A NEW MODEL FOR THE RECOGNITION OF STRINGS OF SYMBOLS

by

Keiichi Abe

Memorandum No. UCB/ERL M78/10

6 February 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A long paper

A NEW MODEL FOR THE RECOGNITION OF STRINGS OF SYMBOLS

Keiichi Abe[†]

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

In this paper the author proposes a new model for the recognition of strings of symbols, which is named Penalty Finite State Automaton (PFSA). The relations between PFSA and some of the models which have already been proposed, that is, finite state automaton (FSA), error-correcting parser for a finite state automaton (ECFSA), stochastic finite state automaton (SFSA), and fuzzy finite state automaton (FFSA) are discussed. Firstly, it is shown that our model PFSA is a generalization of ECFSA. Secondly, there is an interesting analogy among SFSA, PFSA and FFSA, and as a result, PFSA can be looked upon as an intermediate model between SFSA and FFSA. Therefore, we can expect that PFSA should be simpler than SFSA while it should be more powerful than FFSA, at least in some applications.

Parsing algorithms and grammatical inference of PFSA are also investigated. An application to the recognition of strings of symbols extracted from chromosome data shows the possible effectiveness of this model, though its performance must be further studied.

Research sponsored by the National Science Foundation Grant ENG76-84522.

[†]On leave from the Department of Information Science, Shizuoka University, Japan.

1. Introduction

In syntactic or linguistic pattern recognition, a pattern may be represented by a string, a tree, or a graph of pattern primitives. We confine our interest to a string of symbols. There have been studied many models for the recognition of strings of symbols [1-10]. For the sake of simplicity, we will discuss only at finite state language level, though the discussion might be hopefully generalized to context-free or context-sensitive language level.

The simplest model for the recognition of strings of symbols is a finite state automaton (FSA). However, FSA seems to be too rigid for the application to pattern recognition. It behaves on the all-or-nothing basis. For the purpose of pattern recognition it is desirable to introduce some real-valued measure in the model. One way for accomplishing that is to introduce error transformations between symbols. This leads us to the second model, an error-correcting parser for finite state automation (ECFSA) [8]. Two other important models for defininig a real-valued measure on the space of symbol strings are stochastic finite state automaton (SFSA) and fuzzy finite state automaton (FFSA).

We will describe those models which have been already proposed in the next section. Then we will propose a new model named Penalty Finite State Automaton (PFSA) in Section 3. The relations between PFSA and other models will be discussed in Section 4. Parsing and grammatical inference of PFSA and its application to the recognition of chromosomes are discussed in Sections 5 and 6.

2. Some Former Models for the Recognition of Strings of Symbols

Let us start with the definition of a finte state automaton (FSA).

Definition 1 A non-deterministic finite state automaton is a 5-tuple[†]

$$A = (K,\Sigma,\delta,S_o,F) \tag{1}$$

where K is a set of states;

Σ is a set of input symbols;

δ is a mapping from $K\times\Sigma$ to 2^K;

$S_o \in K$ is the initial state;

$F \subset K$ is the set of final states.

A FSA defines a dichotomy on the symbol string space Σ^*. Given an input string $\xi = \xi_1\xi_2\ldots\xi_n \in \Sigma^*$ ($\xi_i \in \Sigma$ for $1 \leq i \leq n$), if a sequence of states $t_o,t_1,\ldots,t_n$ satisfying the following equation is found,

$$t_o = s_o,\ t_n \in F$$

$$\delta(t_{i-1},\xi_i) \ni t_i \text{ for } i = 1,\ldots,n \tag{2}$$

then we can classify ξ into the positive class; otherwise into the negative class.

The set of string ξ's to be classified in the positive class is called the language accepted by FSA A and denoted by L(A). The family of L(A)'s for all FSA's is the well-known family of the finite state languages (FSL's).

By introducing three types of error transformations between sybmols we get the second model, an error correcting parser for FSA (ECFSA) [8].

[†]We will follow the common notations in automata theory.

Definition 2 For any string $\xi \in \Sigma^*$, we assume that we can apply any of the following three error transformations:

(1) substitution T_s

$$\xi_1 a \xi_2 \vdash\!\!\xrightarrow{T_s}\!\! \xi_1 b \xi_2 \quad \text{for any } \xi_1, \xi_2 \in \Sigma^* \text{ and any } a,b \in \Sigma (a \neq b) \tag{3}$$

(2) deletion T_D

$$\xi_1 a \xi_2 \vdash\!\!\xrightarrow{T_D}\!\! \xi_1 \xi_2 \quad \text{for any } \xi_1, \xi_2 \in \Sigma^* \text{ and any } a \in \Sigma \tag{4}$$

(3) insertion T_I

$$\xi_1 \xi_2 \vdash\!\!\xrightarrow{T_I}\!\! \xi_1 a \xi_2 \quad \text{for any } \xi_1, \xi_2 \in \Sigma^* \text{ and any } a \in \Sigma \tag{5}$$

Here the notation $\xi \vdash\!\!\!\rightarrow \eta$ means that ξ can be transformed into η.

