nodes in the automaton found by `ALERGIA` as a function of the size of the sample set generated by the Reber grammar. The number of states is a measure of the complexity of the hypothesis. As seen in the figure, this

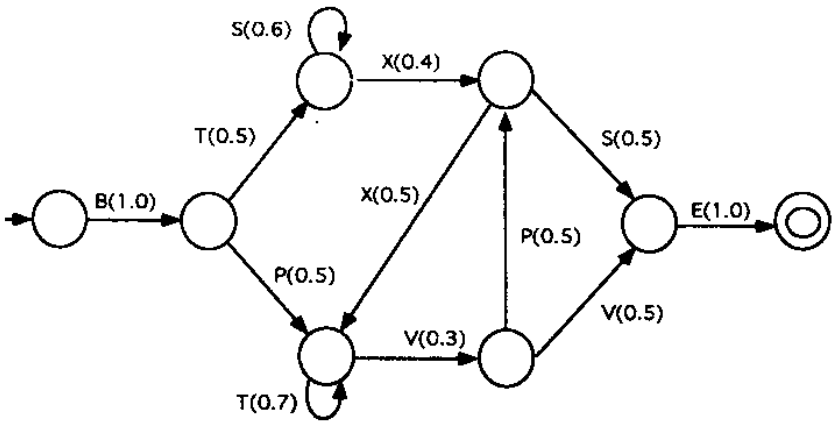

Figure 5: SFA corresponding to the Reber grammar

number always converges to the right value when the sample is large enough. We also checked that the structure of the automaton was correctly inferred. For small samples, the algorithm tends to propose hypothesis which are too complicated. However, when enough information is available it always finds the correct structure. The number of examples needed to achieve convergence is relatively small (about five hundred). This number compares rather favourably with the performance of recurrent neural networks[9] which cannot guaranty convergence for this grammar even after tens of thousands of examples.

In fig. 7, the time needed by the algorithm is plotted as a function of the number of examples in S. Although the temporal complexity could be in an extreme case be cubic, all the experiments showed a linear dependence. The algorithm proves very fast even for huge sample sets.

7 Conclusions

An algorithm has been proposed which identifies any stochastic regular language. Identification is achieved from stochastic samples of the strings in the language, and no information of the strings not belonging to the language is used. Experimentally, the algorithm needs very short times and comparatively small samples in order to identify the regular set. Even for large samples, only a linear time is needed (about one minute for a sample containing one million examples running on a Hewlett-Packkard 715). The algorithm is suitable for recognition tasks where noisy examples or random sources are common. In this line, applications to speech recognition problems are planned.

Acknowledgments

The authors want to acknowledge useful suggestions from E.Vidal and M.L.Forcada.

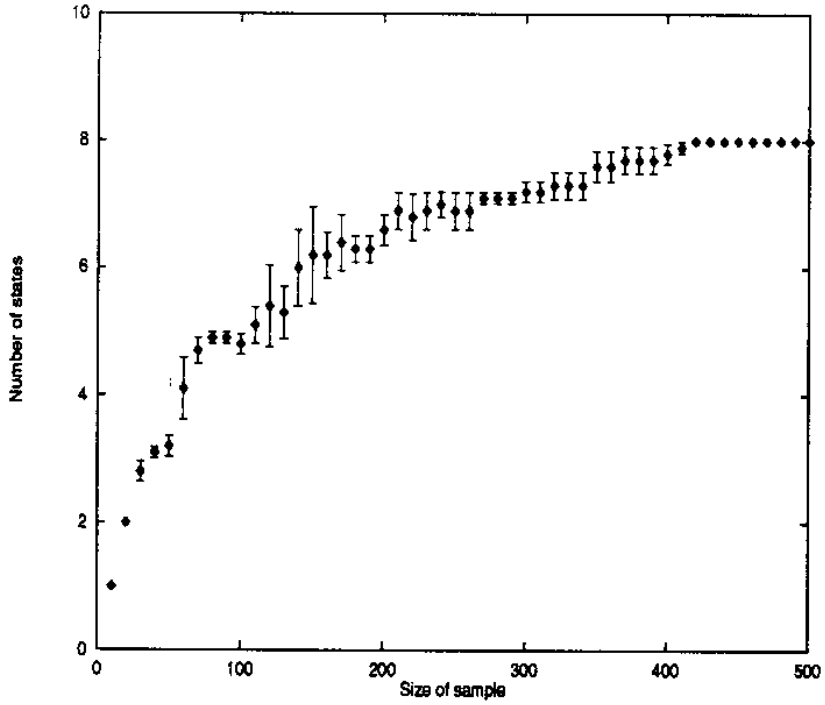

Figure 6: Number of nodes in the hypothesis for the Reber grammar as a function of the size of the sample.

