To the Graduate Council:

I am submitting herewith a thesis written by David Jerome Braunegg entitled "An Approach to Industrial Computer Vision Using Syntactic/Semantic Learning Techniques." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Engineering, with a major in Electrical Engineering.

R. C. Gonzalez

R. C. Gonzalez, Major Professor

We have read this thesis
and recommend its acceptance:

Michael G. Thomason

Ernest L. Hall

Accepted for the Council:

Carolyn Winkel

The Graduate School

AN APPROACH TO INDUSTRIAL COMPUTER VISION USING SYNTACTIC/SEMANTIC LEARNING TECHNIQUES

A Thesis

Presented for the

Master of Engineering

Degree

The University of Tennessee, Knoxville

David Jerome Braunegg

August 1983

DEDICATION

This thesis is dedicated to my parents, who encouraged me through my school years and took pride in me as only parents can.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. R. C. Gonzalez for his guidance in research and assistance in writing this thesis. Appreciation is also extended to Dr. E. L. Hall and Dr. M. G. Thomason for serving on the thesis committee. Gratitude is extended to Mrs. Georgia C. Bunn, whose assistance in typing this manuscript has been invaluable.

ABSTRACT

Syntactic and semantic approaches to object pattern recognition are discussed. A well-known grammatical inference technique for syntactic pattern recognition is augmented with semantic considerations. Experimental results which demonstrate the power of the combined approach to recognition are presented. Limitations encountered with this approach along with suggestions for further research are discussed.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION.	1
II.	ELEMENTS OF SYNTACTIC PATTERN RECOGNITION	6
	Definitions	6
	Pattern Representation.	8
	Automata as Recognizers	9
III.	GRAMMATICAL INFERENCE VIA SYNTHESIS OF FINITE-STATE AUTOMATA.	13
	Concept of Grammatic Inference.	13
	K-Tail Algorithm for Grammatical Inference.	15
	Deriving Deterministic Automata from Non-deterministic Automata.	21
	Minimizing the Deterministic Automaton.	24
	The Class Overlap Problem	29
IV.	SEMANTIC CONSIDERATIONS IN PATTERN RECOGNITION.	36
	Why Semantics?	36
	Previous Research in Semantics.	38
	Selected Approach to Semantics.	44
V.	IMPLEMENTATION OF THE K-TAIL ALGORITHM WITH SEMANTIC CONSIDERATIONS.	48
	Overview.	48
	STRING.	48
	KTAIL	52
	SEPAR	53
	RECOG	54
	SEMAN	55
VI.	EXPERIMENTAL RESULTS.	57
	Size Generalization	57
	Length Generalization	59
	Sensitivity to Rotation	62
	Separation of Classes Via Semantics	63
	Limitations	70
VII.	CONCLUSIONS AND RECOMMENDATIONS	73

CHAPTER PAGE

LIST OF REFERENCES. 77

APPENDICES. 80

A. FORTRAN PROGRAM FLOWCHARTS. 81
B. FORTRAN PROGRAMS. 108
STRING. 108
KTAIL 126
SEPAR 157
RECOG. 160
SEMAN 162

VITA. 166

LIST OF FIGURES

FIGURE		PAGE
1.	State Diagram for the Regular Polygon Automaton.	11
2.	State Diagram for the R^+= {a, aa, baa} Automaton	19
3.	State Diagram for the k = 0 Automaton.	22
4.	State Diagram for the Deterministic Automaton.	25
5.	Merger Table for the Deterministic Automaton	27
6.	State Diagram for the Minimized Automaton.	28
7.	State Diagrams for Automata A' and A".	32
8.	State Diagram for Overlap Automaton A.	34
9.	Freeman Chain Code Vectors and Angles.	50
10.	Squares Automata	58
11.	Rectangles Automata.	61
12.	Bolt Samples and Edge Segmentations.	64
13.	Screw Samples and Edge Segmentations	65
14.	Wirenut Samples and Edge Segmentations	66
15.	Splice Samples and Edge Segmentations.	67
16.	Simplified Flowchart of Program STRING	82
17.	Simplified Flowchart of Program KTAIL.	88
18.	Simplified Flowchart of Program SEPAR.	101
19.	Simplified Flowchart of Program RECOG.	103
20.	Simplified Flowchart of Program SEMAN.	105

CHAPTER I

INTRODUCTION

Object pattern recognition can be divided into two broad methodologies: decision theoretic and structural [Tou and Gonzalez, Gonzales and Thomason (1978), Hall]. Decision theoretic methods are based on measurements of properties of the objects to be recognized, such as orientation angle, color, length, width, perimeter, and area. Structural methods deal with the structure of objects, such as shape and boundary features, internal features, positional relationships between object features, and the validity of the context in which the features exist in the objects. Structural recognition can be subdivided into two categories, syntactic and semantic. Syntactic recognition methods use the existence of features in objects and their positional relationships within the objects to differentiate between classes of objects. Semantic methods deal with the context of objects or object features and the relationships between them as a measure of the well-formedness of the objects. This thesis is concerned with syntactic pattern recognition augmented by semantic techniques.

One common approach to syntactic pattern recognition is to first develop representations of the outlines of the objects to be recognized. At the most basic level, the outlines are segmented and each segmentation vector is taken to be an element in a string which represents the object. More sophisticated methods may divide

the object outline into larger features such as lines, curves, and corners. Other object preprocessing is also possible, but the aim is always the same: extract object features which can be represented in some meaningful way (such as in strings, trees, webs, etc.).

Once a representation for the object features has been selected, a method must be devised to compare the features derived from an object to the features belonging to classes of objects. Many approaches to feature comparison have been developed. One approach is to simply compare listings of object features. A refinement of this approach also considers the order in which the features occur in the object. Another common approach to recognition is parsing, in which an attempt is made to find the sequence of productions from a grammar G which can be used to derive the given pattern representation. If such a sequence of productions can be found, the pattern representation belongs to the language L(G) and the pattern is a member of the pattern class described by L(G). In any case, recognition is achieved when it is determined that the object features belong to a unique pattern class.

After a specific approach to syntactic recognition has been chosen, the properties (features, production rules, etc.) of the pattern classes to be recognized must be determined. An obvious solution to this problem is to let a human operator describe these properties based on his personal knowledge of the classes involved. A different, but very useful, approach is to present the object

recognizer with example objects from the different classes and have it learn the distinguishing properties without human intervention. This learning process is known as grammatical inference.

During the learning process, it may not be practical (or even possible) to present the recognizer with a sample of every object that it will be required to recognize. In this case, it is desirable to have the recognition machine recognize all of the sample objects and other "similar" objects. This same generalization approach to recognition is used by humans who, though they may never have seen a specific dog before, can still recognize the animal as being a dog and not a cat based on their past experiences with dogs.

The power of a recognizer that can generalize from a limited sample set is balanced by a drawback due to the generalization process. When generalizing from training sets of different pattern classes, the recognizer may over-generalize. That is, it may generalize to the point where an input object is recognized as belonging to more than one class, when in reality it belongs to only one class. This presents a problem: if the generalizing capability is too limited in order to eliminate ambiguity, the number of objects that can be recognized which truly belong to a class may also be limited.

Although the area of grammatical inference is still in its infancy, several inference methods have been developed which yield acceptable results. One of these, which has been known for over a decade, deals with inference based on string grammars. This

algorithm is particularly useful for dealing with object boundary structures, which can be represented by strings of boundary features. An attractive feature of this method is the fact that the generalization property is controlled by a single parameter whose properties are well-defined. This grammatical inference algorithm is used as the core of the syntactic recognition portion of the thesis presented here.

Semantic techniques can be used to resolve the ambiguities which result from class generalization. First, the recognition classes are checked for overlap. Second, semantic attributes are determined which, although not capable of distinguishing between classes when used alone, are capable of resolving the class overlap ambiguity. Finally, these semantic attribute recognition techniques are brought into play whenever the recognizer is operating in an overlap area.

Semantic attributes can be associated with individual features, groups of features, or whole objects. One approach is to associate a feature vector with each object feature. Then, using a set of feature transfer functions which are associated with the syntactic production rules, a final feature vector sentence is derived for the object. A discriminant function decides the well-formedness of the final sentence and thus provides the final semantic decision on the syntactic recognition [Tang and Huang]. A simpler approach which is sometimes sufficient to assign global semantic attributes to an object, often through use of statistical recognition-type

measurements. These global attributes are then used to settle any syntactic recognition ambiguities.

A few semantic techniques are presented in this thesis for resolving recognition ambiguities. However, the semantic recognition area is relatively new and many ideas are currently being developed and tested. This thesis presents a framework which will allow the incorporation of many different semantic techniques into the recognition process. Thus, the work presented here provides a base for further research in the field of semantic recognition.

The following chapters will present the implementation of a well-known grammatical inference algorithm. The algorithm is enhanced through the use of global semantic attributes for objects to be recognized. It will be shown that these semantic attributes can indeed resolve ambiguities which result from the generalization properties of the syntactic learning algorithm.

CHAPTER II

ELEMENTS OF SYNTACTIC PATTERN RECOGNITION

Definitions

A few definitions [Gonzalez and Thomason (1978)] are in order before studying syntactic pattern recognition and will be presented here. These concepts originate in formal language theory and as such are similar to one's intuitive understanding of the concepts based on natural language. However, the definitions have strict usages and the comparison must not be taken for granted in all cases.

An "alphabet" V is any finite set of symbols.

A "string" x over an alphabet is any finite sequence of symbols from the alphabet. For example, the strings {0, 10, 010, 0000, . . .} are all valid strings for the alphabet {0, 1}. The terms "sentence" and "word" are also used in the literature as synonyms for string.

An "empty string" λ is a string which contains no symbols. An alphabet which contains the empty string is designated V* while an alphabet which excludes the empty string is designated V^+.

A "null string" ϕ is the condition of an absence of a string. Note that the empty string and the null string are not equivalent.

A "grammar" G is defined as a fourtuple:

$$G = (N, T, P, S)$$

where

N = a set of "nonterminals"

T = a set of "terminals"

P = a set of "productions"

S = the "start" or "root" symbol.

S must belong to the set N, N and T must be disjoint, and the union of N and T must form the alphabet V.

A "nonterminal" N is a variable which may be replaced by one or more terminals, nonterminals, or combinations of terminals and nonterminals depending on the production rules of the grammar involved.

A "terminal" T is an element from the alphabet of the grammar which cannot be further expanded or reduced by means of any production rule.

By convention, nonterminals are designated by upper case letters (A, B, S, etc.), terminals by lower case letters near the beginning of the alphabet (a, b, c, etc.), strings of terminals by lower case letters near the end of the alphabet (w, x, y, etc.), and strings containing both terminals and nonterminals by lower case Greek letters (α, β, γ, etc.).

A "production rule" P is a rule for the grammar which details the replacement of a specific nonterminal by one or more terminals, nonterminals, or combinations thereof.

A "start" symbol S is a nonterminal from which all expansions via the production rules are begun.

A "language" L(G) is any set (not necessarily finite) of strings constructed over an alphabet.

The length of a terminal string $|x|$ is equal to the number of symbols in that string. It is understood that the lengths of the empty string λ and of the null string ϕ are zero.

Pattern Representation

Objects or patterns to be recognized are described by features which can be represented in many ways. In syntactic pattern recognition, the object features are represented by symbols, one distinct symbol for each different feature. The relationship between these features in the object is represented by the structure in which the symbols are contained. These structures can be strings, trees, webs, etc., depending on the structure of the object itself. Since the methods used in this thesis employ strings of symbols, strings will be considered in detail.

To demonstrate how strings can be used to represent objects, take as an example simple geometric figures such as triangles, squares, pentagons, etc. Two features can be used to describe these objects: a straight line l and a corner c. The grammar for this set of objects is the fourtuple

$$G = (N, T, P, S)$$

where

$N = \{S, A, B\}$

$T = \{l, c\}$

$P = S \rightarrow Aclcl$

$A \rightarrow Acl$

$A \rightarrow cl$

and the start symbol is S.

Some sample production sequences generated from this grammar are

$S \Rightarrow Aclcl \Rightarrow clclcl$ (triangle)

$S \Rightarrow Aclcl \Rightarrow Aclclcl \Rightarrow clclclcl$ (square)

and

$S \Rightarrow Aclcl \Rightarrow Aclclcl \Rightarrow Aclclclcl \Rightarrow clclclclcl$ (pentagon).

It can be seen that the language generated by this grammar contains the infinite set of strings of the form

$$(cl)^n clcl$$

which represent all polygons with three or more sides.

Automata as Recognizers

To consider finite automata as pattern recognizers, it is first necessary to define automata themselves. A finite automaton is defined as a five-tuple [Gonzalez and Thomason (1978)]

$$A = (Q, V, \delta, q_0, F)$$

where

Q = a finite, nonempty set of states

V = a finite input alphabet

δ = a mapping of $Q \times V$ into 2^Q, the collection of all subsets of Q

q_0 = the initial state (contained in Q)

F = a set of final states (contained in Q).

Note that if the automaton is deterministic, the δ mapping is from $Q \times V$ into Q only.

As an example to explain the definition of an automaton, consider the finite automaton

$$A = (Q, V, \delta, q_0, F)$$

where

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_\phi\}$$

$$V = \{1, c\}$$

$$F = \{q_6\}$$

and the mapping δ from Q X V into Q is

$$\delta(q_0,c) = q_1 \qquad \delta(q_0,1) = q_\phi$$
$$\delta(q_1,c) = q_\phi \qquad \delta(q_1,1) = q_2$$
$$\delta(q_2,c) = q_3 \qquad \delta(q_2,1) = q_\phi$$
$$\delta(q_3,c) = q_\phi \qquad \delta(q_3,1) = q_4$$
$$\delta(q_4,c) = q_5 \qquad \delta(q_4,1) = q_\phi$$
$$\delta(q_5,c) = q_\phi \qquad \delta(q_5,1) = q_6$$
$$\delta(q_6,c) = q_5 \qquad \delta(q_6,1) = q_\phi$$
$$\delta(q_\phi,c) = q_\phi \qquad \delta(q_\phi,1) = q_\phi \quad .$$

The state diagram for this automaton is shown in Figure 1.

Suppose that the input string clclcl is to be input to the automaton for testing. Starting in q_0 the first input symbol c takes the automaton to state q_1. The next input 1 takes the automaton to state q_2. The same process is followed until the last input symbol takes the automaton to state q_6. Since q_6 is a final state (as denoted in the state diagram by a double circle), this string is accepted by the automaton. If the automaton had ended in

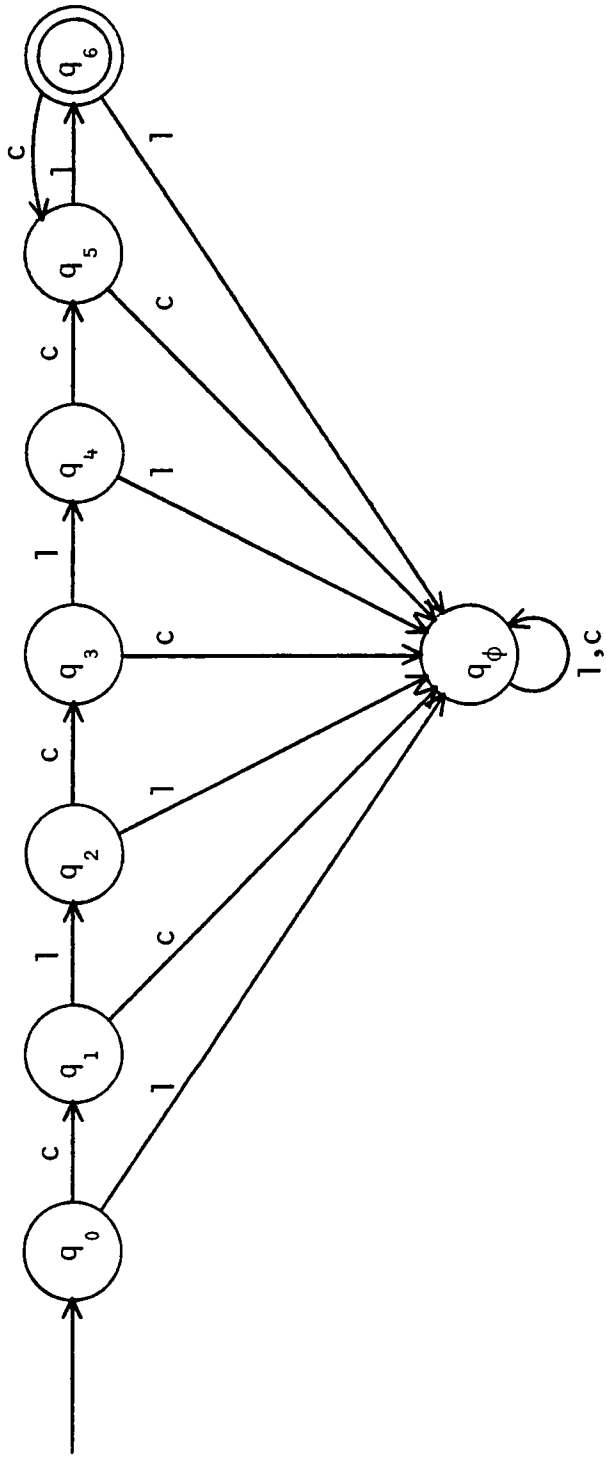


Figure 1. State Diagram for the Regular Polygon Automaton.

any state other than the final state after processing the complete input string, the string would have been rejected. It can be seen that this automaton accepts strings of the form

$$(c1)^n c1c1$$

which are the strings formed by the grammar for regular polygons which was defined earlier.

CHAPTER III

GRAMMATICAL INFERENCE VIA SYNTHESIS OF FINITE-STATE AUTOMATA

Concept of Grammatic Inference

For any pattern recognition system to operate, it must have information concerning the representative features of the pattern class that it is trying to recognize. The recognition system can then compare the features of the pattern under consideration to the class features to determine class membership. A major problem with this process is the determination of which features belong to a given class. One method of determining class features is to have a human operator independently decide which features belong to the class and then give this information to the recognizer. However, this method has several drawbacks. The human operator may not have intimate enough knowledge of the problem at hand to be able to select the appropriate class features. Also, the human operator is prone to errors such as selecting features which are not truly representative of the class under consideration, selecting only a limited subset of the features required for recognition, or selecting features based on his preconceived ideas about the class being studied (which may be erroneous). The most important objection to the involvement of an operator is the obvious necessity of human intervention for each new class that is considered.

The solution to the problems associated with a human operator's involvement in the process of selecting representative class features to be used for recognition is to create an automated method for the determination of these features. The desire here is to create an algorithm that can determine class features from a set of patterns that are representative of the pattern class under consideration [Gonzalez and Thomason (1978)]. This algorithm should be able to determine which features are common to the samples of patterns from the class and thus the features which any other pattern must possess to be considered a member of that class.

In order to reduce the requirement for a truly comprehensive set of representative samples of the pattern class (which may be impossible or impractical to obtain in any case), the concept of generalization can be used. The idea here is to have the recognition machine be able to recognize patterns represented by the sample set along with patterns of a "similar" nature. The notion of "similar" is defined by the generalization process used and the requirements of the application of the recognition system. Examples of "similarity" include (but are not limited to): same general pattern structure with variations in size, repeated occurrences of a feature within a pattern, and the insertion of new features between features present in the sample set of patterns. Although a great deal of work remains in the area of grammatical inference, at least one proven algorithm exists. This algorithm, presented by Biermann and Feldman [1970, 1972], will be referred to in this thesis as the "K-Tail Algorithm."

K-Tail Algorithm for Grammatical Inference

The K-Tail Algorithm deals with patterns which are represented by strings of symbols. The allowable symbols in any string are contained in the alphabet V of symbols. This alphabet is defined for the problem under consideration where each symbol in the alphabet represents one distinct feature which may be encountered. Two special symbols are also added to any alphabet used: the empty string λ and the null string ϕ. The empty string is defined to be a string containing no symbols while the null string denotes the absence of a string altogether.

Key to the development of this grammatical inference procedure is the concept of a k-tail. Let $R^+ \subseteq V^*$ be a non-empty sample set of strings. Also, let z be a string in V^* such that zw is in the sample set R^+. Then, given a positive integer k, the k-tail of z with respect to R^+ is defined as the set

$$h(z, R^+, k)$$

where

$$h(z, R^+, k) = \{w \mid zw \text{ is in } R^+, |w| \leq k\} .$$

That is to say, the k-tail of z with respect to R^+ is the set of strings w with the properties (1) zw is in the sample set R^+, and (2) the length of the k-tail w is less than or equal to k.