We may attach a cost to each of the error transformations. In general, such a cost may be dependent on the transformed symbols a and b. Therefore, in case (1) the cost would be represented by $C_S(a,b)$, in case (2) by $C_D(a)$, and case (3) by $C_I(a)$. We assume that those costs are all positive and $C_S(a,b) \leq C_D(a) + C_I(b)$.

We can transform any string $\xi \in \Sigma^*$ to any other string $\eta \in \Sigma^*$ by applying the above transformations repeatedly. In this process we assume that the cost of transforming ξ to η is the sum of the costs of the individual transformations.

Definition 3 The distance between two strings $\xi, \eta \in \Sigma^*$, denoted by $d(\xi,\eta)$, is defined as the smallest cost of the transformations which derive η from ξ.

Note that if $C_S(a,b) = 1$ for all $a,b \in \Sigma$ and $C_D(a) = C_I(a) = 1$ for all $a \in \Sigma$, then $d(\xi,\eta)$ is defined as the smallest number of error transformations required to derive η from ξ.

Suppose we are given a FSA A and the error transformations with their costs as defined above. Then they define a metric on the string space Σ^*, that is, a mapping from Σ^* to R^+, where R^+ denotes $[0,\infty]$.[†] This is called the distance between ξ and the language L(A), namely,

$$d(\xi,L(A)) = \min_{\eta \in L(A)} d(\xi,\eta) \tag{6}$$

Of course for a string $\xi \in L(A)$, $d(\xi,L(A)) = 0$. An efficient way of computing $d(\xi,L(A))$ for a given input string ξ is called error-correcting parser for the FSA (ECFSA).

Another model for defining a metric over the string space Σ^* is a stochastic finite state automaton (SFSA).

Definition 4 A stochastic finite state automaton is a 6-tuple

$$A = (K,\Sigma,\delta,\gamma,F,\theta) \tag{7}$$

where K is a set of states;

Σ is a set of input symbols;

δ is a mapping from $K\times K\times\Sigma$ to $[0,1]$;

γ is a mapping from K to $[0,1]$;

$F \subset K$ is the set of final states;

θ is a threshold value in $[0,1]$.

γ represents an initial probability distribution among the states in K and $\delta(S,S',\sigma)$ is the transition probability from state S to state S' by reading input symbol σ. Then after reading a given input string $\xi = \xi_1\xi_2\ldots\xi_n$, the probability of that the state of the automaton A

[†]We look upon ∞ as a mere symbol. In practice, it should be interpreted as a sufficiently large value.

is one of the states in F is

$$P(\xi) = \sum_{t_n \in F} \sum_{t_0, t_1, \ldots, t_{n-1} \in K} \gamma(t_0) \prod_{i=1}^{n} \delta(t_{i-1}, t_i, \xi_i) \qquad (8)$$

In order to classify ξ into one of two classes, we should compare $P(\xi)$ with the threshold θ. If $P(\xi) \geq \theta$ then we decide that ξ belongs to the positive class, otherwise to the negative class.

We can make a simplification assuming a unit probability distribution for the initial state probabilities γ, that is, $\gamma(S_0) = 1$ for some $S_0 \in K$ and $\gamma(S) = 0$ for any $S \neq S_0$ in K. Then (8) reduces to

$$P(\xi) \sum_{t_n \in F} \sum_{t_1, \ldots, t_{n-1} \in K} \prod_{i=1}^{n} \delta(t_{i-1}, t_i, \xi_i) \qquad (9)$$

where $t_0 = S_0$.

We may consider $\delta(t_{i-1}, t_i, \xi_i)$ in (9) not as a probability but as a possibility according to what L. A. Zadeh has proposed [11].† Then Σ and Π in (9) are substituted by max and min, and we have

$$P'(\xi) = \max_{t_n \in F} \max_{t_1, \ldots, t_{n-1} \in K} \min_{\{i \mid i=1,\ldots,n\}} \delta'(t_{i-1}, t_i, \delta_i) \qquad (10)$$

where $t_0 = S_0$, $\delta'(t_{i-1}, t_i, \xi_i)$ is a transition possibility from state t_{i-1} to state t_i by reading an input symbol ξ_i, and $P'(\xi)$ is the final possibility of the acceptance of input string ξ. This model is called fuzzy finite state automaton (FFSA).

3. Penalty Finite State Automation

In this section a new model for the recognition of strings of symbols is proposed. This model is named Penalty Finite State Automaton (PFSA).

† In fact, Zadeh's proposal is to consider δ not as a single valued function but as a fuzzy function (of Type 2). However, in order to keep valid the analogy between the models that will be discussed later, we consider δ as single valued here.

Definition 5 A PFSA is a 6-tuple

$$A = (K,\Sigma,\omega,S_0,F,\theta) \tag{11}$$

where K is a set of states;

Σ is a set of input symbols;

ω is a mapping from $K\times K\times\Sigma$ to $[0,\infty]$;

$S_0 \in K$ is the initial state;

$F \subset K$ is the set of final states;

θ is a threshold value in $[0,\infty]$.