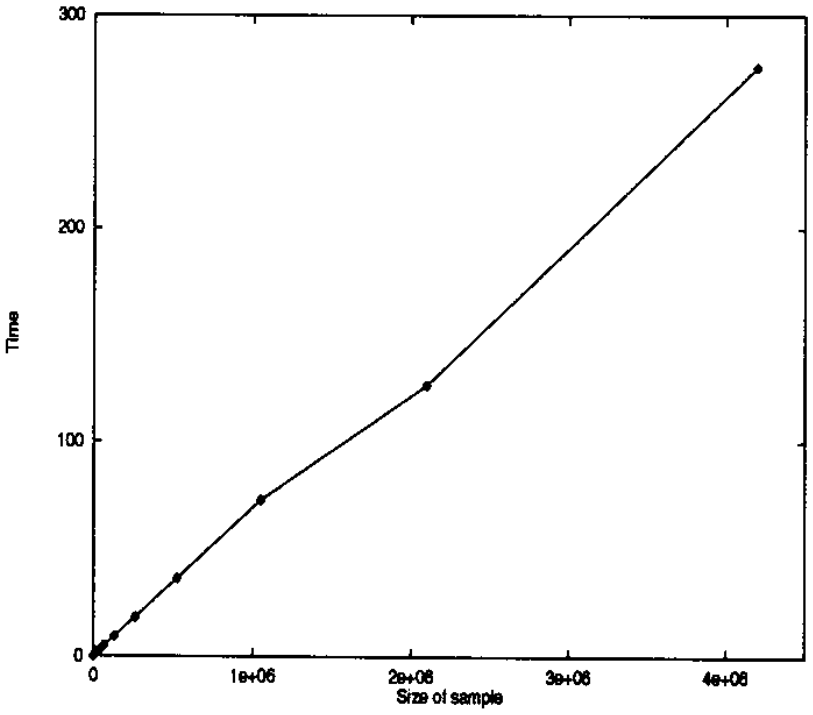

Figure 7: Time needed by our implementation of ALERGIA running on a Hewlett-Packard 715 as a function of the size of the sample.

References

[1] E.M. Gold: Complexity of Automaton Identification from Given Data. *Information and Control* **37** (1978) 302–320.

[2] D. Angluin: Identifying Languages from Stochastic Examples. Internal Report YALEU /DCS /RR–614 (1988).

[3] F.J. Maryanski and T.L. Booth: Inference of Finite-State Probabilistic Grammars. *IEEE Transactions on Computers* **C26** (1977) 521–536.

[4] A. van der Mude and A. Walker: On the Inference of Stochastic Regular Grammars. *Information and Control* **38** (1978) 310–329.

[5] A.W. Smith and D. Zipser: Learning Sequential Structure with the Real-Time Recurrent Learning Algorithm. *International Journal of Neural Systems* **1** (1989) 125–131.

[6] J.B. Pollack: The Induction of Dynamical Recognizers. *Machine Learning* **7** (1991) 227–252.

[7] C.L. Giles: Learning and Extracting Finite State Automata with Second Order Recurrent Neural Networks. *Neural Computation* **4** (1992) 393–405.

[8] R.L. Wartous and G.M. Kuhn: Induction of Finite-state Languages Using Second-Order Recurrent Networks. *Neural Computation* **4** (1992) 406–414.

[9] M.A. Castaño, F. Casacuberta, E. Vidal: Simulation of Stochastic Regular Grammars through Simple Recurrent Networks, in: *New Trends in Neural Computation* (Eds. J. Mira, J. Cabestany and A. Prieto). Springer Verlag, Lecture Notes in Computer Science **686** (1993) 210–215.

[10] A. Stolcke and S. Omohundro: Hidden Markov Model Induction by Bayesian Model Merging. To appear in: *Advances in Neural Information Processing Systems 5* (C.L. Giles, S.J. Hanson and J.D. Cowan eds.) Morgan Kaufman, Menlo Park, California (1993).