The procedure given by Biermann and Feldman [1970] for inferring an automaton from a given sample set R^+ and value of k can now be outlined. Letting

$$A(R^+, k) = (Q, V, \delta, q_0, F)$$

represent the inferred automaton, determine the states of the automaton by letting

$$Q = \{q \mid q = h(z, R^+, k) \text{ for } z \text{ in } V^*\}$$

where V^* is a finite alphabet of input symbols (including the empty string λ). The transition functions are then determined for each input symbol a in V^* via

$$\delta(q, a) = \{q' \text{ in } Q \mid a' = h(za, R^+, k)\}$$

where

$$q = h(z, R^+, k) ,$$

$$q_0 = h(\lambda, R^+, k) ,$$

and

$$F = \{q \mid q \text{ is in } Q, \lambda \text{ is in } q\} .$$

It follows that the states of $A(R^+, k)$ are subsets of all the k-tails that can be formed from the sample set R^+.

As an example to clarify the notation, consider the sample set $R^+ = \{a, aa, baa\}$ with $k = 1$. By the preceding definitions,

$z = \lambda$ $\quad h(z, R^+, k) = \{w \mid zw \text{ is in } R^+, |w| \leq k\}$

$$h(\lambda, R^+, 1) = \{w \mid \lambda w \text{ is in } R^+, |w| \leq 1\}$$

$$= \{a\}$$

$$= q_0$$

$z = a$ $\quad h(a, R^+, 1) = \{w \mid aw \text{ is in } R^+, |w| \leq 1\}$

$$= \{\lambda, a\}$$

$$= q_1$$

$z = aa$ $\quad h(aa, R^+, 1) = \{w \mid aaw \text{ is in } R^+, |w| \leq 1\}$

$$= \{\lambda\}$$

$$= q_2$$

$$z = ba \quad h(ba, R^+, 1) = \{a\} = q_0$$

$$z = baa \quad h(baa, R^+, 1) = \{\lambda\} = q_2$$

$$z = else \quad h(else, R^+, 1) = \phi = q_\phi$$

where "else" represents all other strings not explicitly mentioned. The states are $q_0 = \{a\}$, $q_1 = \{\lambda, a\}$, $q_2 = \{\lambda\}$, and $q_0 = \phi$. The state set and final state set are $Q = \{q_0, q_1, q_2, q_\phi\}$ and $F = \{q_1, q_2\}$, respectively. Note that although the k-tail strings were used to determine the states, only the state labels are entered in the state set Q. The final state set contains those states which contain the empty string λ in their k-tail representations.

Next, following the outlined procedure, the transition functions are determined. Recalling that $q_0 = \{a\} = h(\lambda, R^+, 1)$ or $h(ba, R^+, 1)$,

$$\delta(q_0, a) = h(\lambda a, R^+, 1) \text{ or } h(baa, R^+, 1)$$
$$= h(a, R^+, 1) \text{ or } q_2$$
$$= q_1 \text{ or } q_2$$

and

$$\delta(q_0, b) = h(\lambda b, R^+, 1) \text{ or } h(bab, R^+, 1)$$
$$= h(b, R^+, 1) \text{ or } q_\phi$$
$$= q_\phi \text{ or } q_\phi .$$

The other transition functions are developed in a similar manner to obtain the complete mapping:

$\delta(q_0, a) = q_1, q_2$ $\quad\quad$ $\delta(q_0, b) = q_\phi$

$\delta(q_1, a) = q_2$ $\quad\quad$ $\delta(q_1, b) = q_\phi$

$\delta(q_2, a) = q_\phi$ $\quad\quad$ $\delta(q_2, b) = q_\phi$

$\delta(q_\phi, a) = q_0, q_\phi$ $\quad\quad$ $\delta(q_\phi, b) = q_\phi$

Note that some of the transition functions are non-deterministic. The state diagram developed from these transition functions is given in Figure 2.

The automaton generated by the K-Tail Algorithm for the sample set $R^+ = \{a, aa, baa\}$ with $k = 1$ will obviously accept as inputs all of the strings in the sample set. Upon inspection of the automaton it can be seen that other strings will also be accepted due to the generalization aspect of the K-Tail Algorithm. Some examples of strings similar to those in the sample set that will be accepted by the automaton are

a^c $\quad\quad$ $c = 1, 2; c \geq 4$

$a^d b^e a^f$ $\quad\quad$ $d \geq 0, e \geq 1, f \geq 1$

and other combinations of a's and b's. Note that an input of an a is required to enter any final state in the automaton. Therefore, any string which is accepted by the automaton must end in an a.

The extent to which the automaton accepts strings similar to the sample set strings is controlled by the parameter k in the K-Tail Algorithm. The dependence of the automaton $A(R^+, k)$ on the parameter k is detailed in the following properties [Gonzalez and Thomason (1978)].

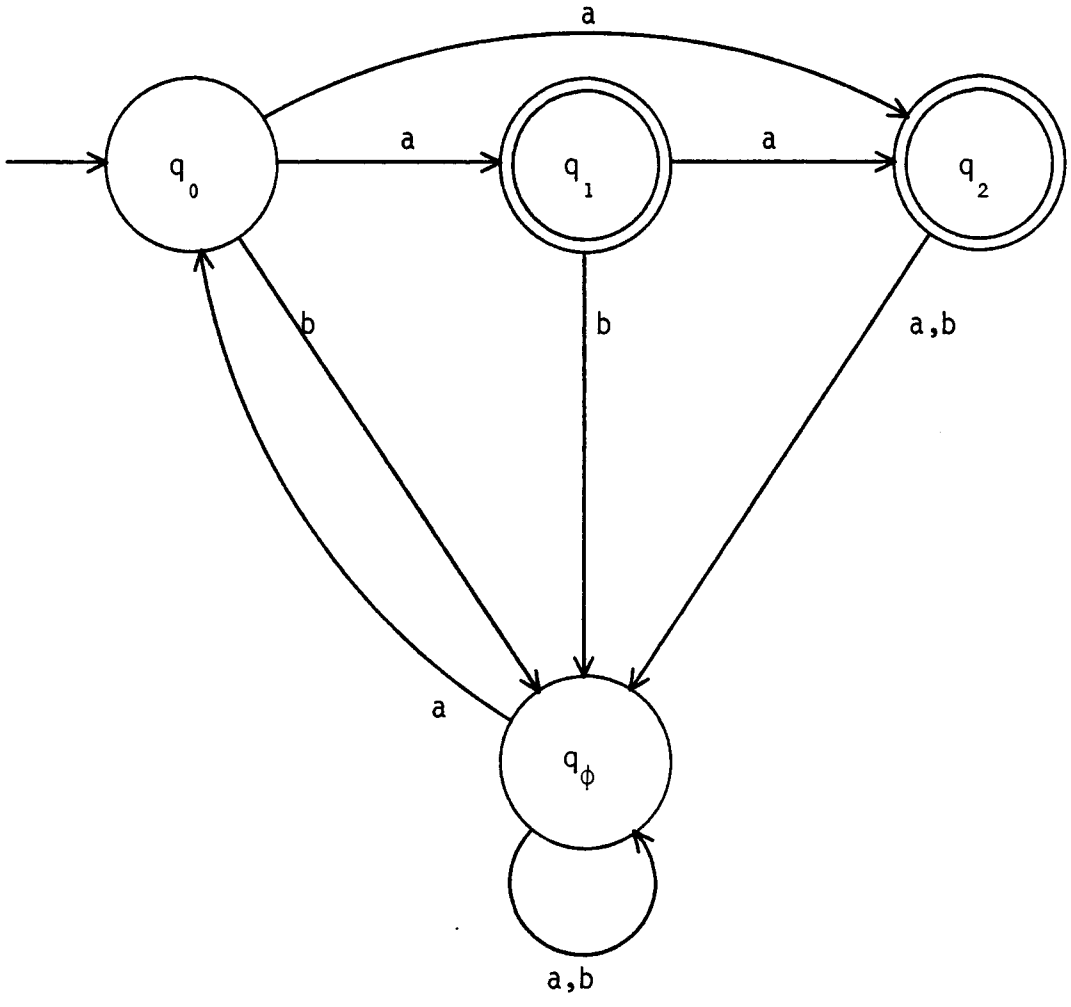


Figure 2. State Diagram for the $R^+ = \{a, aa, baa\}$ Automaton.

Property 1: $R^+ \subseteq L[A(R^+, k)]$ for all $k \geq 0$, where $L[A(R^+, k)]$ is the language accepted by $A(R^+, k)$.

Property 2: $L[A(R^+, k)] = R^+$ if k is equal to or greater than the length of the longest string in R^+.

Property 3: $L[A(R^+, k+1)] \subseteq L[A(R^+, k)]$.

Property 1 states that $A(R^+, k)$ will always accept at least the strings in the sample set R^+. By Property 2, if k is greater than or equal to the length of the longest sample string, then $A(R^+, k)$ will accept only the strings of the sample set. Finally, Property 3 states that as k increases, the scope of the language accepted by the inferred automaton decreases.

Biermann and Feldman had postulated that if $k = 0$, the inferred automaton will accept all strings composed of symbols from the input alphabet V^*. However, a quick example will show that if $k = 0$, the inferred automaton will accept a broad range of input strings, but not all input strings created from the input alphabet.

Let $R^+ = \{a, ab\}$ and $k = 0$. Then the states are

$z = \lambda$ $\quad h(\lambda, R^+, 0) = \phi = q_0$,

$z = a$ $\quad h(a, R^+, 0) = \{\lambda\} = q_1$,

$z = ab$ $\quad h(ab, R^+, 0) = \{\lambda\} = q_1$,

and

$z = else$ $\quad h(else, R^+, 0) = \phi = q_0$.

The transition functions are

$\delta(q_0, a) = q_0, q_1$, $\qquad \delta(q_0, b) = q_0$,

$\delta(q_1, a) = q_0$, $\qquad \delta(q_1, b) = q_0, q_1$,

with $F = \{q_1\}$.

The state diagram for this automaton is shown in Figure 3. From the state diagram, it is obvious that not all strings composed of a's and/or b's will be accepted by this automaton. Strings not containing at least one a will be rejected.

The elegance of the K-Tail Algorithm lies in the fact that the degree of generalization from the sample set is based on a single parameter. This elegant simplicity is translated into practicality by the well-defined properties which detail the effect of this parameter on the generalization process. By understanding the effects that changes in the parameter k have on the resulting automaton, a satisfactory result can be obtained after a few iterations of the automaton generation process.

Deriving Deterministic Automata from Non-deterministic Automata

For automated processing of input strings through an automaton, the automaton must be deterministic. In other words, at each state in the automaton, there must be one and only one state transition for any allowable input symbol. The procedure for forming a non-deterministic automaton from a deterministic one involves creating new states for each transition to more than one state. Thus, for Q states in the original non-deterministic automaton, there can be a maximum of 2^Q states in the new deterministic automaton. The procedure for creating the deterministic automaton is best explained by means of an example.

Consider the non-deterministic automaton given below. The automaton is

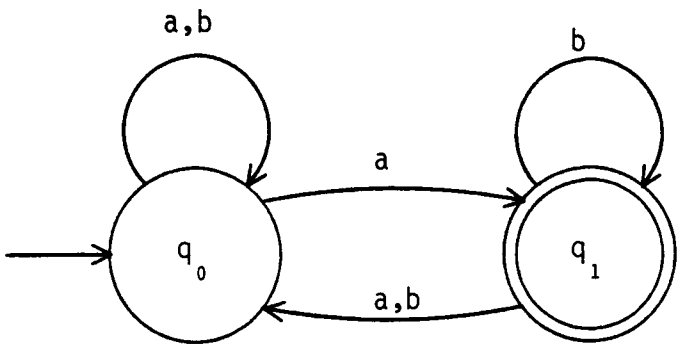


Figure 3. State Diagram for the k = 0 Automaton.

$$A(R^+, k) = (Q, V, \delta, q_0, F)$$

with

$$Q = \{q_0, q_1, q_2, q_\phi\} ,$$

$$V = \{a, b\} ,$$

$$F = \{q_1, q_2\} ,$$

and transition functions

$$\delta(q_0, a) = q_1, q_2 \qquad \delta(q_0, b) = q_\phi$$

$$\delta(q_1, a) = q_2 \qquad \delta(q_1, b) = q_\phi$$

$$\delta(q_2, a) = q_\phi \qquad \delta(q_2, b) = q_\phi$$

$$\delta(q_\phi, a) = q_1, q_\phi \qquad \delta(q_\phi, b) = q_\phi .$$

To form the deterministic automaton, the first step is to form the $2^4 = 16$ new states from the 4 original states. The new state set Q' will contain

$$q_A = \{q_0\} , \qquad q_B = \{q_1\} ,$$

$$q_C = \{q_2\} , \qquad q_D = \{q_\phi\} ,$$

$$q_E = \{q_0, q_1\} , \qquad q_F = \{q_0, q_2\} ,$$

$$q_G = \{q_0, q_\phi\} , \qquad q_H = \{q_1, q_2\} ,$$

$$q_I = \{q_1, q_\phi\} , \qquad q_J = \{q_2, q_\phi\} ,$$

$$q_K = \{q_0, q_1, q_2\} , \qquad q_L = \{q_0, q_1, q_\phi\} ,$$

$$q_M = \{q_0, q_2, q_\phi\} , \qquad q_N = \{q_1, q_2, q_\phi\} ,$$

$$q_P = \{q_0, q_1, q_2, q_\phi\} ,$$

and

$$q_R = \phi .$$

The new set of transition functions can now be determined from the original set,

$$\delta(q_A, a) = \delta(q_0, a) = q_1, q_2 = q_H,$$

$$\delta(q_A, b) = \delta(q_0, b) = q_\phi = q_D ,$$

and likewise

$$\delta(q_H, a) = q_J , \qquad \delta(q_H, b) = q_D ,$$

$$\delta(q_J, a) = q_I , \qquad \delta(q_J, b) = q_D ,$$

$$\delta(q_I, a) = q_N , \qquad \delta(q_I, b) = q_D ,$$

$$\delta(q_N, a) = q_N , \qquad \delta(q_N, b) = q_D ,$$

$$\delta(q_D, a) = q_I , \qquad \delta(q_D, b) = q_D .$$

The new final state matrix contains any state which was formed using at least one of the original final states q_1 or q_2.

$$F = \{q_B, q_C, q_E, q_F, q_H, q_I, q_J, q_K, q_L, q_M, q_N, q_P\}$$

Note that not all of the possible new states for the deterministic automaton are required. The state diagram for the deterministic automaton is given in Figure 4.

Minimizing the Deterministic Automaton

Two final steps can be taken to minimize the deterministic automaton. First, all inaccessible states are eliminated. These are states which are not initial states and cannot be reached from any other state. Second, equivalent states are combined.

Two states are said to be equivalent if and only if both states lead to the same state for each input symbol considered separately. An alternate way of stating this [Gonzalez and Thomason (1978)] is that the same language is generated whether one or the other states is designated as the starting state. Also, it should be obvious that three or more states are equivalent if they are all

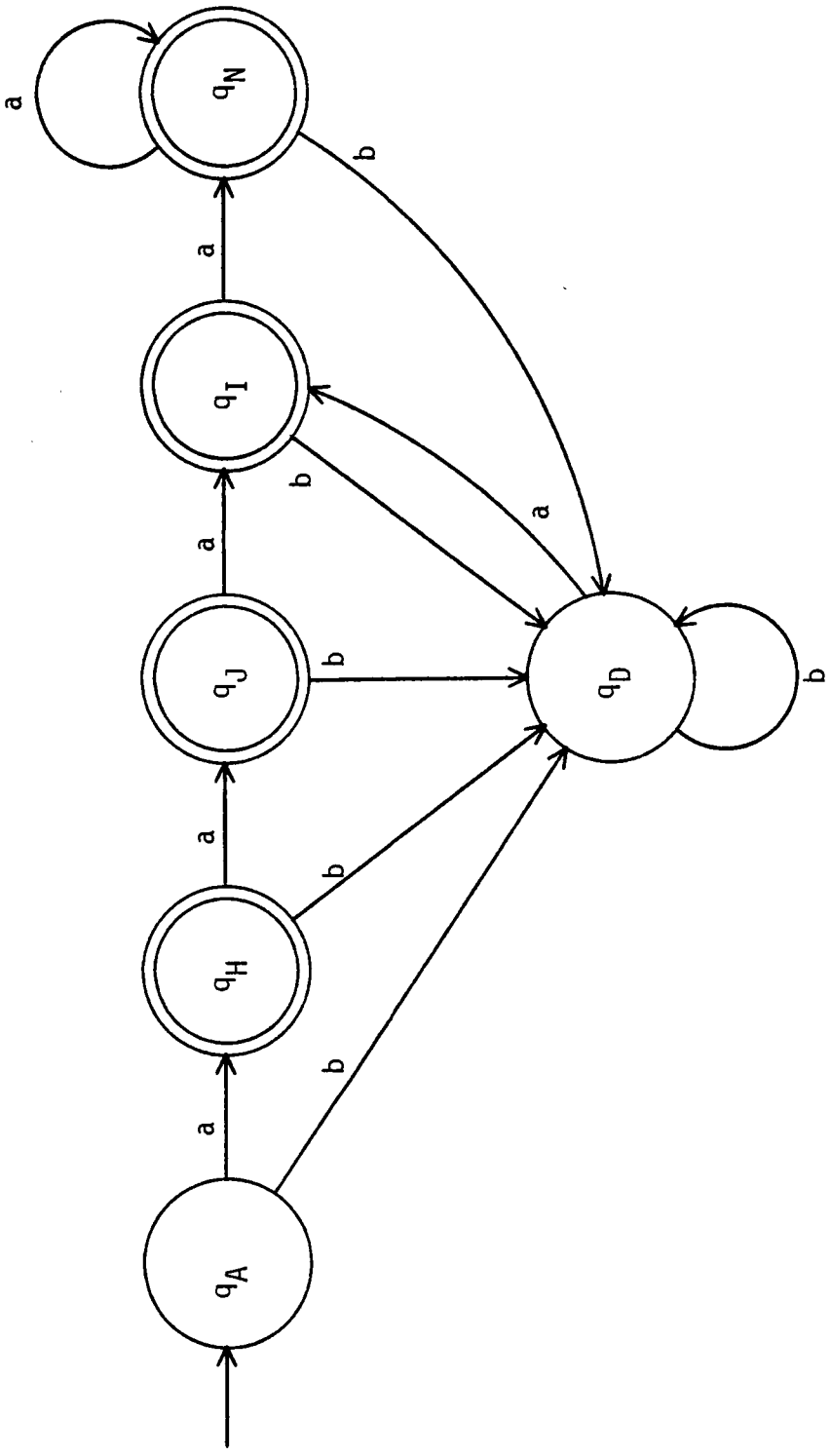


Figure 4. State Diagram for the Deterministic Automaton.

pairwise equivalent. The minimal automaton is obtained by locating and combining all equivalent states after the inaccessible states have been eliminated.

Two conditions are necessary and sufficient for two states q_i and q_j to be equivalent.

(1) q_i and q_j must both be in either F or $\overline{F}$.

(2) For each input symbol a in V, the transitions $\delta(q_i, a)$ and $\delta(q_j, a)$ must lead to the same state.

These two conditions can be checked by setting up a merger table which contains a block for each pair of states in Q. All state pairs which are not equivalent are then eliminated from the table, leaving only equivalent state pairs. Following from Condition (1), the first step is to cross out every block in the merger table for which the two corresponding states are not both in either F or $\overline{F}$. For the second step, the pairs of states which must be equivalent by Condition (2) are entered in each block. Finally, each block which contains a pair of states which has already been determined to be not equivalent is crossed out. This final step is repeated until no more blocks can be crossed out. Each remaining block indicates an equivalent state pair which is then merged into a single equivalent state. The merger table and resulting state diagram for the deterministic automaton of the previous example are given in Figures 5 and 6, respectively.