We call $\omega(S,S',\sigma)$ "penalty associated with the transition from state S to state S' by reading input symbol σ". Note that the transitions are defined as non-deterministic. Given an input string $\xi = \xi_1 \ldots \xi_n$ we can define the minimal sum of penalties required for taking the automaton from the initial state S_0 to one of final states in F by reading the whole string ξ. This can be expressed as

$$W(\xi) = \min_{t_n \in F} \min_{t_1,\ldots,t_{n-1} \in K} \sum_{i=1}^{n} \omega(t_{1-1},t_i,\xi_i) \tag{12}$$

where $t_0 = S_0$. Again we will have a dichotomy of the string space Σ^* by comparing $W(\xi)$ with the threshold θ: if $W(\xi) \leq \theta$ then we put ξ into the positive class, otherwise into the negative class. We refer to the set of strings in the positive class as "the language accepted by PFSA A".

If we exclude the threshold θ from definition 5, we have a transformer which assigns a value $W(\xi)$ to each input string $\xi \in \Sigma^*$. Let us call it a Penalty Finite State Transformer (PFST).

Example 1 Suppose $K = \{S_0, S_1, S_2\}$, $\Sigma = \{a,b\}$, $F = \{S_2\}$, $\delta(S_0,S_0,a) = 1$, $\delta(S_0,S_1,b) = 0$, $\delta(S_1,S_1,b) = 2$, $\delta(S_1,S_2,a) = 0$, $\delta(S_1,S_2,b) = 1$, $\delta(S_2,S_2,a) = 0$, and $\delta(S,S',\sigma) = \infty$ for any other combinations of S, S', σ. This defines a PFST shown in Fig. 1, where the transitions with infinite penalty are omitted.

If we apply a threshold, say $\theta = 2$, we have a PFSA. It is easily seen that its positive class consists of such strings that

$$\{aabaa^{\ell}, abaa^{\ell}, abba^{\ell}, baa^{\ell}, bba^{\ell} | \ell = 0,1,\ldots\}$$

PFST (and analogous transformers derived from SFSA and FFSA) can be used in multi-class pattern classification. Let us assume there are m pattern classes. We can construct m PFST's $A_1, A_2, \ldots, A_m$, one for each class. By putting a given input string ξ in all PFST's simultaneously, we get m minimal penalties $W_1(\xi), W_2(\xi), \ldots, W_m(\xi)$ calculated by (12). By finding the smallest value of them, say $W_k^*(\xi)$, we then decide that the input string ξ belongs to class k* (Fig. 2).

4. Relations between PFSA and other models

4.1 PFSA versus FSA and ECFSA models

We will show that (1) the capability of PFSA is the same as that of FSA from the theoretical point of view and that (2) PFST is a generalization of ECFSA. We will not present the proofs here.

Theorem 1 The family of the languages accepted by PFSA's is the same as the family of FSL's.

This fact does not imply that the performances of PFSA and FSA are the same in practice. A minimum-state FSA equivalent to the PFSA in example 1 has seven states. In fact, we can expect that in general a PFSA should have fewer states than an equivalent FSA.

Theorem 2 The family of ECFSA's is a proper subset of the family of PFST's.

This theorem states the following: supposing we are given an arbitrary ECFSA, we can construct an equivalent PFST in the sense that the PFST defines the same metric on $\Sigma^* - \{\lambda\}$ as the given ECFSA does. On the other hand, we cannot always construct an equivalent ECFSA to an arbitrarily given PFST. Roughly speaking, the reason for that is the fact that the costs of error transformations are defined as position-independent (i.e. independent of the index i of a symbol ξ_i in an input string ξ). On the other hand, the penalties of a PFST can be dependent not only on the position of an input symbol but also on the whole history of the input string read so far.

There is another interpretation of the difference of performances between PFST and ECFSA. In ECFSA the costs of error transformations are defined on $\Sigma' \times \Sigma'$, where Σ' denotes $\Sigma \cup \{\lambda\}$, while in PFST the penalties are defined on $K \times \Sigma$. If we consider a FSA as a grammar, the cost of ECFSA are defined on a pair of terminal symbols, while the penalties of PFST are defined on the nonterminal-terminal symbol pair. It is easy to reconstruct a finite state grammar into an equivalent grammar in which each terminal symbol is derived from exactly one nonterminal symbol. This gives us an informal proof of one side of theorem 2, that is, of the fact that the family of PFST's includes the family of ECFSA's, though we still need some trick to deal with the empty string λ.

Thus we can expect that PFST should be a more powerful model than ECFSA for the recognition of strings of symbols and that we could have more flexibility in the design of such recognizers.

Example 2 Suppose we are given a set of symbols $\Sigma = \{a,b\}$, a FSA shown in Fig. 3(a), and the costs of error transformations $C_S(x,y) = 1$ for any $x,y \in \Sigma$ and $C_D(x) = C_I(x) = 1$ for any $x \in \Sigma$. Then we can construct an equivalent PFST with λ-transitions as shown in Fig. 3(b). Next we can eliminate the λ-transitions and obtain an equivalent PFST shown in Fig. 3(c). On the other hand, PFST shown in Fig. 1 has no equivalent ECFSA.