[11] J. Oncina and P. García: Inferring Regular Languages in Polynomial Time, in: *Pattern Recognition and Image Analysis* (N. Pérez de la Blanca, A. Sanfeliu and E. Vidal eds.) World Scientific (1992).

[12] K.S. Fu: *Syntactic Pattern Recognition and Applications.* Prentice Hall, Englewood Cliffs, N.J. (1982).

[13] J.E. Hopcroft and J.D. Ullman: *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Massachusetts (1979).

[14] W. Hoeffding: Probability inequalities for sums of bounded random variables. *American Statistical Association Journal* **58** (1963) 13–30.

[15] W. Feller: *An introduction to probability theory and its applications.* John Wiley and Sons, New York (1950)

[16] A.S. Reber: Implicit Learning of Artificial Grammars. *Journal of Verbal Learning and Verbal Behaviour* **6** (1967) 855–863.

A Appendix

We will present a simple example to show how ALERGIA works. Assume that the automaton depicted in fig. 8 outputs the sample:

$$S = \{110, \lambda, \lambda, \lambda, 0, \lambda, 00, 00, \lambda, \lambda, \lambda, 10110, \lambda, \lambda, 100\}.$$

Only for illustration purposes, the parameter α will be arbitrarily set to 0.8. With this choice one has:

$$\gamma := \sqrt{\frac{1}{2}\log\frac{2}{\alpha}} \simeq 0.67$$

The algorithm starts by building the PTA, as shown in fig. 9. Each node is labelled with a number corresponding to its lexicographic order. In brackets, the number of strings arriving and terminating at that node are plotted. Every arc has a label with the symbol (0 or 1) inducing the transition, and in brackets appears the number of strings using that arc. Next, the algorithm checks if nodes q_2 and q_1 (labelled 2 and 1) are equivalent. This requires a comparison of the probabilities coming from these nodes. For instance, the termination probabilities are found to be similar:

$$|p - p'| = \left|\frac{1}{3} - \frac{1}{9}\right| \simeq 0.26 < \gamma(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}}) \simeq 0.55$$

Also, the outgoing transitions have similar probabilities:

$$|p - p'| = \left|\frac{2}{3} - \frac{3}{15}\right| \simeq 0.46 < 0.55$$

In addition, equivalence of q_2 and q_1 also requires (due to the recursive definition) compatibility between destination nodes (namely q_4 and q_2). Proceeding in an analogous way, this is found to be the case and therefore $q_2 \equiv q_1$. The result after merging these two states is plotted in fig. 10.

In the next step the algorithm tries to merge the couple (q_3, q_1). However, they are found to be non-compatible when the termination probabilities are compared (0.6 and 0.0 respectively, whose difference is larger than 0.53)

Afterwards, comparison of q_5 and q_1 is done. The following similarities are accepted: $q_7 \equiv q_1$, $q_8 \equiv q_3$, $q_{10} \equiv q_6$, and $q_{11} \equiv q_9$. After the merges are performed one finds the automaton in fig. 11.

The merging of nodes q_6 and q_1 is rejected due to the difference in the transition probabilities labelled with 0 (0.75 being larger than 0.61). Instead, $q_6 \equiv q_3$ will be accepted, and the algorithm ends with the automaton plotted in fig. 12 as hypothesis. The estimated probabilities are shown in fig. 13 to be compared with 8.

The structure is the same but the probabilities have been only roughly estimated, due to the small size of the sample. Obviously one needs larger samples in order to find more accurate probabilities and in order to choose a reasonable confidence level (α).