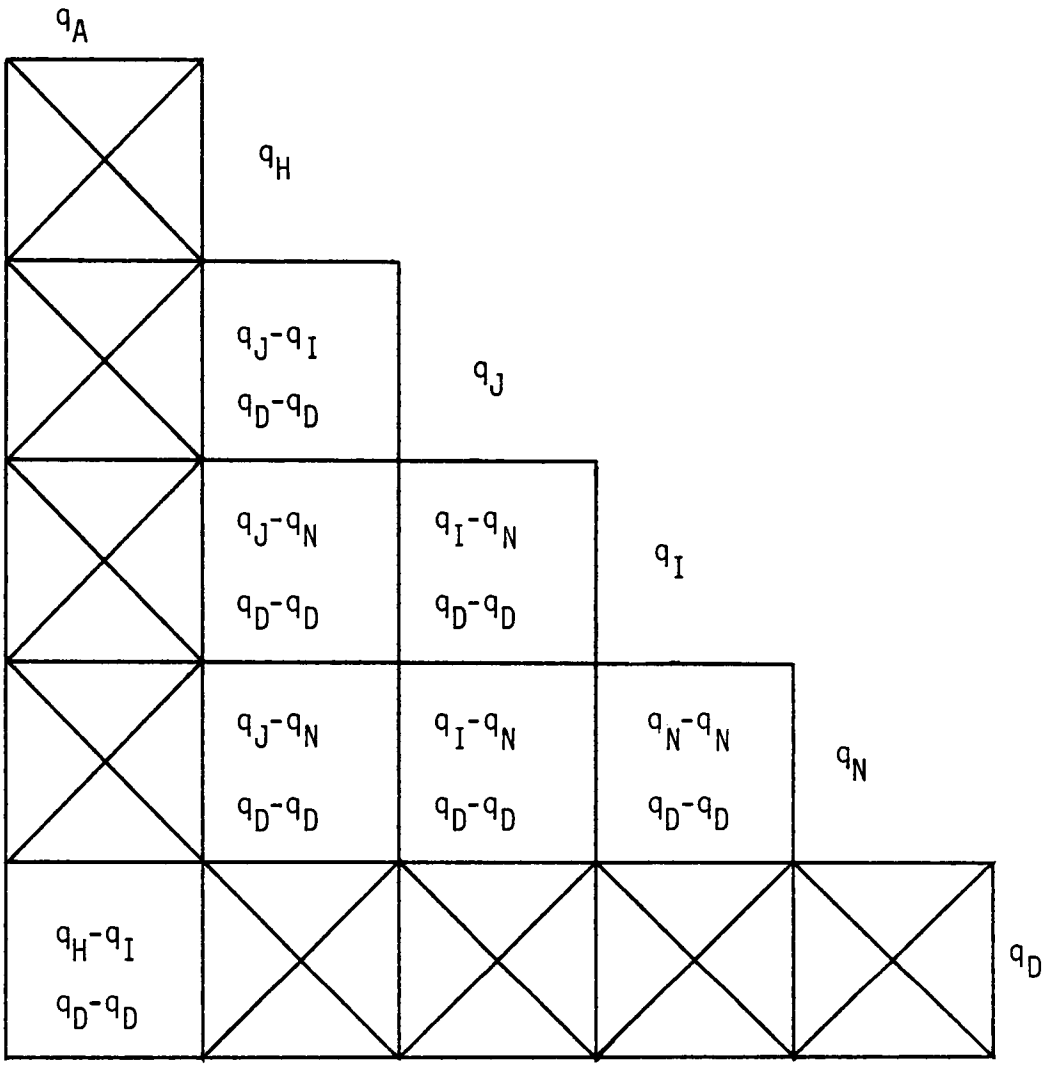


Figure 5. Merger Table for the Deterministic Automaton.

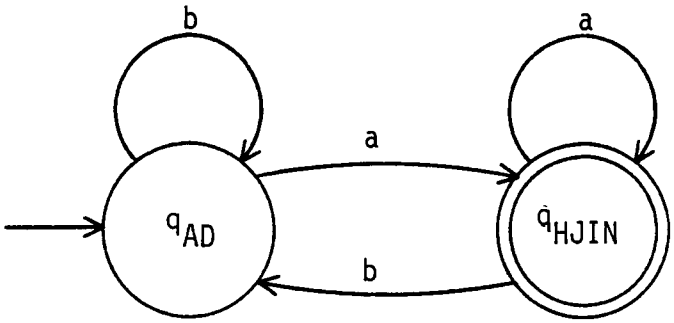


Figure 6. State Diagram for the Minimized Automaton.

The Class Overlap Problem

Due to the generalizing nature of the K-Tail Algorithm, it is possible that two or more automata can be synthesized from different sample sets and yet both accept an input string that only belongs to one automaton's recognition class. If this occurs, semantics can often be used to resolve the recognition ambiguity. It is important to know if a class overlap can occur before recognition is attempted to allow the inclusion of the semantic techniques in the recognition scheme. A method has been suggested by Gonzalez and Thomason [1982] [also Gonzalez] which can determine the overlap (or lack thereof) between two finite-state automata. This method will generate the overlap automaton which will indicate what types of input strings will not be "well behaved" in the sense of class separability in recognition. The method used for checking class separability follows.

Suppose that two deterministic finite automata $A' = (Q', V', \delta', q_0', F')$ and $A'' = (Q'', V'', \delta'', q_0'', F'')$, which have been derived from two distinct sample sets, recognize the two regular languages $L(A')$ and $L(A'')$, respectively. Then an automaton A which will recognize the language $L(A)$ such that $L(A) = L(A') \cap L(A'')$, the recognition class overlap, is given by

$$A = (Q, V, \delta, q_0, F)$$

where

(1) $Q = Q' \times Q'' = \{(q',q'') \mid q' \in Q', q'' \in Q''\}$

(2) $V = V' \cup V''$

(3) δ is a mapping from Q X V onto Q such that, for any symbol "a" from V

$\delta((q', q''), a) = (\delta'(q',a), \delta''(q'',a))$

(4) $q_0 = (q_0', q_0'')$ is the starting state

(5) $F = \{(q', q'') \mid q' \in F', q'' \in F''\}$.

This automaton A accepts an input string if and only if both automata A' and A" accept it. That is, A recognizes all strings in the intersection of L(A') and L(A") and only those strings. The following example will clarify the notation.

Consider the two automata

$$A' = (Q', V', \delta', q_0', F')$$

where

$Q' = \{q_0', q_1, q_2\}$

$V' = \{a, b\}$

$\delta'(q_0', a) = q_0'$, $\delta'(q_0', b) = q_1$

$\delta'(q_1, a) = q_2$, $\delta'(q_1, b) = q_1$

$\delta'(q_2, a) = q_2$, $\delta'(q_2, b) = q_2$

$F' = \{q_1\}$

and

$$A'' = (Q'', V'', \delta'', q'', F'')$$

where

$Q'' = \{q_0'', q_4, q_5\}$

$V'' = \{a, b\}$

$\delta''(q_0'', a) = q_0''$, $\delta''(q_0'', b) = q_4$

$\delta''(q_4, a) = q_5$, $\delta''(q_4, b) = q_5$

$$\delta''(q_5, a) = q_5 \quad , \qquad \delta''(q_5, b) = q_5$$

$$F'' = \{q_5\} \quad .$$

The state diagrams for automata A' and A" are shown in Figures 7(a) and 7(b), respectively. The strings recognized by these languages are given by the languages

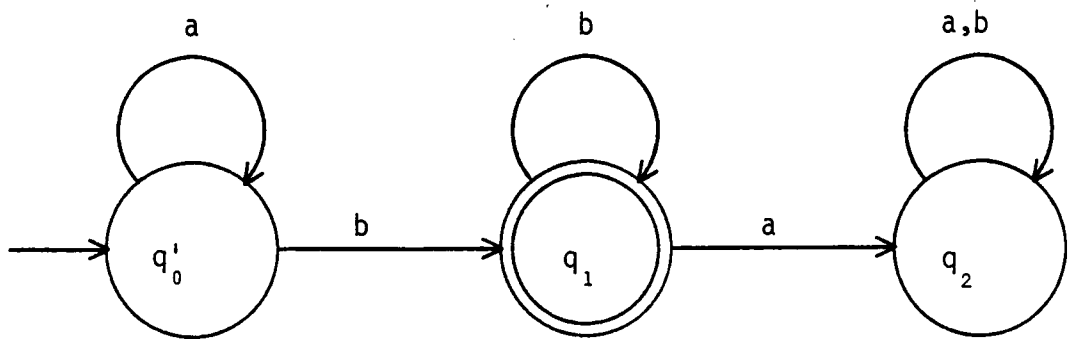
$$L(A') = a^c b^d \qquad c \geq 0,\ d \geq 1$$

and

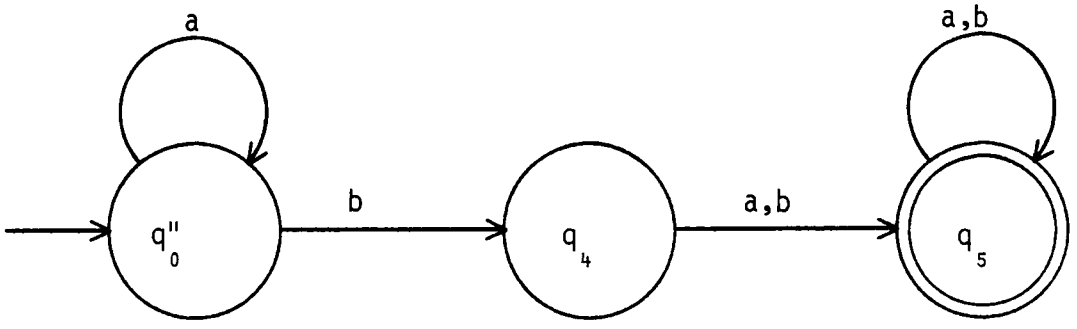
$$L(A'') = a^e b(a + b)^f \qquad e \geq 0,\ f \geq 1$$

where "+" denotes "or." Automaton A' therefore accepts inputs which consist of a (possibly empty) string of a's followed by a string of at least one b, while automaton A" accepts inputs which consist of a (possibly empty) string of a's followed by a single b followed by any (nonzero) number of a's and/or b's.

To obtain the automaton A which recognizes the overlap between A' and A", the procedure is to apply the first input symbol to both automata starting at q_0' and q_0'', respectively. A state pair which will be represented by a single state in the overlap automatom results from the transitions. These state pairs are formed from the transitions for each allowable input symbol. The process is repeated by applying all allowable input symbols to the resulting state pairs until all of the state-pair transitions have been found. Any state pair in which both component states are final states in their respective automata will be designated as a final state pair. Each state pair is then represented by a single state in the resulting overlap automaton A. The automaton A may be thought of as a parallel implementation of A' and A" simultaneously. A will accept an input



A. Automaton A'



B. Automaton A"

Figure 7. State Diagrams for Automata A' and A".

string if and only if both A' and A" will accept it. Continuing with the example, the overlap automata will be

$$A = (Q, V, \delta, q_0, F)$$

where

$$q_0 = (q_0', q_0'')$$

$$Q = Q' \times Q'' = \{(q_0',q_0''), (q_0',q_4), (q_0',q_5),$$
$$(q_1,q_0''), (q_1,q_4), (q_1,q_5),$$
$$(q_2,q_0''), (q_2,q_4), (q_2,q_5)\}$$

$$V = V'' \cup V'' = \{a, b\}$$

$$\delta((q_0',q_0''), a) = (q_0',q_0'')$$

$$\delta((q_0',q_0''), b) = (q_1,q_4)$$

$$\delta((q_1,q_4), a) = (q_2,q_5)$$

$$\delta((q_1,q_4), b) = (q_1,q_5)$$

$$\delta((q_2,q_5), a) = (q_2,q_5)$$

$$\delta((q_2,q_5), b) = (q_2,q_5)$$

$$\delta((q_1,q_5), a) = (q_2,q_5)$$

$$\delta((q_1,q_5), b) = (q_1,q_5)$$

and

$$F = \{(q_1, q_5)\} .$$

The state diagram for this overlap automaton is shown in Figure 8.

Note that no transition functions were specified for the state pairs (q_0', q_4), (q_1, q_0''), (q_0', q_5), (q_1, q_0''), and (q_2, q_4). This is simply due to the fact that the two states in any of these pairs cannot be reached simultaneously in A' and A" for any input string. These unreachable states can be eliminated from the state set Q with

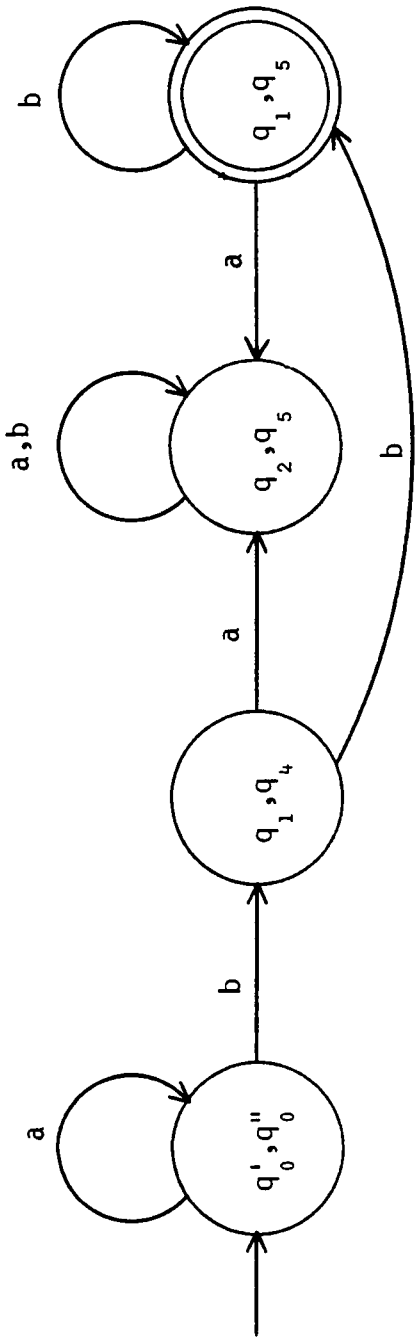


Figure 8. State Diagram for Overlap Automaton A.

no effect on the language L(A) that contains the strings accepted by A.

The form of the strings which will be accepted by the automaton A (and thus constitute the overlap set) is given by the language

$$L(A) = L(A') \cap L(A'') = a^g b^h$$

where

$g \geq 0$ and $h \geq 2$.

The method just described for determining overlap between the classes recognized by two automata has two important features. First, if there is no overlap between the classes, the language L(A) will be empty. Second, if there is a class overlap, the structure of the overlap can be determined by examining the automaton A which was generated. Knowledge of the form of the overlap may help determine which semantic techniques would be useful in resolving the overlap.

CHAPTER IV

SEMANTIC CONSIDERATIONS IN PATTERN RECOGNITION

Why Semantics?

It has been shown that the k-tail grammatic inference algorithm, although powerful in the sense that it can generalize to accept strings similar to those of a sample set, has a drawback in that it may overgeneralize, leading to a problem with class separability between two or more automata. It has also been indicated that semantics can be used to resolve this separability problem. Reasonable questions to ask might be, "Why use semantics at all? Why not simply limit the generalization of the K-Tail Algorithm or define 'better' primitives to describe the objects?" Indeed, in some cases these are reasonable solutions to the class overlap problem. However, limiting the k-tail generalizations may yield a recognizer which does not accept all of the strings which are in the class to be recognized. Also, the primitive alphabet cannot always be expanded due to computational limitations or due to having a previously encoded set of object representations which cannot be generated again using a new set of primitives because of time, object availability, or other constraints. Semantics can be added to a recognition scheme without changing the syntactic recognizer which may already be in place.

A major benefit of semantic considerations is the fact that semantics can add quantitative information to the recognition

process. For example, a semantic pattern description can include the direct measurement of properties such as area, perimeter, and color. The quantitative information contained in such a semantic pattern representation naturally complements the structural information conveyed by a syntactic pattern representation. A syntactic/semantic recognition system can thus consider both the structure and quantitative measurements of a given pattern to determine its class membership.

Another area in which semantic techniques are useful is in checking the context or appropriateness of data which has been determined to be structurally correct. Computer programmers supply this semantic consideration of context whenever they check computer programs. Although compilers provide syntactic recognition to determine if a statement is structurally correct (such as the FORTRAN statement C = A/B), the programmer must check the context of the elements in each statement to detect errors (such as the possibility of B being equal to zero).

Semantics can also be used to check the well-formedness of objects recognized by syntactic methods. For example, if a syntactic grammar were recognizing the outlines of building-block shapes by looking for sequences of straight lines and right-angle corners, semantics could be used to check that the sum of all the $\pm$ 90 degree corners equalled 360 degrees to insure that the figures are closed.

Another benefit of semantics is its ability to consider objects on a global level whereas some syntactic methods, such as

boundary following techniques, are necessarily local operations. This limitation can easily be seen by considering an hourglass-shaped object. Although the distance between the boundary points directly opposite each other in the pinch area is short in a global sense, these points would appear quite far apart in a string representation of the boundary. As another example, syntactic techniques can be used to detect interior features of an object while semantics can be used to describe the global relationships between these (local) features.

Finally, an important aspect of semantics is the ability to assign meaning to symbols recognized by syntactic techniques. Knuth [1968] gives an excellent example of a context-free grammar which will generate rational binary numbers. This grammar can be used to recognize binary numbers while rejecting numbers which are not structurally correct. He then derives the value of the binary numbers through the use of semantics.

Previous Research in Semantics

Although the area of semantics is not as well-developed as syntactic recognition, some research has been performed and the body of knowledge in this field is continually growing. Early work by Knuth [1968] dealt with the semantics of context-free languages. The binary number example given earlier was used by Knuth to demonstrate the ability of semantics to assign meaning to a string by defining attributes of the symbols in a derivation tree for that string. He defined the attributes by means of functions associated

with each production of the grammar. Thus, in the case of the binary numbers, a numerical value could be assigned to a string of one's and zero's by assigning a value (attribute) to each digit as the number was formed. This assigning of attributes was accomplished simultaneously with the application of each production rule. The concept of assigning attributes to individual symbols which are then modified by attribute production rules in parallel with the syntactic production rules is a principle employed by many researchers in the field.

Knuth also introduced the interesting concept of synthesized and inherited attributes. An inherited attribute is defined in terms of the attributes of the ancestors of the nonterminal symbol. Thus if the symbol 1 is assigned the numeric value (attribute) one and the symbol 0 is assigned the numeric value (attribute) zero, the string 10 in a binary number system will be assigned the numeric value (inherited attribute) two. The process of multiplying the value of the right-most symbol by 2^0 and the next symbol to the left by 2^1 and finally summing the results to obtain the value for the complete string is an example of a semantic production rule for inherited attributes.

In the case of a rational binary number such as 10.1, it is obvious that the previously stated rule for evaluating the attribute of the string would fail. It is necessary to decompose the string 10.1 into its constituent parts

10 · 1

in order to correctly evaluate its attribute. The correct value for the right-hand 1 is one-half. This value can be correctly determined after the syntactic production rule has been applied which combines the strings 10, ., and 1 into the final string 10.1. Attributes which are defined in terms of the descendants of the corresponding nonterminal symbol are defined to be synthesized attributes. Thus, the attribute "one-half" assigned to the righthand symbol 1 in the string 10.1 is a synthesized attribute since it is defined in terms of the descendant 10.1 of the symbol 1 along with the strings . and 10.

It is obvious that a problem of circularity could develop in which an inherited attribute depends (via an attribute production rule) on a synthesized atrribute which in turn depends on an inherited attribute, ad infinitum. Knuth has developed a test for circularity which can check for this undesirable condition. It is interesting to note that Knuth has shown that this problem can be avoided entirely by proving that either synthesized or inherited attributes alone are sufficient to define the meaning associated with any derivation tree. However, semantic rules which depend on only one type of attribute may be considerably more complicated than semantic rules which allow both kinds of attributes.

In their paper on semantic picture recognition, Baird and Kelly [1974] define some useful terms for work in semantics. For example, keeping in mind that an object can have several attributes assigned to it via semantic production rules, they define a quality

to be a set of mutually exclusive properties or attributes, such as {color, temperature, shape, length}. They also deal with the global nature of semantics by defining two-place predicates such as "_____ touches _____" and "_____ is above _____." In addition, Baird and Kelly define operations and rules of inference which are useful in scene description.