4.2 PFSA versus SFSA and FFSA models

No explicit result has been found about the relation among the PFSA, SFSA, and FFSA models. Nevertheless, eqs. (9), (10), (12) show an interesting analogy among them. By substituting Σ for Π and min for Σ in (9) of SFSA, we obtain (12) of PFSA. If we substitute max for min and min for Σ in (12) of PFSA, we obtain (10) of FFSA. This analogy can be interpreted as follows. Equation (9) says that we have to calculate first the products of transition probabilities $\delta(t_{i-1},t_i,\xi_i)$ along the paths $t_0, t_1,\ldots,t_n$ starting with the initial state $t_0 = S_0$ and ending at one of the final states $t_n \in F$. Then we have to sum up those products over all possible such paths. If we would pay attention only to the path with the largest products of transition probabilities and neglect all the other paths, then this would be equivalent to having Σ in (9) substituted by max. On this assumption (9) and (12) are equivalent to each other by putting

$$\omega(t_{i-1},t_i,\xi_i) = -\log P(t_{i-1},t_i,\xi_i) \tag{13}$$

$$W(\xi) = -\log P(\xi) \tag{14}$$

In other words, using PFSA as a recognizer is equivalent to picking up only the most dominant transition path in SFSA and neglecting all

the other paths. The choice of summation rather than multiplication in PFSA was merely for making the calculations simpler.

The analogy between PFSA and FFSA can be interpreted in a similar way. A FFSA does not calculate the sum of penalties $\omega(t_{i-1},t_i,\xi_i)$ as in PFSA but pays attention only to the largest penalty value (note that we have to negate the values in order to substitute max for min).

These analogies show that PFSA is an intermediate model between SFSA and FFSA. It is simpler than SFSA; yet more powerful than FFSA. The simplicity of PFSA compared with SFSA yields two advantages: faster parsing algorithms and easier grammatical inference. A PFSA can be used as a recognizer model in such cases that SFSA is too complicated to apply (especially in grammatical inference) and FFSA is too simple to obtain a good result unless the transition possibilities $\delta'(t_{i-1},t_i,\xi_i)$ are generalized to fuzzy functions.

5. Parsing and Grammatical Inference of PFSA

5.1 Parsing for PFSA

Parsing for a PFSA means the process of computing the minimum sum of penalties $W(\xi)$ in (12) for a given input string ξ and finding the correspondent sequence of states $t_0, t_1,\ldots,t_n$. We will propose here three different mehtods for such a computation: (1) dynamic programming method, (2) table lookup method, and (3) heuristic search method.

5.1.1 Dynamic Programming Method

We can compute $W(\xi)$, for a given $\xi = \xi_1\xi_2\ldots\xi_n$, by starting with initial conditions

$$\gamma(0,S_0) = 0$$
$$\gamma(0,S) = \infty \text{ for any } S \neq S_0 \text{ in } K \qquad (15)$$

and using the following recurrence relation:

$$\gamma(i,S) = \min_{S' \in K} [\gamma(i-1,S') + \omega(S',S,\xi_i)] \quad (i = 1,\ldots,n) \tag{16}$$

Calculating $\gamma(i,S)$ for all $S \in K$ and sequentially for $i = 1,\ldots,n$, we finally get

$$W(\xi) = \min_{S \in F} \gamma(n,S) \tag{17}$$

We can easily decide the sequence of states which corresponds to $W(\xi)$ by tracing the above steps backward and picking up the states which yield the minimum value in each step.

This computation process can be looked upon as a solution for the optimization problem (12) using dynamic programming. In general, the dyanmic programming method is simple to describe but it requires large memory and a lot of computation time. The latter defect might be avoided by using a special-purpose hardware perhaps with the ability of parallel computations.

5.1.2 Table Lookup Method

The dynamic programming method becomes very wasteful when most of the penalties attached to state transitions are infinite. In this case we can compute only the elements d(i,S)'s with finite value, put them in a table, and look up the table when they are needed for the next step of calculation. This method has been used by several authors [3,4,12].

5.1.3 Heuristic Search Method

Both dynamic programming method and table lookup method calculate d(i,S)'s parallelwise for all necessary states S in K for a fixed i, and then proceed to the next value of i (breadth-first search). Alternatively we can proceed the calculation for the most prospective sequence of

d(i,S)'s incrementing i at once (depth-first search). If we fail somewhere or get a feasible solution, then we have to backtrack. This type of calculation is called heuristic search for shortest path problem on a graph [13], or branch and bound method [14] in a more general terminology.

A few comments are worth noting. For a ECFSA dynamic programming method is not applicable as straight-forward as for a PFSA as shown here. Heuristic search method cannot be applied to SFSA because in SFSA the sum of the probabilities for all the transition paths must be calculated. These facts show an advantage of PFSA over the other models because in PFSA we can choose the fastest parsing algorithm among the three types of methods.

5.2 Grammatical Inference of PFSA

Grammatical inference of PFSA means to construct a PFSA which classifies correctly two given finite sets of sample strings — one to be classified into the positive class, the other into the negative class. This can be readily generalized to multi-class classification.