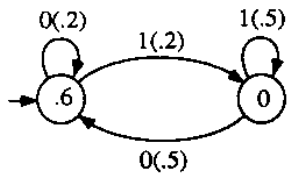

Figure 8: Stochastic finite automaton

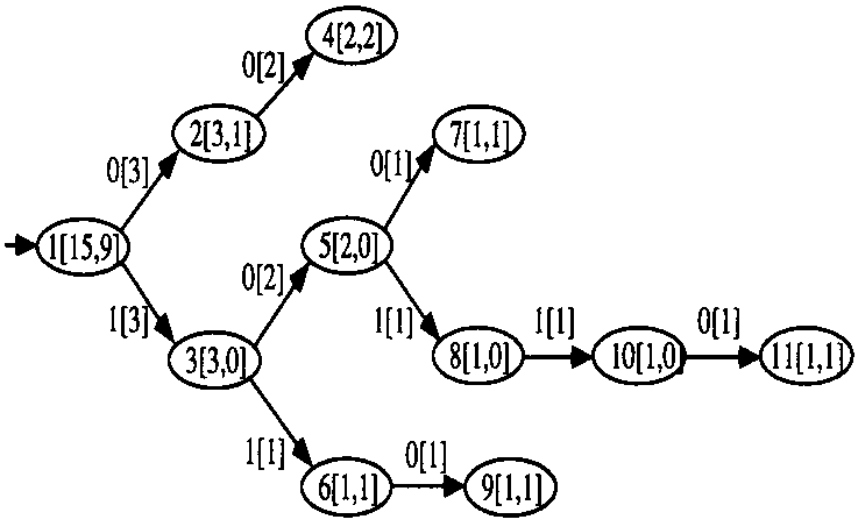

Figure 9: Prefix tree acceptor for the sample S ={ 110, $\lambda, \lambda, \lambda$, 0, λ, 00, 00, $\lambda, \lambda, \lambda$, 10110, λ, λ, 100}.

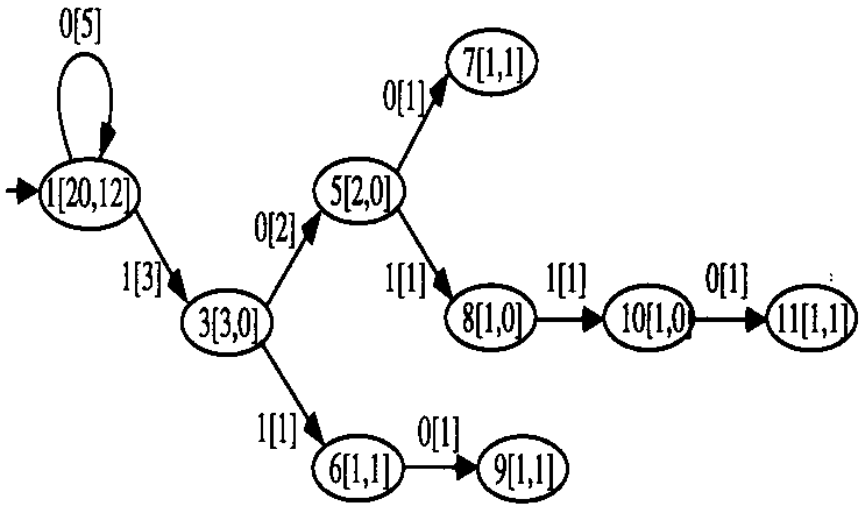

Figure 10: Prefix tree acceptor after merging q_2 and q_1.

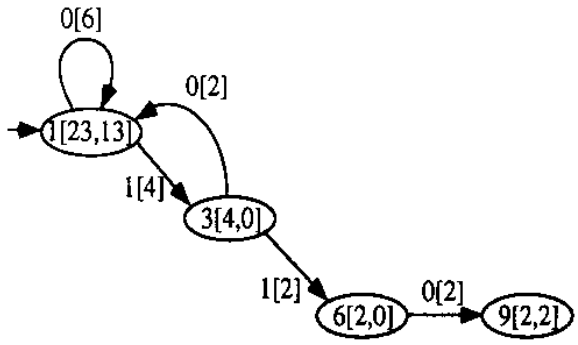

Figure 11: Automaton after merging q_5 and q_1.

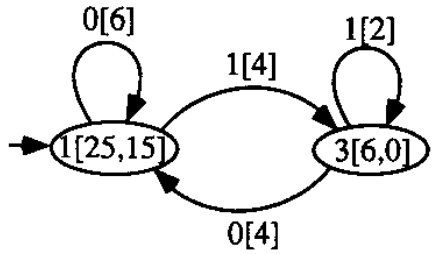

Figure 12: Automaton after merging q_6 and q_3.

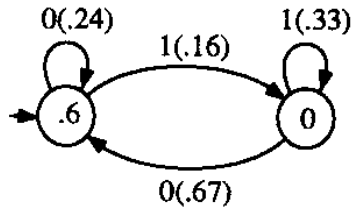

Figure 13: The final output.