As justification for the semantic approach to picture recognition, Baird and Kelly explained the two philosophical approaches to pattern recognition: extensional and intensional. An extensional description of a class is simply a set of descriptions of each individual object in the class. Alternatively, an intensional class description is a set of essential properties which are peculiar to the objects which are denoted by the single class name. They note that although extensional descriptions can be satisfied through the use of syntactic techniques alone, the use of intensional class descriptions requires the introduction of semantic considerations.

Finally, Baird and Kelly show that the use of rules of inference (semantic production rules) allow properties to be assigned to objects which are not locally observable. These rules of inference allow contextual information to be exploited and object descriptions to be obtained such that recognition can occur based on intensional class descriptions. They demonstrate the usefulness of rules of inference for scene description by means of several examples in a limited balloon-and-ball world.

An interesting application of semantics to obtain syntactic primitives was developed by Pavlidis and Ali [1979]. They describe a general shape grammar in terms of four object boundary primitives. The primitives are assigned formal names for convenience:

(1) arcs which can be approximated by a quadratic curve are given the formal name QUAD;

(2) sharp protrusions or intrusions are given the name TRUS;

(3) long linear segments are assigned the name LINE;

(4) short segments having no regular shape are grouped into the final category and named BREAK.

As the object boundary is followed, semantic considerations are used to segment the boundary into the primitives defined above. For example, a LINE will be derived from a single vector or a set of mutually near-colinear vectors. The semantics define the degree of colinearity required for a sequence of vectors to form a line. A QUAD will be obtained when a subsequence of vectors match a polygonal approximation where both the length and the vertex angle are more or less uniform. Semantics determine the limits on "more or less." The other boundary primitives have similar semantic bases. After the semantic-based boundary primitives have been obtained, normal syntactic techniques can be used on the segmented object boundary.

A similar approach was taken by You and Fu [1979] for determining syntactic recognition primitives via semantic techniques. The two primitives they defined are an open curve segment and the connection between two consecutive curve segments. Four attributes

(direction vector, length of curve, angular change of curve, and a measure of symmetry) were used to describe each curve segment while a single attribute (angular change at concatenating point) was used to describe the connection. An interesting feature of the work by You and Fu was the definition and use of a transformation which rendered all of the attributes independent of translation, scaling, and rotation of the objects under study. During the recognition process these attributes were operated on by a set of attribute rules, each of which corresponded to a syntactic production rule.

In working with a string derived from an object boundary there is always the problem of locating the starting point of the string. You and Fu's solution to this problem was to define multiple starting points. Sharply convex points were located in the image boundary and each of these were tried as possible starting points during the recognition phase. By trying all of these points they were assured of finding the one which was used for training the recognizer.

In this work, You and Fu introduced some techniques to allow for image noise. In a later work [You and Fu (1980)], they expanded these concepts to achieve distorted shape recognition. An error-correcting parser was developed and demonstrated on images which either were truncated or had spurious objects concatenated onto them. The attributed grammars which allowed the use of semantics played an important part in these distorted shape recognition techniques.

Finally, an interesting application of semantics was investigated by Tang and Huang [1979]. A syntactic and semantic description

was developed of an object which was to be located in a scene. This description was then used to construct a creation machine which attempted to find the desired object amid the noisy picture primitives. The syntactic description was used to guide the search for pattern primitives while the semantic part of the description indicated which of the next possible syntactic categories were actually qualified to be the next pattern primitive. The efficacy of this approach was demonstrated with examples of finding airports in aerial photographs.

Selected Approach to Semantics

The previously cited research into syntactic/semantic pattern recognition gives many suggestions for the application of semantics to resolving the class overlap problem which may occur when using the K-Tail Algorithm. It has been seen that an important facet of semantics is its ability to consider global features and quantitative information in an image to complement the local, structural nature of the syntactic approach. This facet will be exploited to resolve the class overlap problem.

The global, quantitative semantic information for each image considered was derived while the string representation for the image was being formed. Information concerning the area, perimeter, and centroid of each image was gathered. Another global measure which was found to be useful for semantic recognition of an image was the number of local maxima of the function of distance to the object boundary from its centroid as the boundary is followed. This measure

yields important information concerning the number of corners on the object edges. Other semantic measures can also be derived if they are found useful for object recognition. The particular semantic information required is highly dependent on the nature of the recognition problem at hand.

It should be noted that the definition of attributes to be associated with each syntactic primitive was expressly avoided. If the semantic information can be limited to strictly global information while still achieving satisfactory recognition results, the semantic processing for recognition can be kept fairly simple. This can result in fast semantic recognition algorithms, whereas a detailed structure of semantic information could require time-consuming methods such as the construction and implementation of semantic automata. The simpler global approach to semantics suits the needs of the problem at hand--namely, the choosing of one recognition class over another after the syntactic processor has already narrowed the number of possible classes down to two or three. The class finally assigned to a given object is simply determined by pairwise semantic comparisons until only one acceptable class remains.

To express the selected approach to semantics more concisely, some explicit notation is useful. For each pattern considered, a set P consisting of the list of syntactically acceptable pattern classes and the derived semantic features for the pattern is formed. The set P is defined as

$$P = \{c_1, c_2, \ldots, c_m; s_{1,2}, s_{1,3}, \ldots, s_{n-1,n}\}$$

where n is the number of pattern classes considered by the recognition scheme, the c_i are the syntactically acceptable pattern classes, $1 \leq m \leq n$, and the $s_{i,j}$ are the semantic feature measurements which will be used to resolve class overlaps between classes i and j.

The trivial case of recognition occurs when m = 1. In this case, the pattern is simply assigned to class c_1. If m > 1, the semantics must be used to resolve the recognition class overlap. The semantic feature measurements allow pairwise decisions to be made concerning class membership according to the rule

$$s_{i,j} \begin{cases} > 0, \text{ keep class i} \\ < 0, \text{ keep class j.} \end{cases}$$

These pairwise semantic decisions are continued until only one acceptable class remains.

As an example, consider the pattern set

$$P = \{c_1, c_2, c_3; s_{1,2}, s_{1,3}, s_{1,4}, s_{2,3}, s_{2,4}, s_{3,4}\}$$
$$= \{1, 3, 4; -6, -18, -4, -12, 3, 8\}$$

where n = 4. Since m = 3 (prior syntactics have eliminated class 2), the semantic feature measurements must be considered to be able to assign a unique pattern class to the pattern which generated this set P. The first two allowable classes, $c_1 = 1$ and $c_2 = 3$, are compared

$$s_{1,3} = -18 < 0$$

to eliminate class 1 (c_1). The remaining two classes, $c_2 = 3$ and $c_3 = 4$, are then compared

$$s_{3,4} = 8 > 0$$

to eliminate class 4 (c_3). Thus the pattern class that is assigned to the pattern which generated the set P is class 3 (c_2).

From the above notation, it can be seen that to apply semantics to a recognition problem, the semantic measures $s_{i,j}$ must be derived for each pair of pattern classes. The derivation of these measures is achieved heuristically for each application of this recognition scheme. Keeping in mind the quantitative and global nature of semantics and using the class overlap information available through application of the method described in Chapter III, an operator should be able to derive semantic considerations which will allow the separation of the pattern classes involved in a specific recognition problem.

CHAPTER V

IMPLEMENTATION OF THE K-TAIL ALGORITHM WITH SEMANTIC CONSIDERATIONS

Overview

The approach to pattern recognition which has been described in this paper was implemented on a DEC PDP 11/40 mini-computer in five separate modules. The first module, STRING, is an image processing module which derives the boundary segmentation string. The second module, KTAIL, implements the K-Tail Algorithm to obtain the recognition automaton for an object class. The third module, SEPAR, checks the recognition automata for separability. The fourth module, RECOG, attempts the recognition of a given input string via the recognition automaton. The final module, SEMAN, uses semantic information from the object to be recognized to resolve class overlaps if they occur.

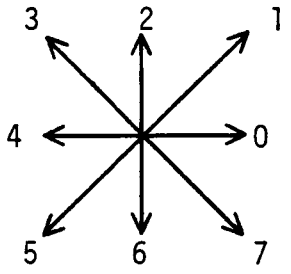
STRING

STRING accepts a 256 X 256 pixel digitized image of the object to be recognized. The image is expected to be a silhouette of the object against a white background. STRING thresholds the image to separate the object from the background and follows the object boundary to derive the edge string. The edge is segmented into vectors of a length input by the user. (This allows the user to obtain a "reasonable" segmentation of the input image. That is, the

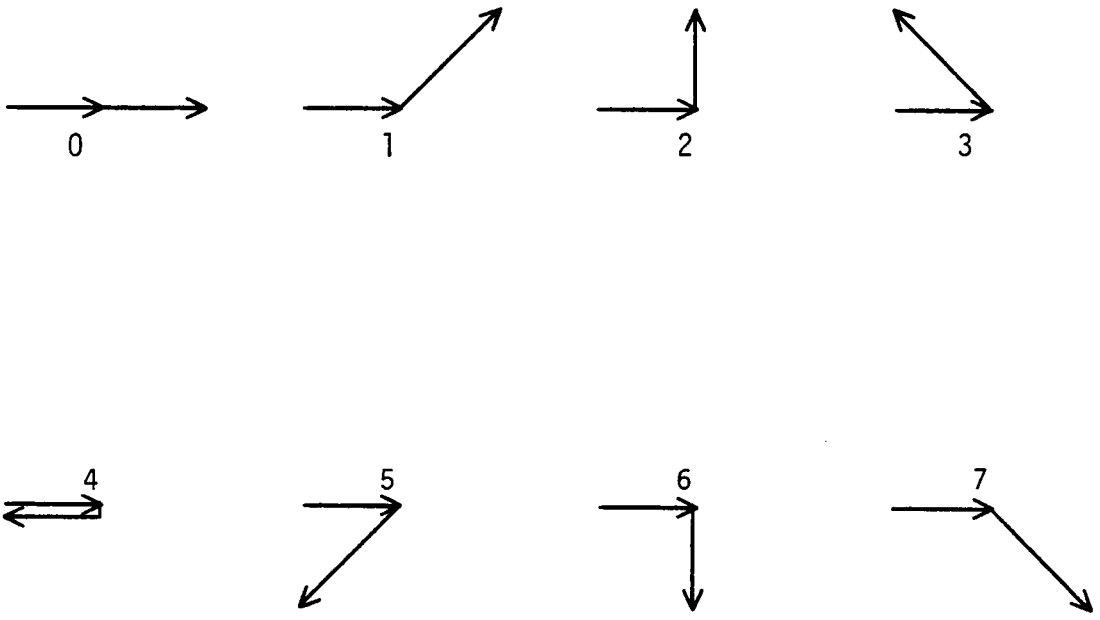
vector boundary approximation should give a representation of the true object outline with a minimum number of vectors required to obtain this representation.) These vectors are those of the familiar Freeman chain code and are shown in Figure 9. A string consisting of the angles between the edge vectors is then derived for use in the recognition phase. Angles between the edge vectors are used rather than the vectors themselves (Figure 9) to attempt to make the object string representation independent of rotation. Both the angle string and an image consisting of the edge segmentation vectors are stored to disk.

To derive the object edge segmentation, STRING chooses as its starting point the first object pixel it encounters in a raster scan of the image. Since it is desired to be able to recognize an object in any rotation orientation, the representative sample set of objects for any class (which is used in the learning phase) must contain objects in all of the rotation orientations which would yield different starting points.

Successive object edge pixels are found by performing a clockwise search of the eight nearest neighbors of the current pixel until the next edge pixel is encountered. This pixel then becomes the new current pixel. The segmentation vectors are also determined at this time. When a pixel is found which is one vector length away from the starting pixel in either a horizontal or vertical direction, a best-fit Freeman chain code vector is chosen to connect the starting pixel to a point as close to the current pixel as possible. The



A. Freeman Chain Code Vectors



B. Vector Angles

Figure 9. Freeman Chain Code Vectors and Angles.

process is then repeated each time an edge pixel is encountered which is one vector length away from the tip of the last vector. Since the segmentation vectors all have the same width and length (unless the width or length is zero), this vector-finding process is guaranteed to match the tip of the final vector with the tail of the initial vector.

STRING, since it operates on the image of the object of be recognized, also derives the semantic information which will be used to resolve class overlaps. For the semantic recognition example of Chapter VI, four global object attributes were derived for each object. These were the area, length of the perimeter, centroid, and number of boundary maxima for each object. The area and perimeter were used to calculate the PSA (Perimeter Squared divided by Area) which gives a measure of the compactness of the object. Long, thin objects have a large PSA while short, squat objects have a comparatively small PSA. This information was stored to disk in a semantic attribute file for each object. A simplified flowchart of STRING is shown in Figure 16, Appendix A. The complete FORTRAN program listing is given in Appendix B.

It should be noted that the image acquisition and processing by the STRING module are the most time-consuming operations in the recognition process. For a 256 X 256 pixel image the sampling typically takes ten seconds and the processing by STRING about one minute. However, image acquisition systems are available which can digitize an image of this size in 1/30th of a second. Also, hardware

realization of the image processing operations such as thresholding and edge detection offer significant improvements over the speed of the software algorithms which were implemented.

KTAIL

KTAIL operates on a set of angle strings derived by STRING to derive the recognition automaton. First the K-Tail Algorithm is implemented to create a nondeterministic, nonminimized automaton. This automaton is then transformed into a deterministic automaton and inaccessible states are eliminated. Finally, the automaton is minimized and the resulting deterministic, minimized automaton is stored to disk.

The implementation of the K-Tail Algorithm which was discussed in Chapter III is rather straightforward. By representing the eight possible vector angles by the numbers 0 through 7, each angle in an angle string can be stored in only three bits of computer memory. This economy of memory is important since for each angle string, all of the truncated angle strings and their associated k-tail strings are stored in order to form the recognition automaton. After the k-tails have been formed, states are assigned to distinct k-tails. These states, along with the empty state and the initial state (if not already formed), are then used to form a non-deterministic transition table.

Transitions from any state to more than one state for a single input are eliminated by forming additional states which take the place of the multiple states for these transitions. These new states, of

course, must also have their transitions recorded in the transition table and checked for transitions to multiple states. This process is repeated until a deterministic transition table (automaton) is obtained. The resulting transition table, though deterministic, is by no means minimal. Equivalent states are combined and inaccessible states are eliminated to form the final deterministic, minimized automaton. This final recognition automaton, including final state information, is then stored to disk in the form of a transition table for use by RECOG in testing input strings for class membership. A simplified flowchart of KTAIL is shown in Figure 17, Appendix A. The complete FORTRAN program listing is given in Appendix B.

Execution time for the KTAIL module depends on the number of strings input, the length of those strings, and the value of k selected. For a set of eight strings containing from twenty-eight to thirty-two elements each, execution of the KTAIL module took four seconds for k = 1 and thirteen seconds for k = 5. As with the STRING module, hardware implementation of the KTAIL module will greatly improve its speed.

SEPAR

SEPAR implements the previously described algorithm which checks for overlap between two automata. It accepts as input two automata as derived by KTAIL and returns the overlap automaton if an overlap exists.

The two input automata are in the transition table form supplied by KTAIL. New states are formed which encompass all

pairwise combinations of the states of these two automata. A transition table is built for these new states which becomes the overlap automaton for the state pairs associated with the input automata. This overlap automaton is also stored to disk. A simplified flowchart of SEPAR is shown in Figure 18, Appendix A. The complete FORTRAN program listing is given in Appendix B.

Typical execution times for the SEPAR module are under one second. The simple nature of the class separation algorithm yields fast results for a wide range of input automata sizes.

RECOG

RECOG accepts as inputs an automaton derived by KTAIL which will be used to test an input string for class membership and an object edge angle string as provided by STRING to be tested. It returns a yes-no decision regarding the acceptance or rejection of the input string.

The input object string and the automaton against which it is to be tested are read into RECOG's memory. The elements of the input string are then sequentially applied to the automaton transition table. If, after the final string element is applied to the transition table, the current state is one of the final states, the string is accepted as belonging to the class represented by the automaton. Otherwise, it is rejected.

For complete recognition, an input string must be tested against all object class automata. If the string is accepted by two or more automata, the input string falls into a class overlap and

semantic considerations must be used to resolve the disparity. A simplified flowchart of RECOG is shown in Figure 19, Appendix A. The complete FORTRAN program listing is given in Appendix B.

Execution speed of the RECOG module is very fast (under one second) but is obviously related to the length of the string to be recognized. The recognition process will take longer as more pattern classes are added since each must be considered separately for every input string.

SEMAN

Finally, SEMAN accepts as input the semantic information file for an object which falls into a class overlap. Using this information and the class separation routine developed for the particular classes involved, SEMAN attempts to assign the object to one of the overlap classes.

The class separation routines are derived heuristically by considering each overlap case individually when the original recognition classes are formed. The approach to finding a method of resolving class overlaps is to ask, "What, in a global sense, makes these similar objects different?" The semantic considerations used to resolve the class overlap are then developed through the human operator's knowledge of the object classes at hand. The process of injecting semantic considerations is also guided by the type of overlap between recognition automata discovered by the SEPAR module. The semantic considerations will naturally be different for different class overlaps. The tests used to resolve each overlap are

determined heuristically and are brought into play when each particular overlap is encountered. Thus the concurrent recognition of a test object by the automata for object classes 1 and 2 (for example) will key the semantic module to apply the semantic considerations which were predetermined for overlaps between classes 1 and 2. The SEMAN module accepts as inputs the attribute file for the object which is causing the overlap and the overlap class numbers. SEMAN then calls the appropriate routine for semantic resolution of the class overlap. The SEMAN program is written in such a way that the addition of new semantic recognition routines is easily accomplished. Examples of specific semantic tests are given in Chapter VI, Experimental Results. A simplified flowchart of SEMAN is shown in Figure 20, Appendix A. The complete FORTRAN program listing is given in Appendix B.

The simple global semantic considerations implemented during testing (see Chapter VI) yield very fast (under one second) execution times for the SEMAN module. If more extensive semantic techniques are needed, the execution times of this module would be expected to lengthen.

CHAPTER VI

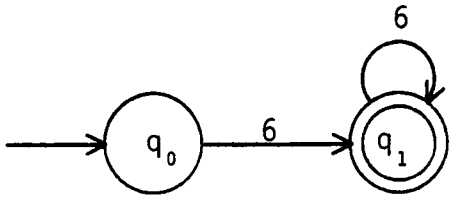
EXPERIMENTAL RESULTS

Size Generalization

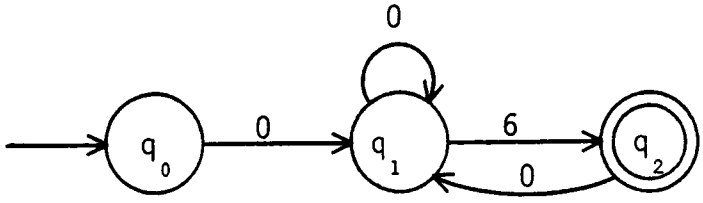
This experiment was designed to demonstrate the power of the K-Tail Algorithm to generalize from a sample set of various sized squares to accept squares of any arbitrary size. Four squares of unequal sizes were used as the sample set. Their angle string representations were derived via the STRING module and supplied as input to the KTAIL module which in turn derived the recognition automata for several values of k. SEPAR was not used in this experiment as only one class was under consideration.

The sample set of four string representations of squares was first applied as input to RECOG to verify that each recognition automaton would accept its own generating sample set. In every case, the generating sample set was accepted. The angle strings for squares of other than the ones from the sample set were then input to RECOG for automata generated for values of k from 0 to 4. All of the squares were recognized by each of these automata.

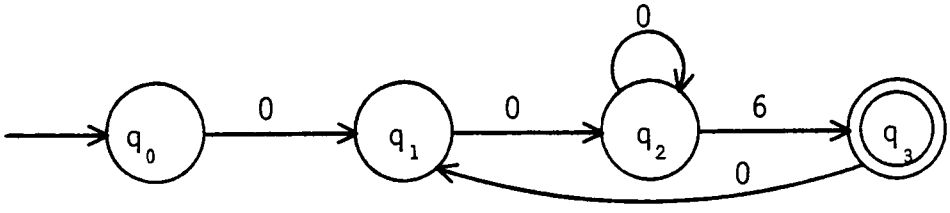
The recognition automata for the squares for k = 0, 1, and 2 are shown in Figure 10. For k = 0, any string ending in a 6 (which is a right-turn right angle, see Figure 9, page 50) is accepted as a square. This broad generalization from the sample set of squares is not very useful--many shapes other than squares can be recognized. For k = 1, strings of the form



A. k = 0



B. k = 1



C. k = 2

NOTE: Transitions for any input not shown go to state q_0.