Grammatical inference of PFSA can be divided into two stages: grammatical inference of FSA (PFSA with all zero penalties) and adjustment of penalty values. The first stage, grammatical inference of FSA, is identical with that of other models, ECFSA or SFSA. There have been proposed several grammatical inference algorithms of FSA [2,9,10]. But they don't seem to work well for sample sets of limited number of relatively long strings, which are often the case in pattern recognition applications. Alternatively we can use a heuristic method of grammatical inference of FSA. A schematic diagram of this method is shown in Fig. 4.

Its basic idea is (1) to find the minimum discrepancy (that is, dissimilar substrings or, inversely, longest common subsequences [15]) between the new input string and a previous string and (2) to add the necessary transitions to FSA or merge the existing states into one state of FSA in order to make up the discrepancy. The distance between strings $d(\xi,\eta)$ is defined here by error transformations with the costs $C_S(x,y) = 2$ for all $x,y \in \Sigma$ and $C_I(x) = C_D(x) = 1$ for all $x \in \Sigma$.

The second problem in grammatical inference of PFSA is how to decide on the values of penalties. A few methods are possible:

(1) decision based on the distances between states (this leads to almost the same results as error transformations)

(2) using statistics of sample strings and eq. (13)

(3) training similar to that used in Perceptron or so

(4) indications by human beings

These methods and their combinations are yet to be further studied.

6. An Application to Chromosome Data

Lee and Fu have tried to classify chromosome images by stochastic context-free grammars [16,17]. In one of their experiments they used 29 chromosome samples. They first enhanced original digitized images of microscopic pictures, encoded boundaries of images, smoothed derived chain codes, and then extracted strings of primitive features. Those strings are shown in Table 1. They used 12 samples (2 median, 7 submedian, and 3 acrocentric chromosomes) for grammatical inference of stochastic context-free grammars. Then they tried to classify the other 17 samples (3 median, 8 submedian, 6 accrocentric) into three classes.

The same set of training sample strings and the same of test sample strings were applied to PFSA. The underlying FSA's were inferred as described in the previous section. Then the values of penalties were chosen based on a sort of distance between states of the FSA's. Training of the penalties was not needed because the training samples were all classified correctly with the penalties thus defined.

Both Lee-Fu's and the author's experiments and their results are summarized in Table 2. The increase of errors of the current experiment shown in the second row of the table seems to be caused by the scarcity of the training samples and an unbalance of the numbers of them in the three classes. For this reason, a simple weighting mechanism was introduced: the sum of penalties $W(\xi)$ for each class was weighted by the reciprocal of the numbers of training samples in that class. The result improved and became closer to Lee-Fu's result as shown in the third row of the table.

Considering that our model is finite state while Lee-Fu's model is context-free and that Lee-Fu also used structural information (indication of substrings which should be considered as a subunit) for the purpose of grammatical inference of their model, the overall result is very favorable to our current model (PFSA). Though this experiment is too small to draw a definite conclusion, it has, at least, shown a good performance of PFSA.

7. Conclusions

A new model for recognition of strings of symbols has been proposed. The study of relations between this model PFSA and other models suggests the possible usefulness of the model. An application for recognition

of strings of primitives derived from chromosome data supports this presumption. However, it seems necessary to make experiments with more large sample sets and to study grammatical inference of PFSA further.

Acknowledgement

Part of this study was made at Advanced Automation Research Laboratory of Purdue University. The author is grateful to Professor K. S. Fu who suggested him to use Dr. H. C. Lee's preprocessed chromosome data and gave him helpful suggestions and discussions.

References

[1] K. S. Fu, Syntatic Methods in Pattern Recognition, Academic Press, New York, 1974.

[2] K. S. Fu and T. L. Booth, "Grammatical inference: introduction and survey," Part I, IEEE Trans. on Systems, Man, and Cyber., Vol. SMC-5, No. 1, pp. 95-111 and Part II, ibid., No. 4, pp. 409-422, 1975.

[3] A. V. Aho and T. G. Peterson, "A minimum distance error-correcting parser for context-free languages," SIAM J. Comput., Vol. 1, No. 4, pp. 305-312, 1972.

[4] E. Persoon and K. S. Fu, "Sequential classification of strings generated by SCFG's," Intern'l J. of Comp. and Inform. Sciences, Vol. 4, No. 3, pp. 205-217, 1975.

[5] P. H. Swain and K. S. Fu, "Stochastic programmed grammars for syntactic pattern recognition," Pattern Recognition, Vol. 4, pp. 83-100, 1972.

[6] T. Huang and K. S. Fu, "Stochastic syntactic analysis for programmed grammars and syntactic pattern recognition," Computer Graphics and Image Processing, Vol. 1, pp. 257-283, 1972.

[7] E. Tanaka and K. S. Fu, "Error-correcting parsers for formal languages," Tech. Rept. TR-EE 76-7, School of Elec. Engrg., Purdue Univ., March 1976.

[8] K. S. Fu and S. Y. Lu, "A clustering procedure for syntactic patterns," IEEE Trans. on Systems, Man, and Cyber., Vol. SMC-7, No. 10, pp. 734-742, 1977.