Figure 10. Squares Automata.

$(0^a 6^b)$ $\quad a \geq 1, b \geq 1$

are accepted. This represents the set of squares with sides consisting of at least one angle 0 (two straight segments). For k = 2, the accepted strings have the form

$(0^a 6^b)$ $\quad a \geq 2, b \geq 1$.

In this case the squares must have sides consisting of at least two angle 0's. The generalization from the sample set is more limited in this case as evidenced by the fact that the smallest square in the sample set had sides with two angle 0's. The automata for k > 2 are even more specific, but are not needed for recognition since any strings to be recognized will have been derived from closed figures.

Note that the automata presented above will also accept rectangles. To be able to recognizes squares but reject rectangles, a recognizer would be required to consider the whole object boundary to determine that all the sides were the same length. As was stated when the K-Tail Algorithm was first presented, if k matches the length of the sample set strings, only the sample set strings will be recognized. Thus, small values of k are used to derive the recognition automata for the square strings. If it were necessary to be able to distinguish squares from rectangles, semantic considerations would be used.

Length Generalization

This test is similar to the size generalization experiment except that rectangles which varied in length only were used as the

sample set. This provides an indication of the K-Tail Algorithm's ability to generalize in only one dimension instead of two. Four rectangles of varying lengths (but equal widths) were used in the sample set. Again, the sample set was used to test each automaton with complete recognition being achieved. Other rectangles of the same width but different lengths were then input to the recognizer for various values of k and all were recognized.

The recognition automata for the rectangle strings for k = 4 and 5 are shown in Figure 11. For k = 0, 1, 2, and 3, the recognition automata which were derived for the rectangles are the same as the automata for the square strings. This is reasonable to expect, since squares are special cases of rectangles and a high degree of generalization takes place at low values of k.

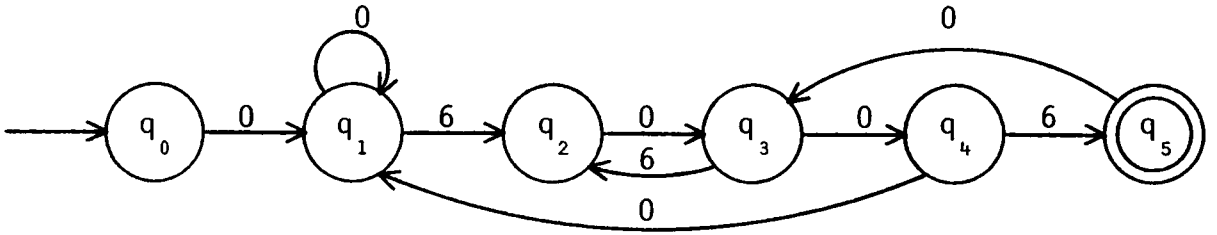
For k = 4, strings of the form

$(0^a6006)^b$ $\quad a \geq 1,\ b \geq 1$

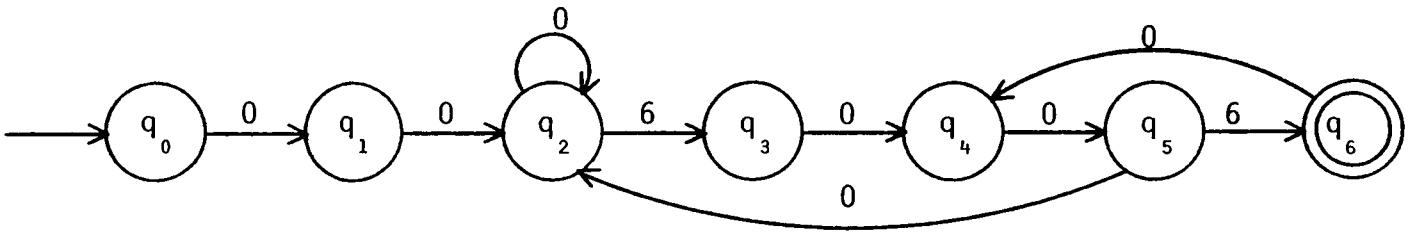
are accepted by the automaton. Note that every other side must have two consecutive 0 angles. This agrees with the sample set of rectangles which all had the same width--three segments (two 0 angles). For k = 5, the strings which are accepted have the form

$(0^a6006)^b$ $\quad a \geq 2,\ b \geq 1$.

Again, every other side must consist of two 0 angle strings. However, the intervening sides must now be at least three segments long, reflecting the fact that no rectangles in the sample set had a length less than its width. For values of k > 5, the requirements on the angle strings become more restrictive, but are not needed to recognize



A. k = 4



B. k = 5

NOTE: Transitions for any input not shown go to state q_0.

Figure 11. Rectangles Automata.

rectangles since it is assumed that all angle strings to be tested have been derived from closed figures.

Sensitivity to Rotation

To examine the sensitivity of the recognizer to the rotational orientation of objects presented to it, a test was performed using a bolt as the object to be recognized. This shape has four different possible starting points for an edge segmentation to be developed depending on the rotational orientation of the bolt. (The two corners at the tip of the threaded portion and the two corners at the top of the head can serve as starting points for the string representations.) The eight bolts included in the sample set were oriented from 0 to 315 degrees from horizontal in 45 degree intervals (defining 0 degrees as a horizontal bolt with the bolt head to the left). Again, complete recognition was achieved for the sample set.

For rotations of the bolt other than those of the sample set, complete recognition is not obtained. This is due in part to the Freeman chain coding. Since all object edges must be represented by the Freeman vectors, all edge angles are quantized by 45 degree increments. This quantization led to object distortion as the edge angles turned away from 45 degree alignments. If the bolt to be recognized was within ± 5 degrees of the 45 degree alignments, it was accepted by the automata generated for k's less than 3. When bolt was presented to the recognizer at orientations outside of this ± 5 degree spread, the results were erratic, with recognition being achieved at some rotations but not at others. This indicates that

for complete recognition of an object at any orientation angle, comprehensive training of the recognizer using a sample set containing samples of each training object through 360 degrees of rotation in increments of at most 10 degrees would be required. Other methods can be employed to overcome this sensitivity to rotation and will be discussed in the section on limitations.

Separation of Classes Via Semantics

To show the power of semantics for resolving class overlaps in recognition, four object classes were chosen for a test. The classes are bolt, screw (sheet-metal screw), wire splice (an insulated metal barrel into which two wires are inserted--the splice is then crimped to form an electrical connection), and wirenut (an insulated nut which is threaded onto two wire ends which have been twisted together--again to form a solid electrical connection). These particular objects were chosen because two class overlaps were expected, bolt-screw and splice-wirenut. The bolt and screw are very similar objects, differing mainly at the tip. The wirenut and splice are also similar objects, both having a generally rectangular shape.

The sample set consisted of three different samples each of bolts (Figure 12), screws (Figure 13), and wirenuts (Figure 14), and two different splices (Figure 15). Each item was oriented horizontally, with the bolt and screw heads to the left. Only one orientation of the objects was chosen to limit the storage requirements of the KTAIL program. For three objects in eight rotations each, twenty-four string representations would have to be stored to derive

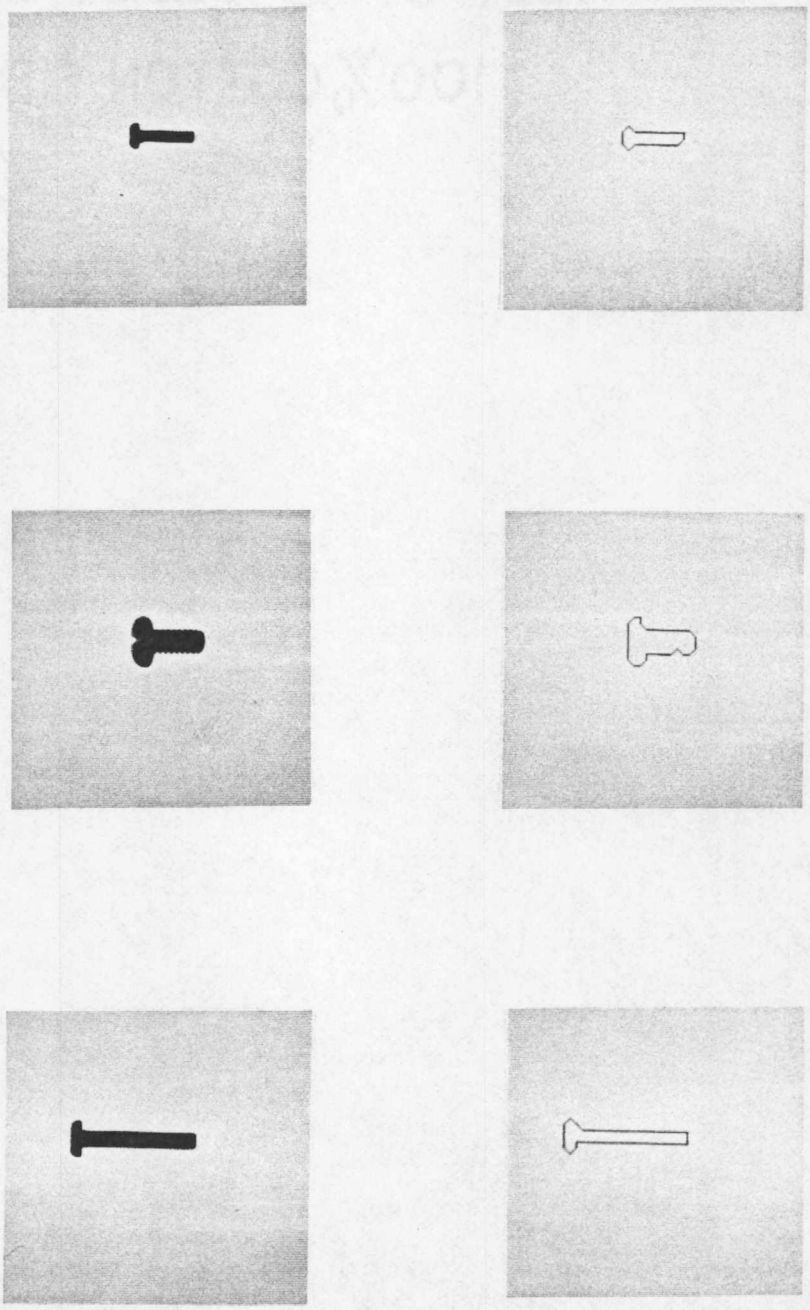


Figure 12. Bolt Samples and Edge Segmentations.

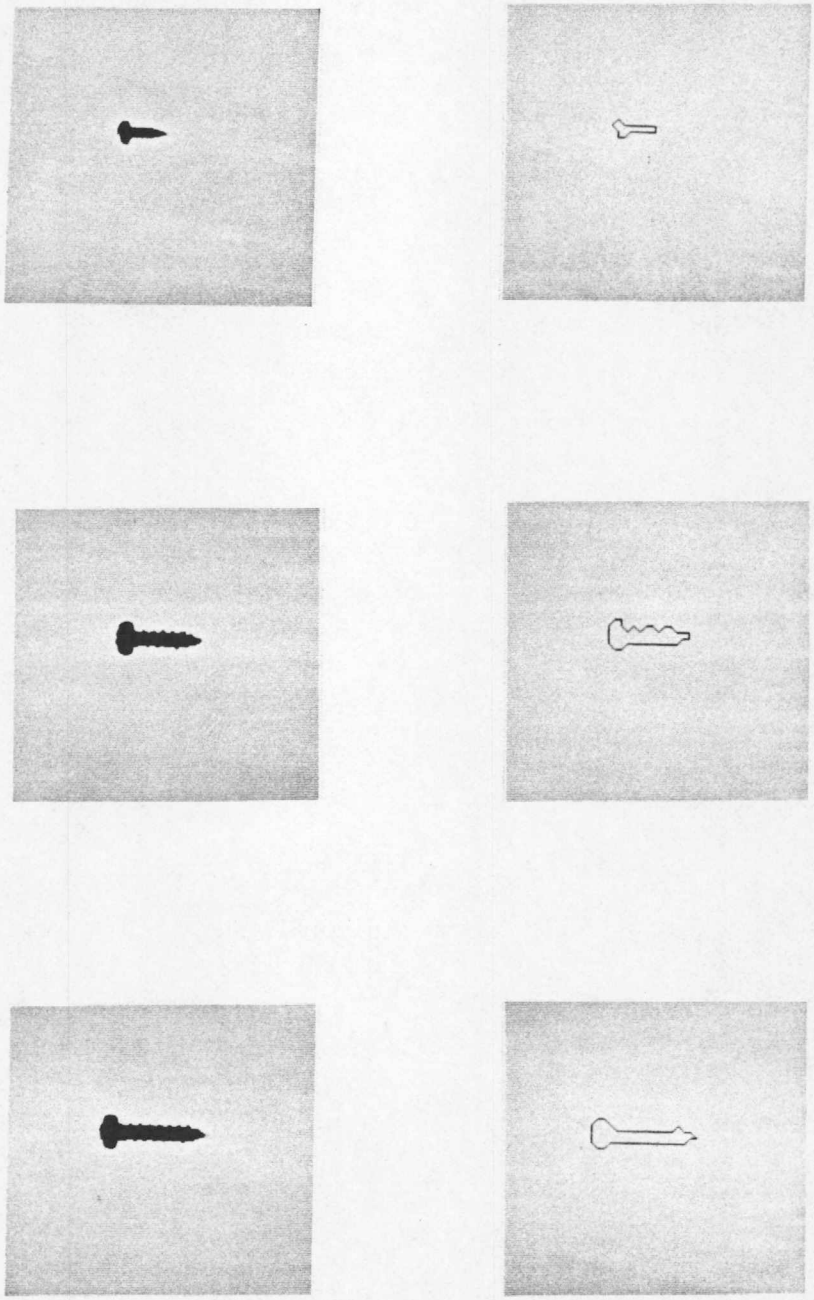


Figure 13. Screw Samples and Edge Segmentations.

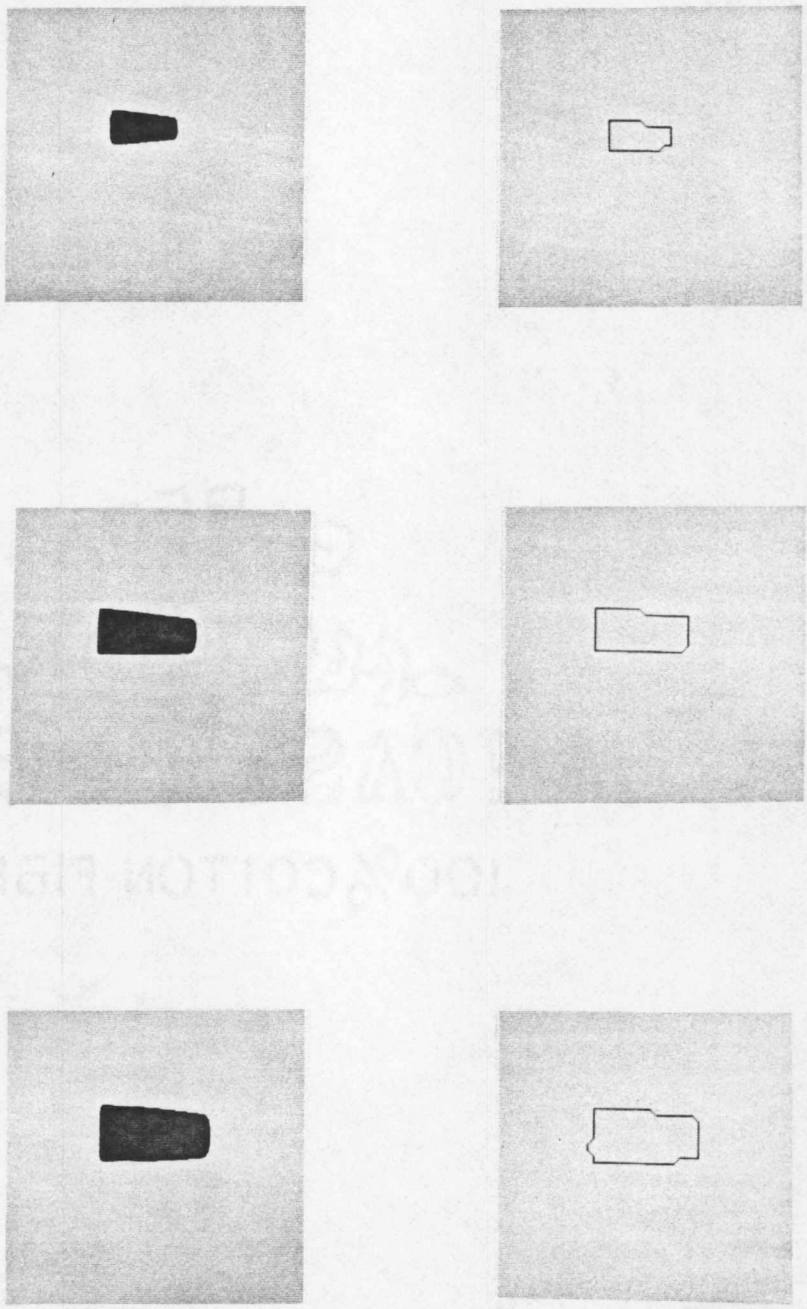


Figure 14. Wirenut Samples and Edge Segmentations.

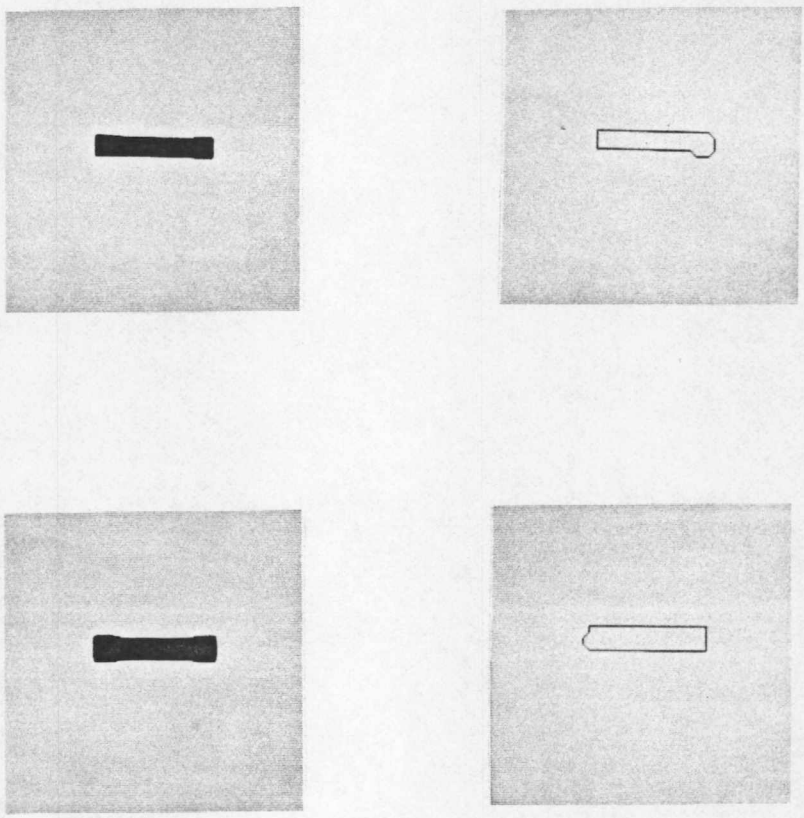


Figure 15. Splice Samples and Edge Segmentations.

the k = 0 automaton. Each increase in k would require twenty-four more strings to be stored as the original object string representations were truncated (refer to Chapter III: K-Tail Algorithm for Grammatical Inference). Thus, a single orientation for the objects was chosen for this test. An implementation on a machine with more core memory available than on the PDP 11/40 would allow several rotations to be considered.

The recognition automata were derived for the sample set for several values of k. The resulting overlaps are tabulated below. It can be seen that the automata erroneously recognized similar objects for the lower values of k. The k = 0 case was ignored because of its great generality.

Recognition Automata Overlaps for Bolt, Screw, Splice, and Wirenut

Class	Also Accepts	K
Bolt	Screw	1 - 3
Screw	Bolt	1 - 3
Splice	Wirenut	1 - 2
Wirenut	Splice	1 - 2

If an object is recognized by more than one automaton, semantics are turned to for the final recognition. As stated earlier, a bolt and a screw are two very similar objects. The main differences between them are that a bolt has a blunt end and a relatively fine thread while a machine screw has a pointed end and a coarser thread. The segmentation used for the bolts and screws was too coarse to catch the character of the pointedness of the ends of the parts with consistency and was certainly not able to discern the differences in

thread pitch. Rather than decrease the edge vector size to reliably convey the point characteristics, semantics are used to make the bolt-screw decision. In this way, the longer strings which would have resulted from the finer segmentation do not need to be handled and the computational overhead for string derivation and recognition is kept to a minimum. Since the point is the most obvious difference between a bolt and a screw, this feature was chosen for the semantic approach. It was noticed that both the bolt and the screw have lengthwise symmetry. This fact leads to the observation that there will be an even number of corners on a bolt but an odd number of corners on a screw since the point falls on the axis of symmetry. Instead of corners, the number of points which were local maxima of the function of distance from the object centroid to the boundary as the boundary was followed were counted and used as the semantic criterion. Note that this is a global, quantitative measure which cannot be handled by syntactics.

It has already been stated that the splice and wirenut both have generally rectangular shapes. However, splices are long and thin while wirenuts are comparatively short and squat. The result of the perimeter squared divided by the area (PSA) for an object was used as a measure of the compactness of the object. The PSA is higher for splices than it is for wirenuts and allows separation of the two classes if it is known a priori that the object under consideration is either a splice or a wirenut. Note that the PSA is a global, quantitative measure which cannot be determined by syntactics.

Neither the number of object maxima points nor the PSA could be used to recognize the objects alone. The PSA for both bolts and screw falls in the same range and the number of maxima points for both splices and wirenuts is four (four corners). In addition, these measures cannot reject objects which do not belong to any of these four classes. To use these semantic measurements for object recognition, it must be known into which possible classes the object falls.

For a test of the semantic considerations' power to resolve class overlaps, the recognition automata for k = 1 were used. These automata entail the least computation for recognition and are therefore a logical choice for this case. If the automata for k larger than two or three were selected, the generalization might be too limited to correctly identify objects belonging to the recognition classes which were not included in the sample set.

Limitations

During testing of the different parts of this recognition scheme several limitations were encountered which must be considered in any practical implementation. The largest limitation of the STRING module is the time-consuming image processing required. On a PDP 11/40 the STRING module run time for a 512 X 512 image is approximately four minutes while the time for a 256 X 256 image is around one minute. It should be noted that although the use of a mini-computer is important for realistic testing of the recognizer (the PDP 11/40 is commonly used in industry), hardware realizations of various parts of the recognition scheme discussed in this thesis

can significantly decrease the time required to achieve recognition (see Chapter VII).

An obvious limitation of the STRING module is the fact that the selection of an appropriate vector length for edge segmentation of the objects requires human intervention, although this must only be done once for each class of subjects considered. The segmentation process itself introduces errors into the image representations. Sides of an object which do not lie along the segmentation vectors (which are in 45 degree increments) do not always get segmented into straight lines. Instead, the segmentation tends to oscillate about the line in a zig-zag fashion. Finally, to obtain a representative sample set of the objects to be recognized, each object must be presented to the STRING module in every orientation which would provide a different starting point for the edge segmentation.

The main limitation of the KTAIL module is its memory requirement. For each increment of k in the K-Tail Algorithm, the truncated object string and its k-tail must be stored. Thus, the combination of long strings and high values of k can cause KTAIL to quickly exceed the machine memory available. The other problem with the K-Tail Algorithm is its tendency to only generalize on the ends of the sample set strings. This is due to the way that the k-tail for each object string is formed from the end of the string. Thus, if the distinguishing features of the object appear near the beginning of the string representation, they may not be considered in generating the recognition automaton.

A difficulty with this recognition scheme is its sensitivity to rotational variations of the objects to be recognized. An attempt was made to overcome this problem by defining the object description in terms of the angles between the edge segmentation vectors instead of the vectors themselves. This approach has some limited success but is still limited by the angular quantization error inherent in the Freeman chain coding used. Although a comprehensive sample set for training the recognizer will help reduce this sensitivity to rotation, other methods can be used to overcome this problem. A well-known solution to the rotation problem is suggested in Chapter VII.

Finally, the SEMAN module has the obvious limitation that the determination and implementation of the semantic considerations for class overlap distinction are totally heuristically derived and manually formed into recognition algorithms. No learning algorithms currently exist for the selection of semantic features and the formation of semantic recognition procedures.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

The method described in this work for implementing a system for industrial computer vision using syntactic/semantic techniques will allow the construction of a useful parts recognizer. The learning feature of this recognizer, which is due to the use of the k-tail grammatical inference algorithm, allows the recognizer to be used after training with a simple, representative sample set of the parts involved. An exhaustive set of each and every part that may be encountered by this recognizer need not be available for training. This ability to generalize from a relatively few samples of an object class to be able to accept any object from that class is a major advantage of this scheme. The problem of class overlap which may be encountered due to this generalizing ability can often be resolved by the judicious use of semantic recognition techniques. These semantic considerations, being quantitative in nature, naturally complement rather than duplicate the main syntactic recognition algorithm. After an off-line training and testing phase, this parts recognition algorithm could be used in a practical industrial application. The requirements of presenting parts individually to the recognizer and having these parts backlighted to provide easy determination of their outlines are not too severe to make this method infeasible.

To make the parts recognition operation faster, the image processing phase of recognition (STRING module) should access the imager's memory or some other form of core memory instead of images written to disk. Since the actual segmentation need not be displayed in an operating recognizer, forming the segmented image can be eliminated at a great savings in processing time.

For greater reliability in the recognition of objects not oriented along the segmentation vectors, the sample set used for training the recognizer should include examples of the object at 10 degree rotation intervals. The rotation problem could also be helped by defining segmentation vectors at intervals of 22.5 instead of 45 degrees [Freeman (1970)]. An alternative solution to the rotational sensitivity of the recognizer is to pre-process the object images via the Hotelling transformation [Gonzalez and Wintz (1977)]. This transformation will result in an image which is oriented along its principal eigen-axes. Thus, for any object with discernable axes in an arbitrary rotation, the boundary representation can be developed for the object in a predictable orientation. (If the object has no discernable axes, the problem of rotational orientation does not occur.)

The sensitivity of this method of parts recognition to the accuracy of the edge segmentation could be greatly improved through the definition of pattern primitives other than the segmentation vectors themselves. These vectors could be subjected to a pre-processing operation which develops object edge strings consisting of

corners, bays, curves, lines, etc., much as in the recognition scheme by Gonzalez [1983]. These larger features would be more robust for the description of objects in the presence of noise and varying rotational orientations of the objects.

An improved grammatical inference algorithm would be of great benefit in making this recognition scheme more reliable for limited training sets. As mentioned earlier, the K-Tail Algorithm tends to concentrate its generalizations near the ends of the sample set strings. An inference algorithm is needed which would perform its generalization based on the whole string of each sample without restricting its generalization property at the same time.

The semantic portion of this parts recognition scheme obviously provides the most opportunity for further research. The area of semantic recognition is relatively new and is constantly evolving. Work should be directed to taking the human operator out of the semantics determination operation. A learning algorithm for semantics, though probably several years in the future, will be a major advancement in this area. The advantages of such a learning algorithm, such as the removal of heuristic considerations and the complete automation of the semantics determination process, are significant enough to warrant dedicated research toward the realization of a semantic learning algorithm.

This thesis should lay the groundwork for additional research along the lines described above. New developments can be tested by substituting them for the appropriate parts of this work. The

modularity of the implementation and the straightforward approach taken in the realization of the algorithms involved should allow other researchers to build from this work with little trouble. The results of this thesis work show the promise of learning algorithms for parts recognition which rely on both syntactics and semantics. With additional effort, very robust general purpose parts recognizers can be built.

LIST OF REFERENCES

LIST OF REFERENCES

Baird, M. L., and M. L. Kelly. "A Paradigm for Semantic Picture Recognition," Pattern Recognition, Vol. 6, No. 1, pp. 61-74, June 1974.

Biermann, A. W., and J. A. Feldman. "On the Synthesis of Finite-State Acceptors," Stanford Artificial Intelligence Memo AIM-114, April 1970.

Biermann, A. W., and J. A. Feldman. "On the Synthesis of Finite-State Machines from Samples of Their Behavior," IEEE Transactions on Computers, Vol. C-21, No. 6, pp. 592-597, June 1972.

Freeman, H. "Boundary Encoding and Processing," in Picture Processing and Psychopictorics, B. S. Lipkin and A. Rosenfeld, eds. New York: Academic Press, 1970, pp. 241-266.

Freeman, H., and L. S. Davis. "A Corner Finding Algorithm for Chain Coded Curves," IEEE Transactions on Computers, Vol. C-26, pp. 297-303, March 1977.

Freeman, H. "Application of the Generalized Coding Scheme to Map Data Processing," Proceedings of the IEEE Pattern Recognition and Image Processing Conference. Chicago, IL: May 31-June 2, 1978, pp. 220-226.

Freeman, H., and A. Saghri. "Generalized Chain Codes for Planar Curves," Proceedings of the 4th International Joint Conference on Pattern Recognition. Kyoto, Japan: November 7-10, 1978, pp. 701-703.

Gonzalez, R. C. "Syntactic/Semantic Techniques for Feature Description and Character Recognition," NORDA Technical Note 185, January 1983.

Gonzalez, R. C., and M. G. Thomason. Syntactic Pattern Recognition. Reading, MA: Addison-Wesley, 1978.

Gonzalez, R. C., and M. G. Thomason, personal communications, 1982.

Gonzalez, R. C., and P. Wintz. Digital Image Processing. Reading, MA: Addison-Wesley, 1977.

Hall, E. L. Computer Image Processing. New York: Academic Press, 1979.

Knuth, D. E. "Semantics of Context-Free Languages," Mathematical Systems Theory, Vol. 2, No. 2, pp. 127-145, 1968.

Lee, S. C. Digital Circuits and Logic Design. Englewood Cliffs, NJ: Prentice-Hall, 1976.

Pavlidis, T. "Algorithms for Shape Analysis of Contours and Waveforms," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, pp. 301-312, July 1980.

Pavlidis, T., and F. Ali. "A Hierarchical Syntactic Shape Analyzer," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 1, pp. 2-9, January 1979.

Tang, G. Y., and T. S. Huang. "A Syntactic-Semantic Approach to Image Understanding and Creation," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, pp. 135-144, April 1979.

Tou, J. T., and R. C. Gonzalez. Pattern Recognition Principles. Reading, MA: Addison-Wesley, 1974.

You, K. C., and K. S. Fu. "A Syntactic Approach to Shape Recognition Using Attributed Grammars," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-9, No. 6, pp. 334-345, June 1979.

You, K. C., and K. S. Fu. "Distorted Shape Recognition Using Attributed Grammars and Error-Correcting Techniques," Computer Graphics and Image Processing, Vol. 13, No. 1, pp. 1-16, May 1980.

APPENDICES

APPENDIX A

FORTRAN PROGRAM FLOWCHARTS

This appendix contains flowcharts of the FORTRAN program described in this thesis. The programs are STRING (Figure 16), KTAIL (Figure 17), SEPAR (Figure 18), RECOG (Figure 19), and SEMAN (Figure 20).

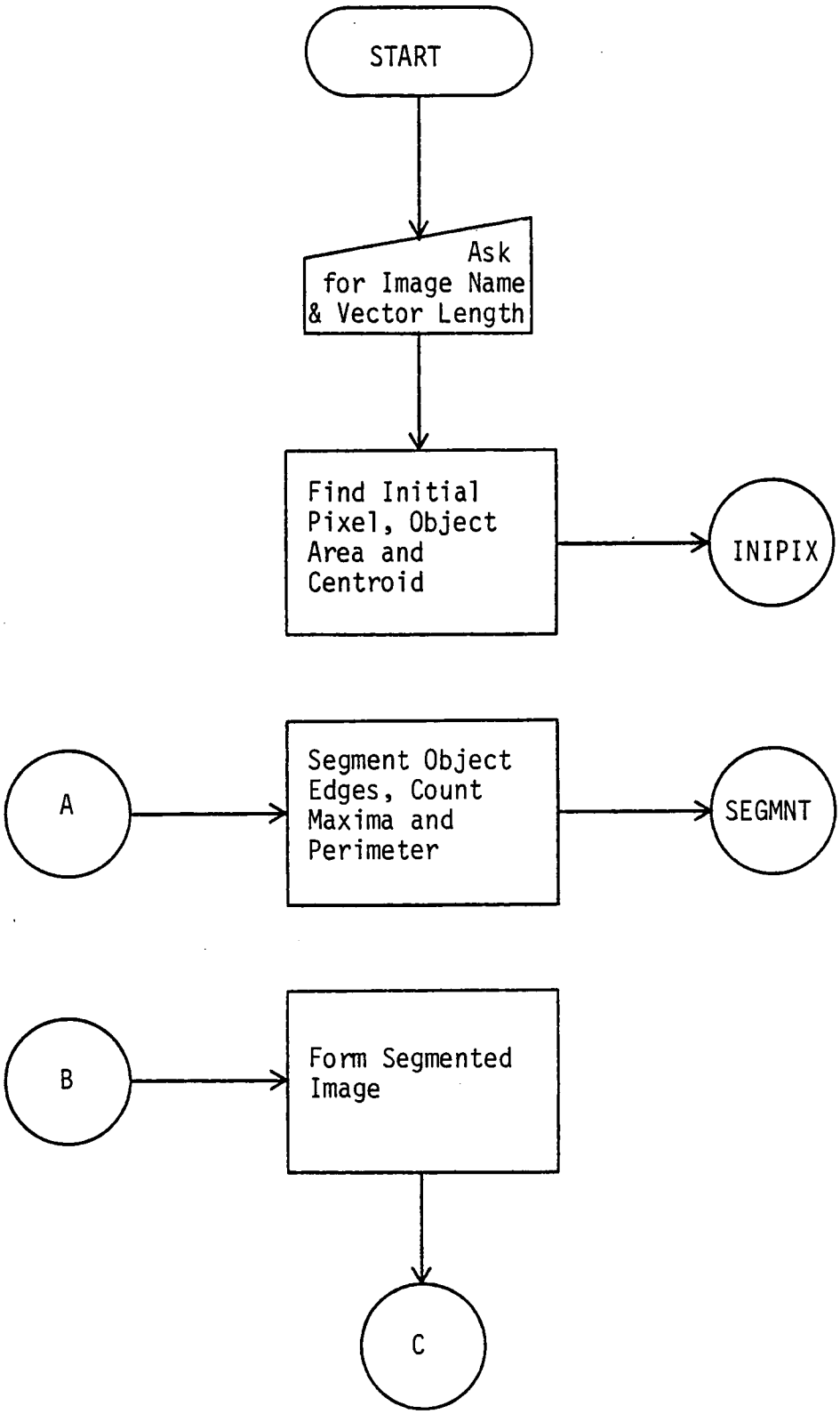


Figure 16. Simplified Flowchart of Program STRING.

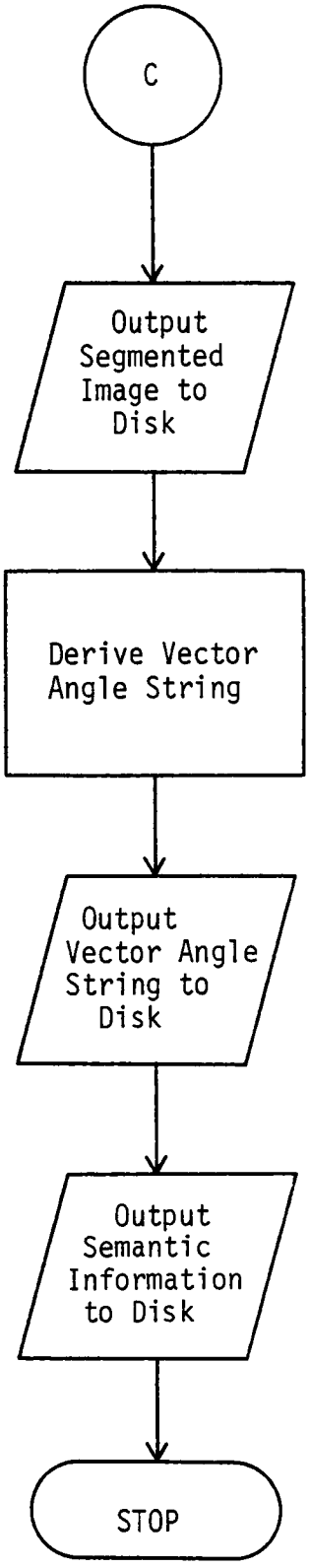


Figure 16. (Continued)

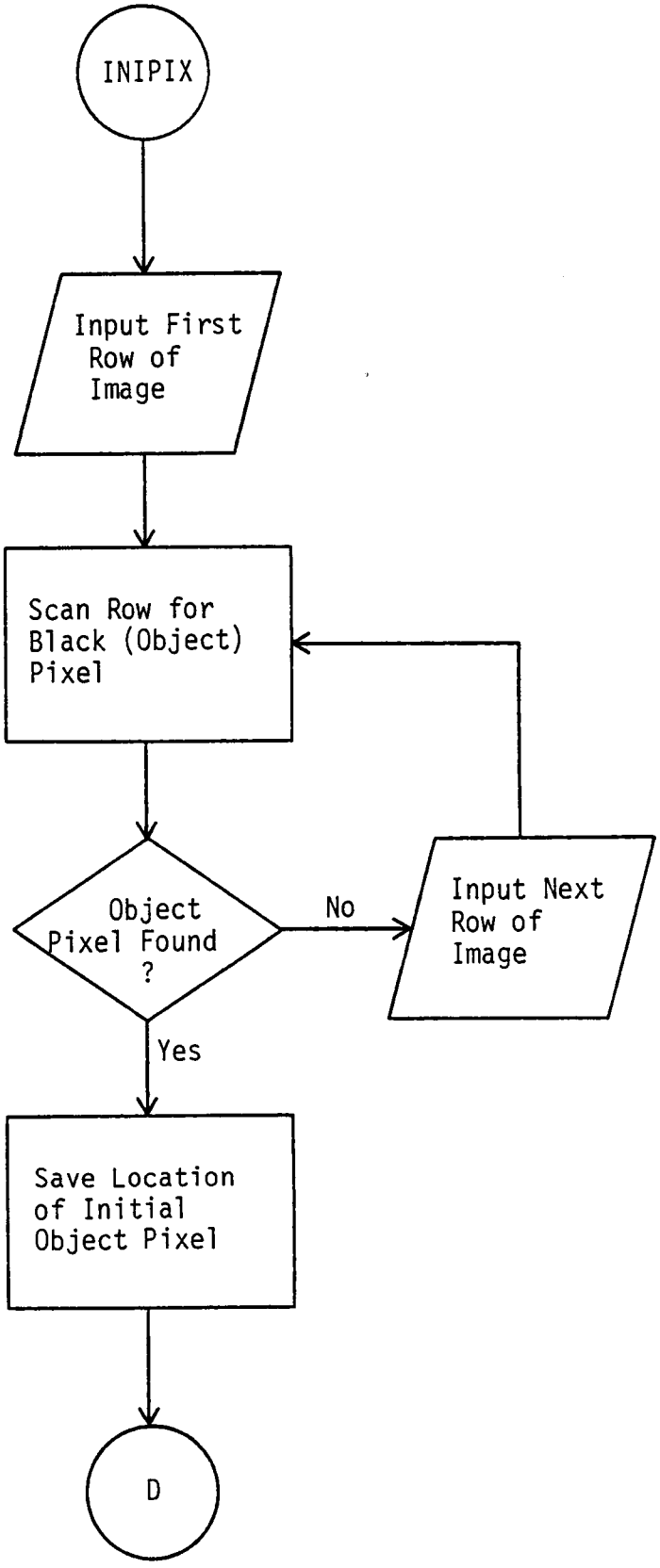


Figure 16. (Continued)