[9] A. W. Bierman and J. A. Feldman, "On the synthesis of finite state machines from samples of their behavior," IEEE Trans. on Comput. Vol. C-21, No. 6, pp. 592-597, 1971.

[10] H. Enomoto, E. Tomita and S. Doshita, "Synthesis of automata that recognize given strings and characterization of automata by representative sets of strings," 1-st USA-Japan Computer Conf. Proc. p. 21, 1972.

[11] L. A. Zadeh, "Fuzzy sets and their relation to pattern classification and cluster analysis," Memorandum No. ERL-M607, Electronic Research Lab., College of Engrg., Univ. of California, Berkeley, October 1976.

[12] S. Y. Lu and K. S. Fu, "Error-correcting syntax analysis for tree languages," Tech. Rept., TR-EE 76-24, School of Elec. Engrg., Purdue Univ., July 1976.

[13] N. J. Nilsson, Problem Solving Methods in Artificial Intelligence, Chap. 3, McGraw-Hill, New York,1971.

[14] E. L. Lawler and D. E. Wood, "Branch and bound methods, a survey," Oper. Res., Vol. 149, No. 4, 1966.

[15] R. A. Wagner and M. J. Fisher, "The string-to-string correction problem," J. ACM, Vol. 21, No. 1, pp. 168-173, 1974.

[16] H. C. Lee and K. S. Fu, "Stochastic linguistics for picture recognition," Tech. Rept., TR-EE 72-17, School of Elec. Engrg., Purdue Univ., June 1972.

[17] H. C. Lee and K. S. Fu, "A stochastic syntax analysis procedure and its application to pattern classification," IEEE Trans. on Comp., Vol. C-21, No. 7, pp. 660-666, 1972.

Table 1. Strings derived from chromosome data by Lee and Fu.

String	Training Sample	Manual Classification	Lee-Fu's result	Abe's result
FBBBBEBBBBHBBEBHBBEBBBB	x	A	A	A
FBBBABBBGBABBGBABGBBBABBB	x	S	S	S
FBBBBEBBHBBBBEBBGBBEBBBBB	x	A	A	A
FBBAHBABBGBABKBBABBHBBBBABB		M	X*	M
FBBABKABGBABBGBABBGBABBBB		M	X*	X*
FBABBBBHBABBHBBBBABB		A	A	S*
FBBBABBBGBABGBABGBBBABBBB		S	S	S
FBBBABBBGBBAGABBBBBBGBAB		M	S*	X*
FBBBBBBABBGBBABBHBABBGBBBABBBB		S	S	S
FBBABBBGBBABBGBBABBGBBBABBB	x	M	M	M
FBBBBBBBBEBBBBHBBBEKBBEBHBBBBEBBBBBBB		A	A	A
FBBBBBBEKEBBBBGBBBEGEBB	x	S	S	S
FBBBBBBABBBBGBBBABGBBBBABBB		S	S	S
FBBBBBABBBHBBBAHBABHBBBBABBB		A	S*	M*
FBABBBGBBBBABBBBBBGBBAB	x	S	S	S
FBBBABBGBBBABGBABGBBBABB	x	S	S	S
FBBBABBBGBAHBABGBBBABBB		S	S	S
FBBBBABBBBBBGBABGBBBBBBBBABBBB	x	S	S	S
FBBBABBBBGBABHABGBBBBBABBB	x	S	S	S
FBBABBBBGBABGBABGBBBBABB		S	S	S

Table 1 (continuted)

String	Training Sample	Manual classification	Lee-Fu's result	Abe's result
FBBBBABBBGBBABGBBBABBB		S	S	S
FBBBBBABBHBBBBBBABBBGBBABBBBB		A	A	S*
FBBBBABBGBBAGABGBBBBABBBB		S	S	S
FBAKBABHBBBBBBBBBABBB		A	S*	S*
FBBBABBHBBBBABHBBBBABB	x	A	A	A
FBBBBABBBBGBABHBABGBBBBABBBB	x	S	S	S
FBBABBBGBABGBABBHBBBBABB		S	S	S
FBBBBABBGBABBBGBBABGBBAB	x	M	M	M
FBBBABBBBHAGABGBBBBABBBB		A	S*	S*

Key

S: submedian

M: median

A: acrocentric

X: rejection

*: recognition error

Table 2. Summary of the Methods and Results

Method	Automata (Grammars)	Metric	Additional information	No. of errors	No. of rejections	Recognition rate
Lee and Fu	Push-down (Context-free)	Probability	Structural information Weighting	4	2	64%
Abe	Finite-state	Penalty	None	9	0	47%
			Weighting	5	2	58%