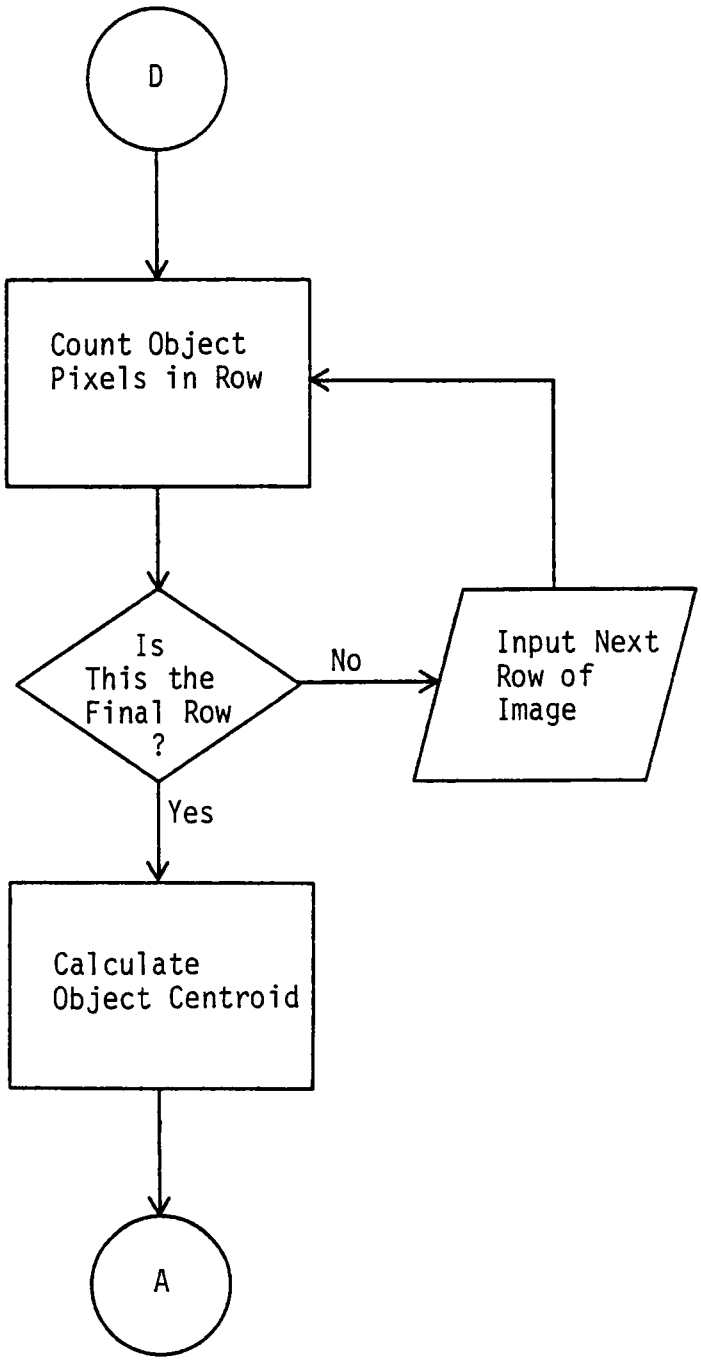


Figure 16. (Continued)

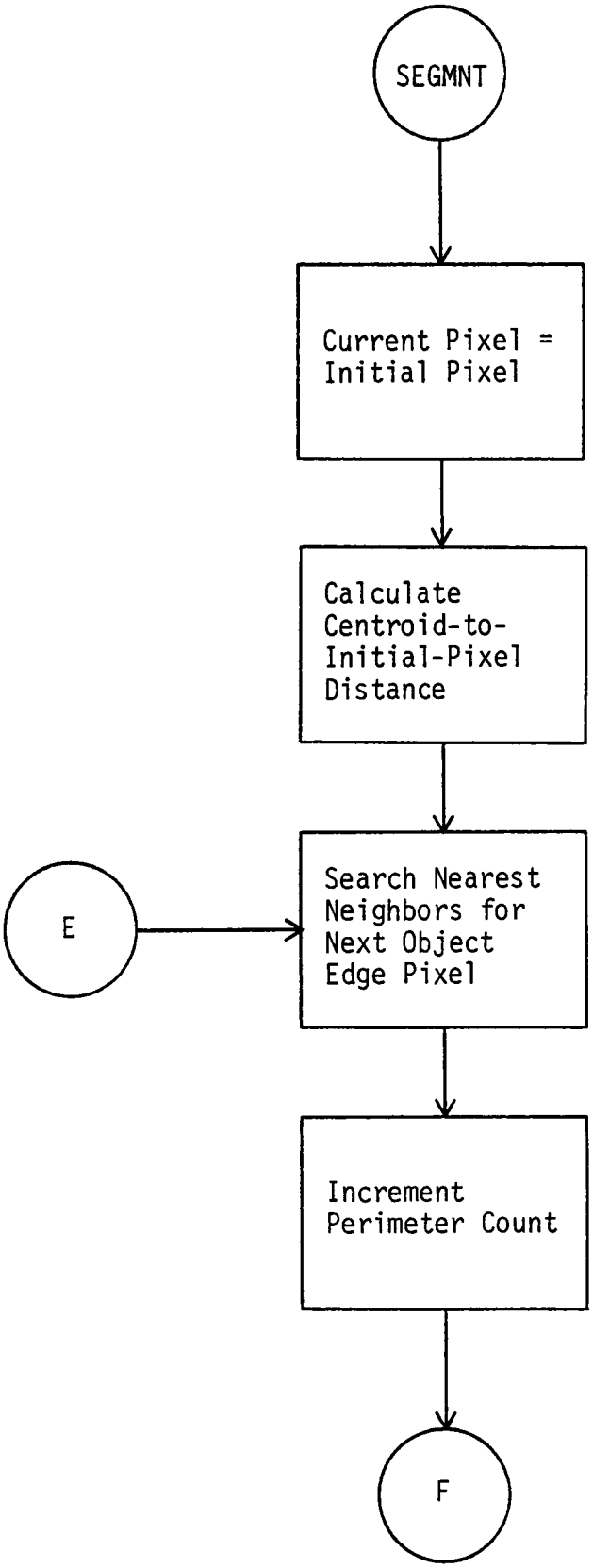


Figure 16. (Continued)

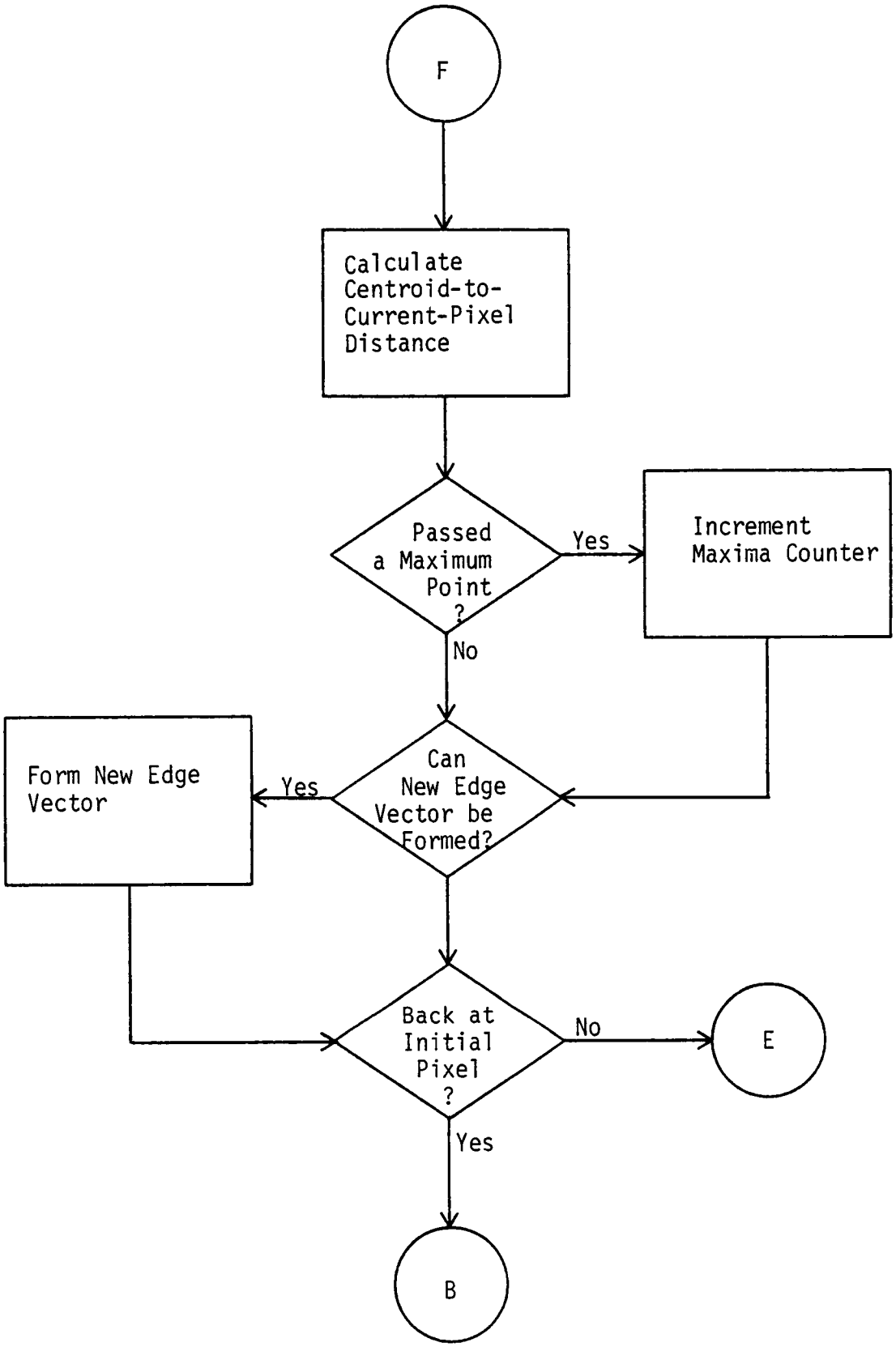


Figure 16. (Continued)

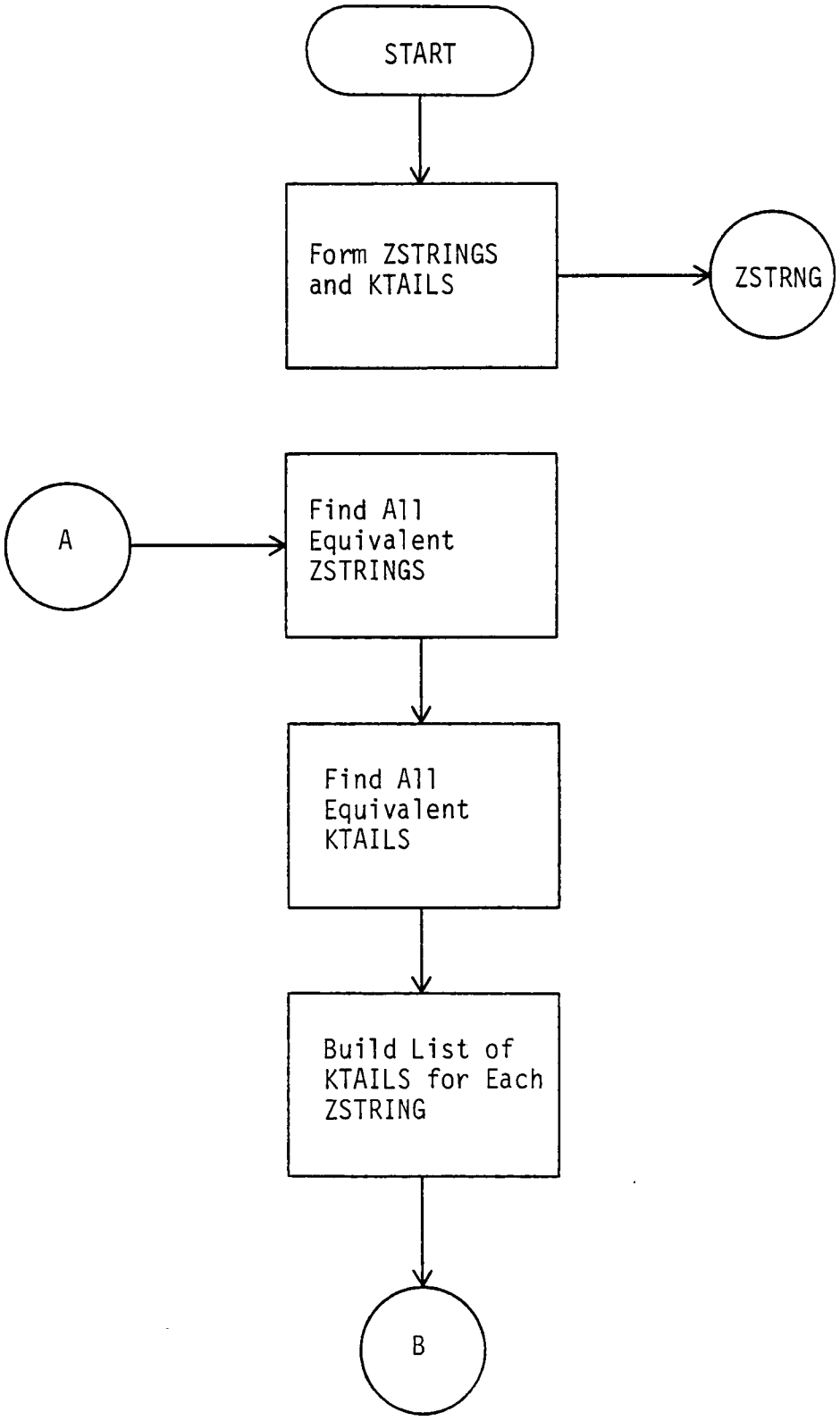


Figure 17. Simplified Flowchart of Program KTAIL.

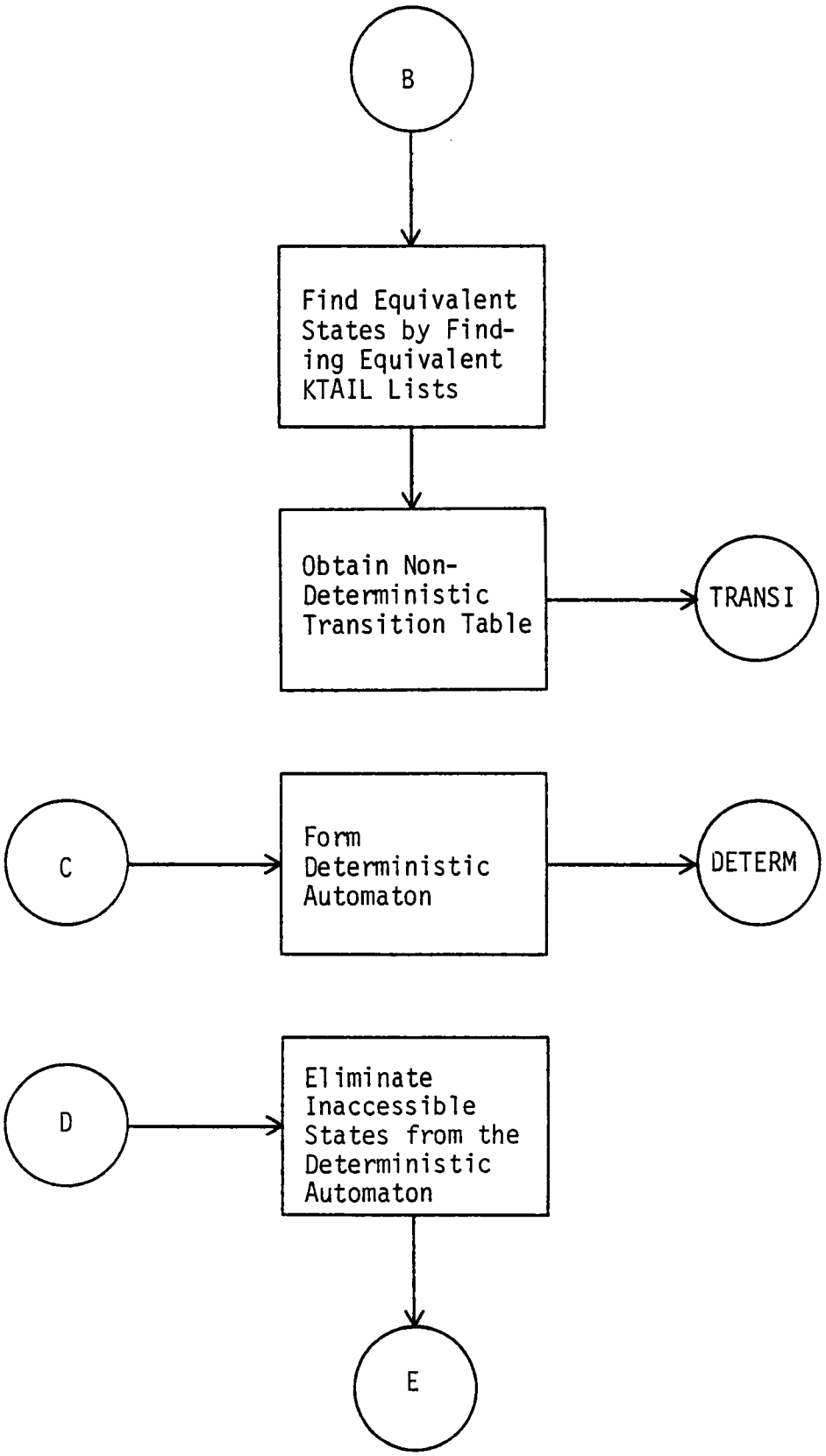


Figure 17. (Continued)

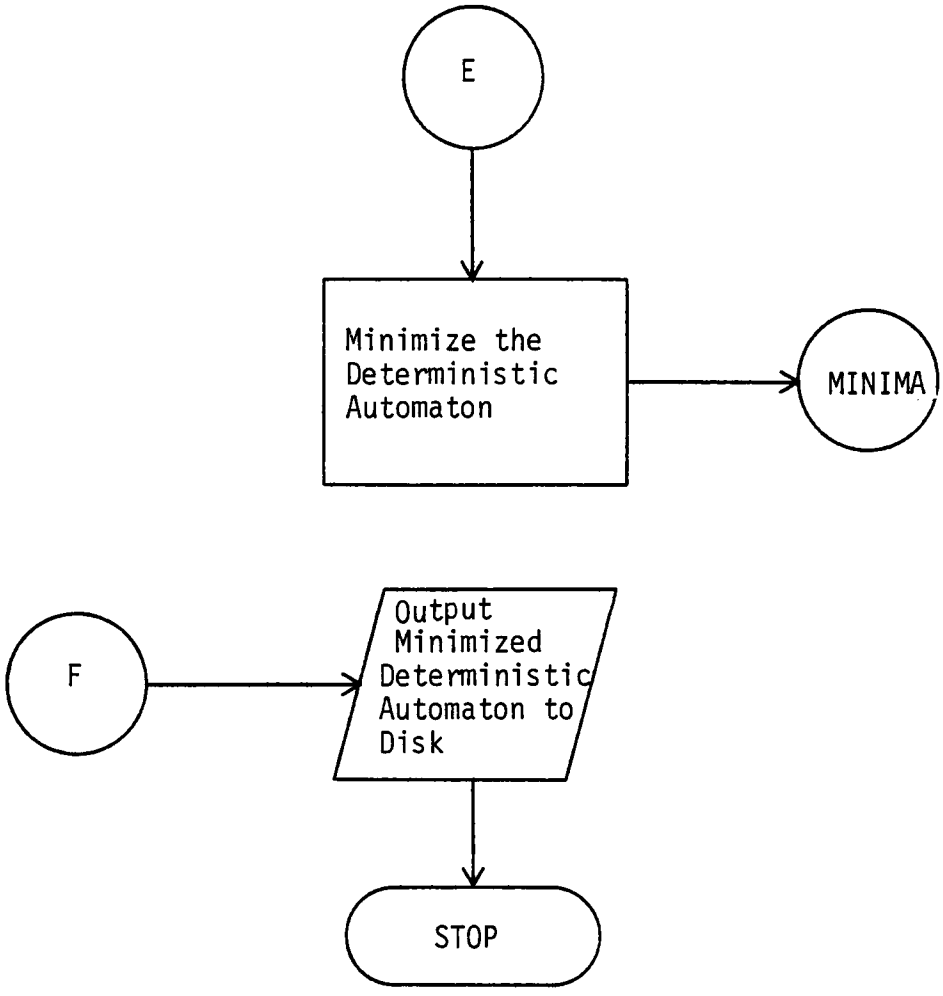


Figure 17. (Continued)