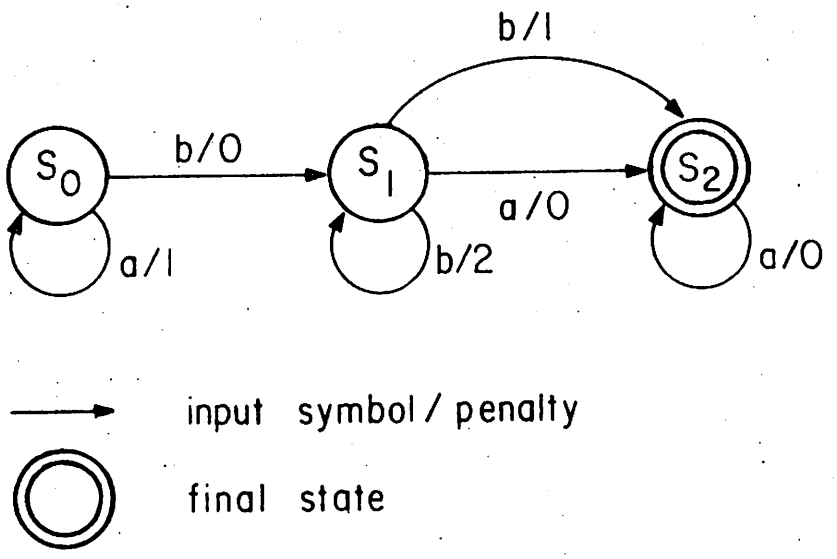


Figure 1. An example of PFST

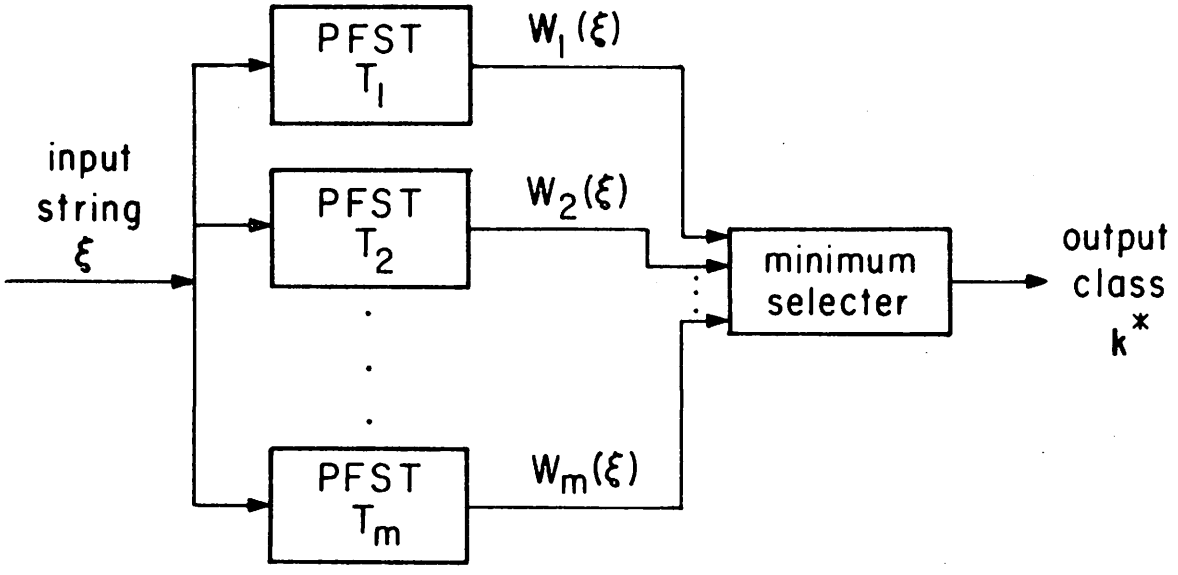
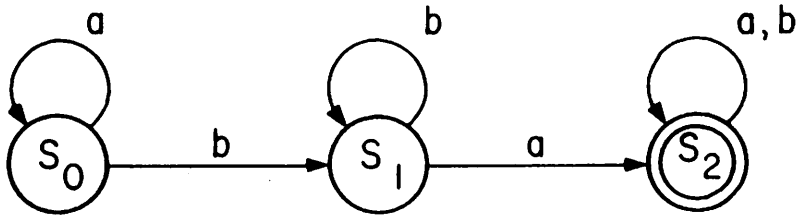
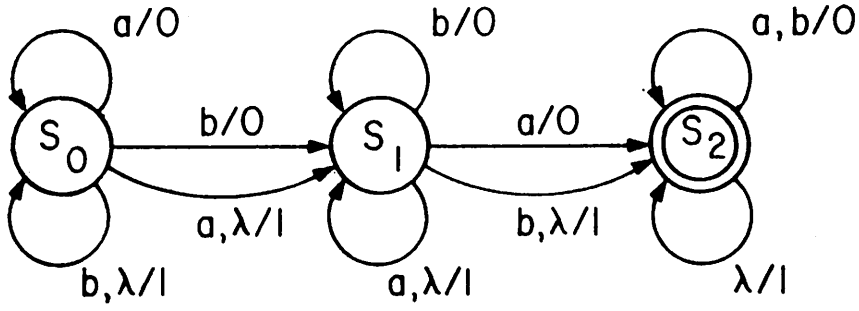


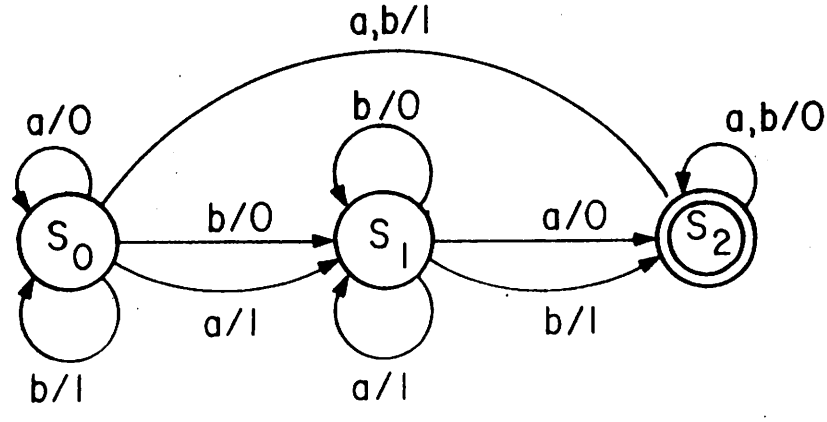
Figure 2. Multi-class recognizer using PFST's.



(a) A FSA.



(b) An equivalent PFST with λ-transitions.



(c) An equivalent PFST.

Figure 3. An example of a PFST which is equivalent to an ECFSA.

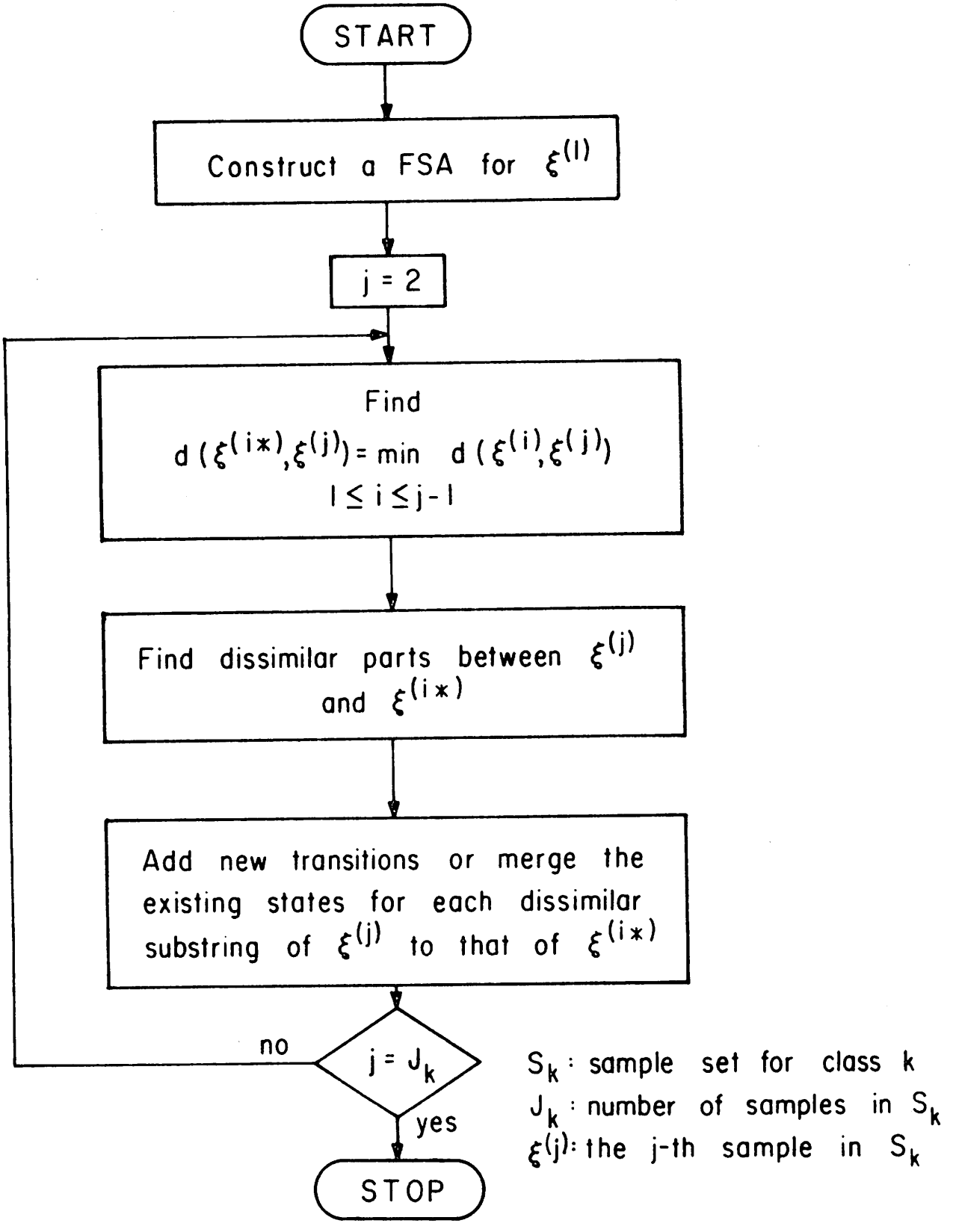


Figure 4. A method for grammatical inference of FSA.