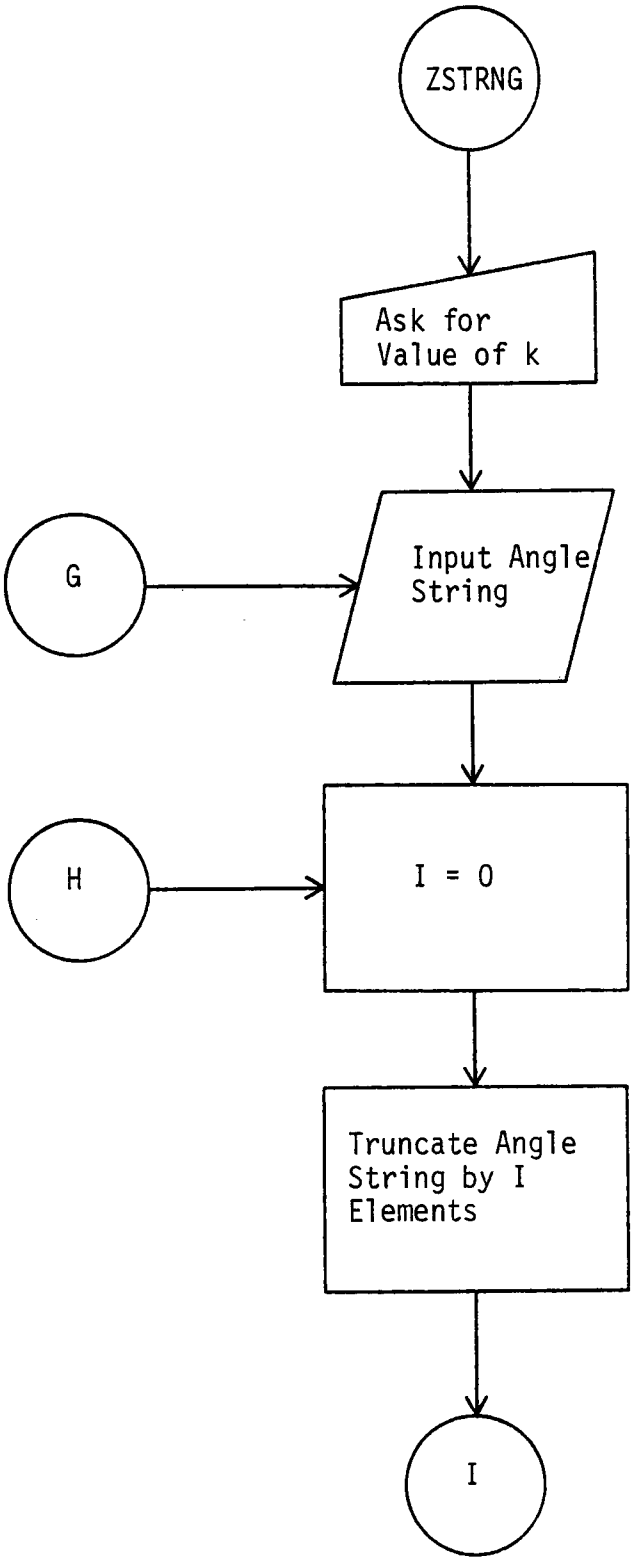


Figure 17. (Continued)

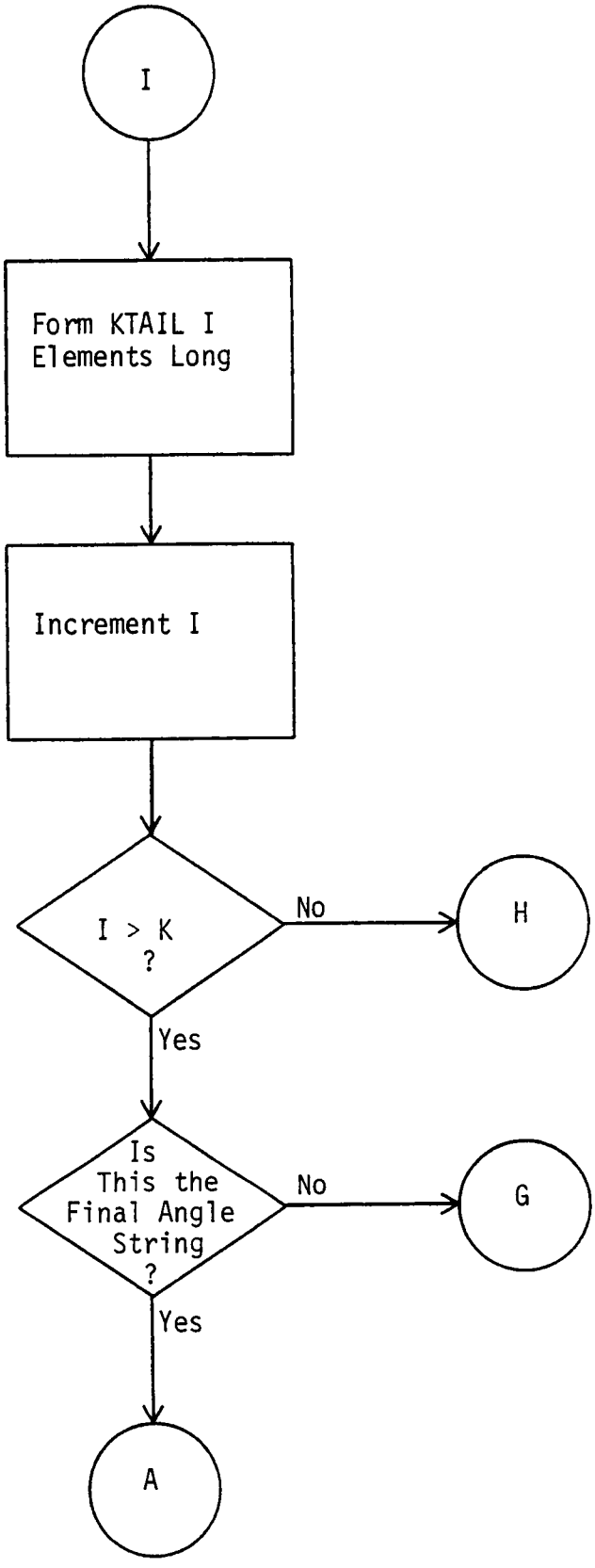


Figure 17. (Continued)

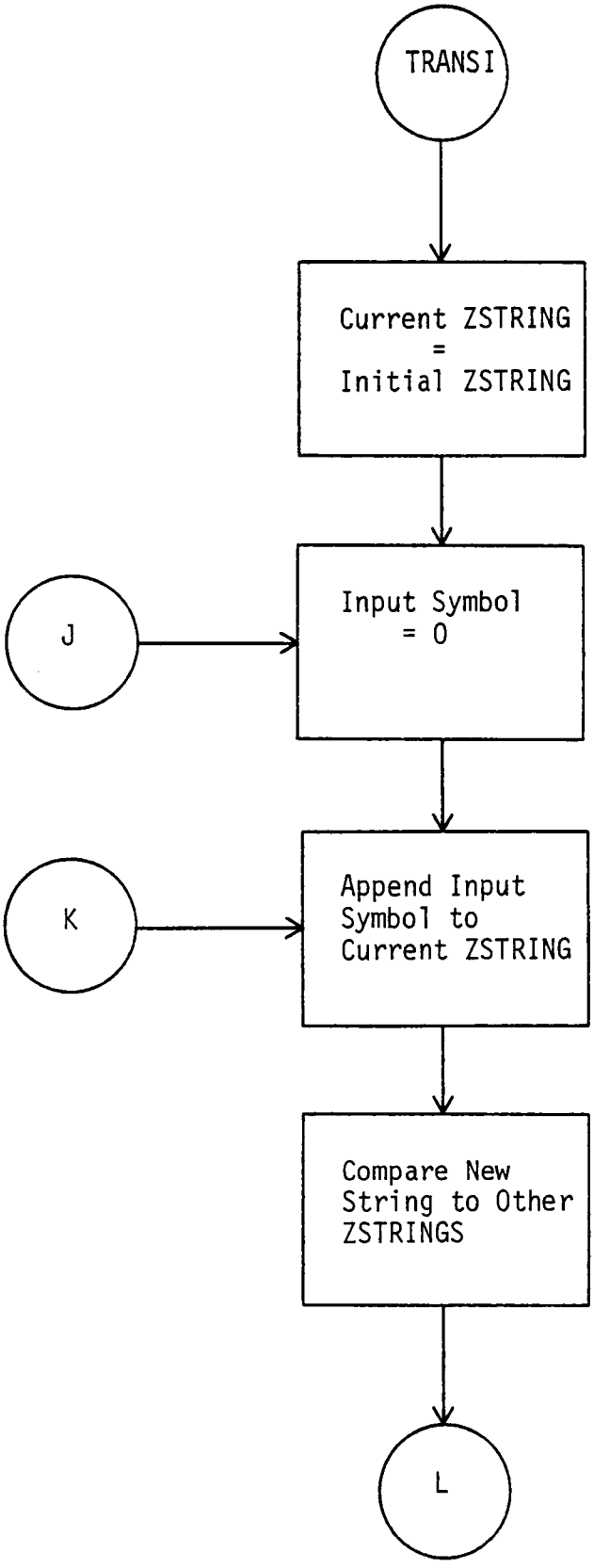


Figure 17. (Continued)

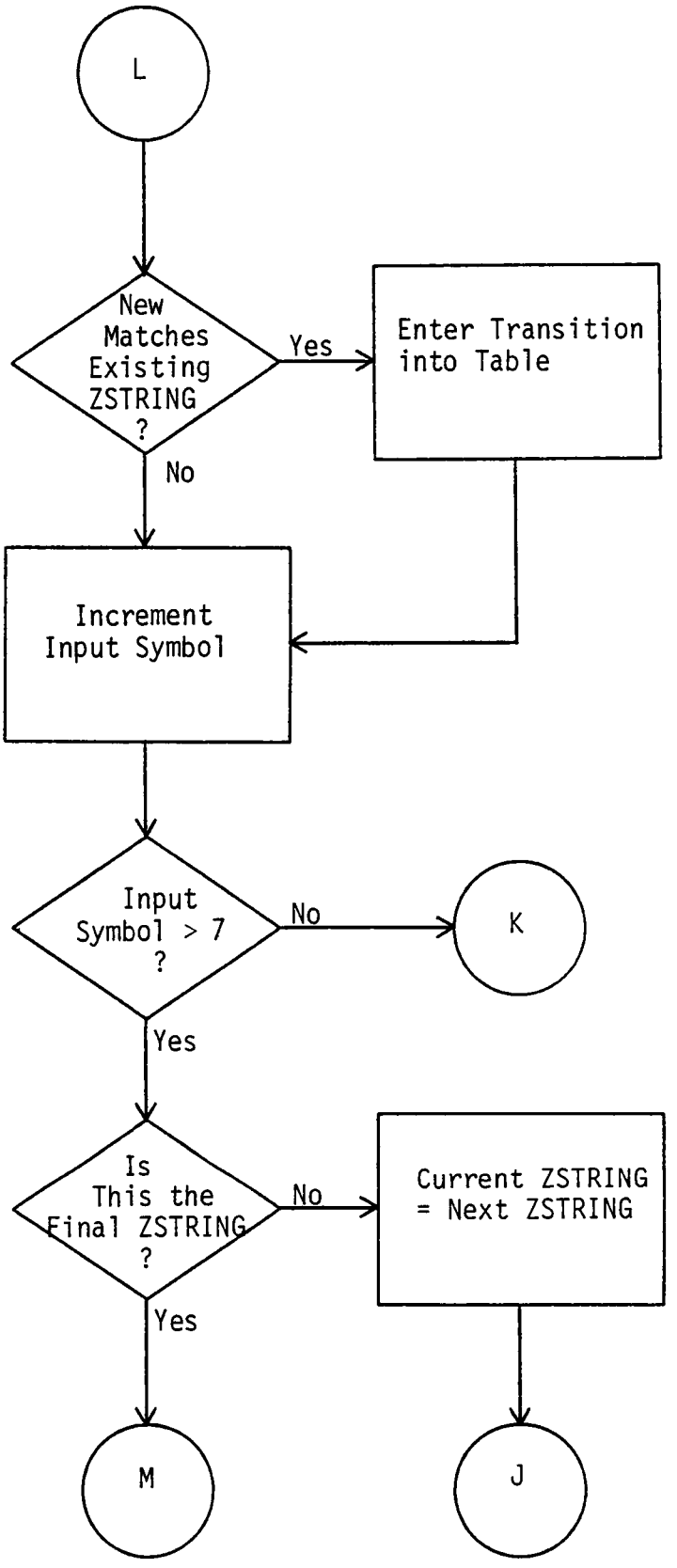


Figure 17. (Continued)

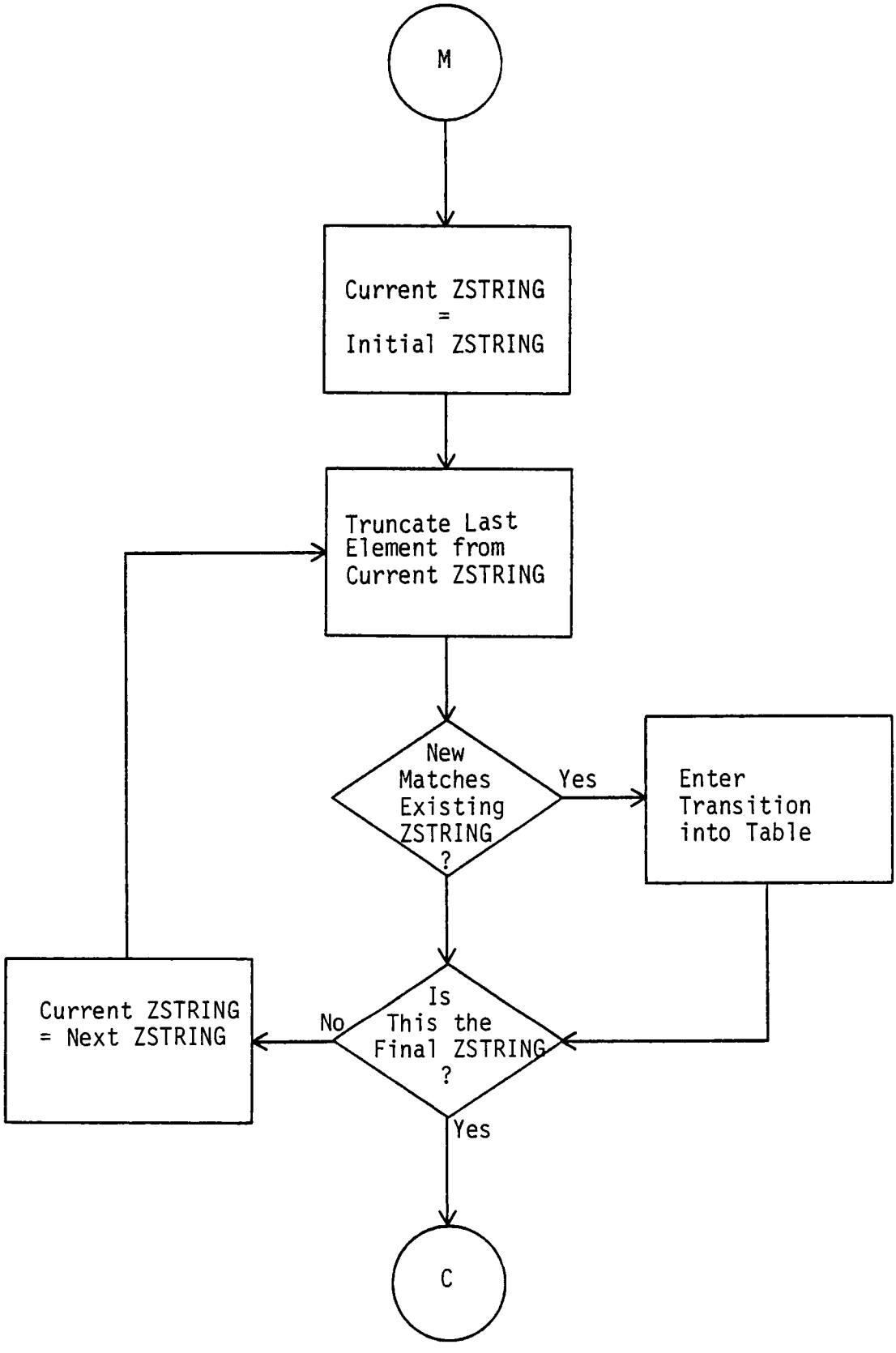


Figure 17. (Continued)

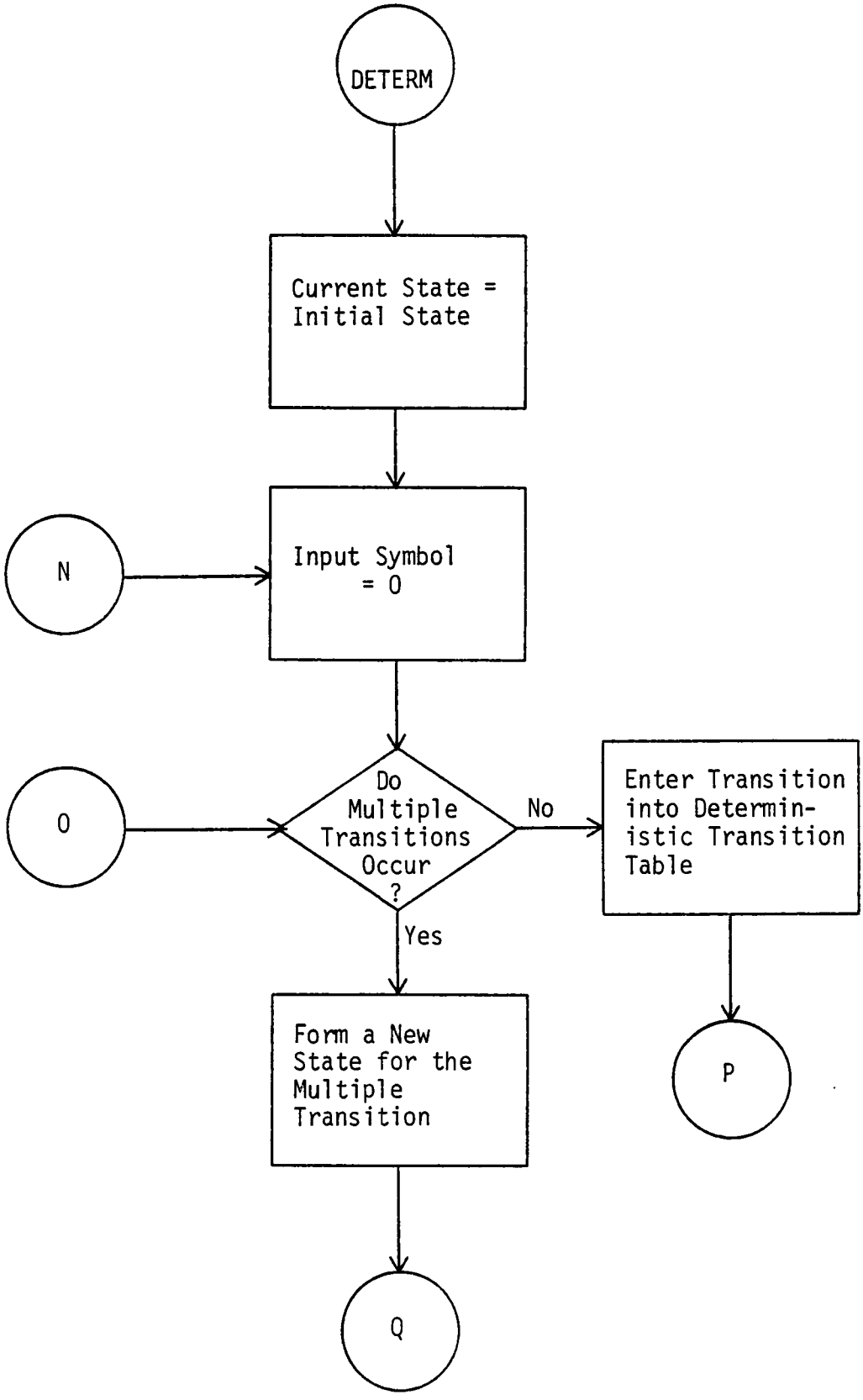


Figure 17. (Continued)

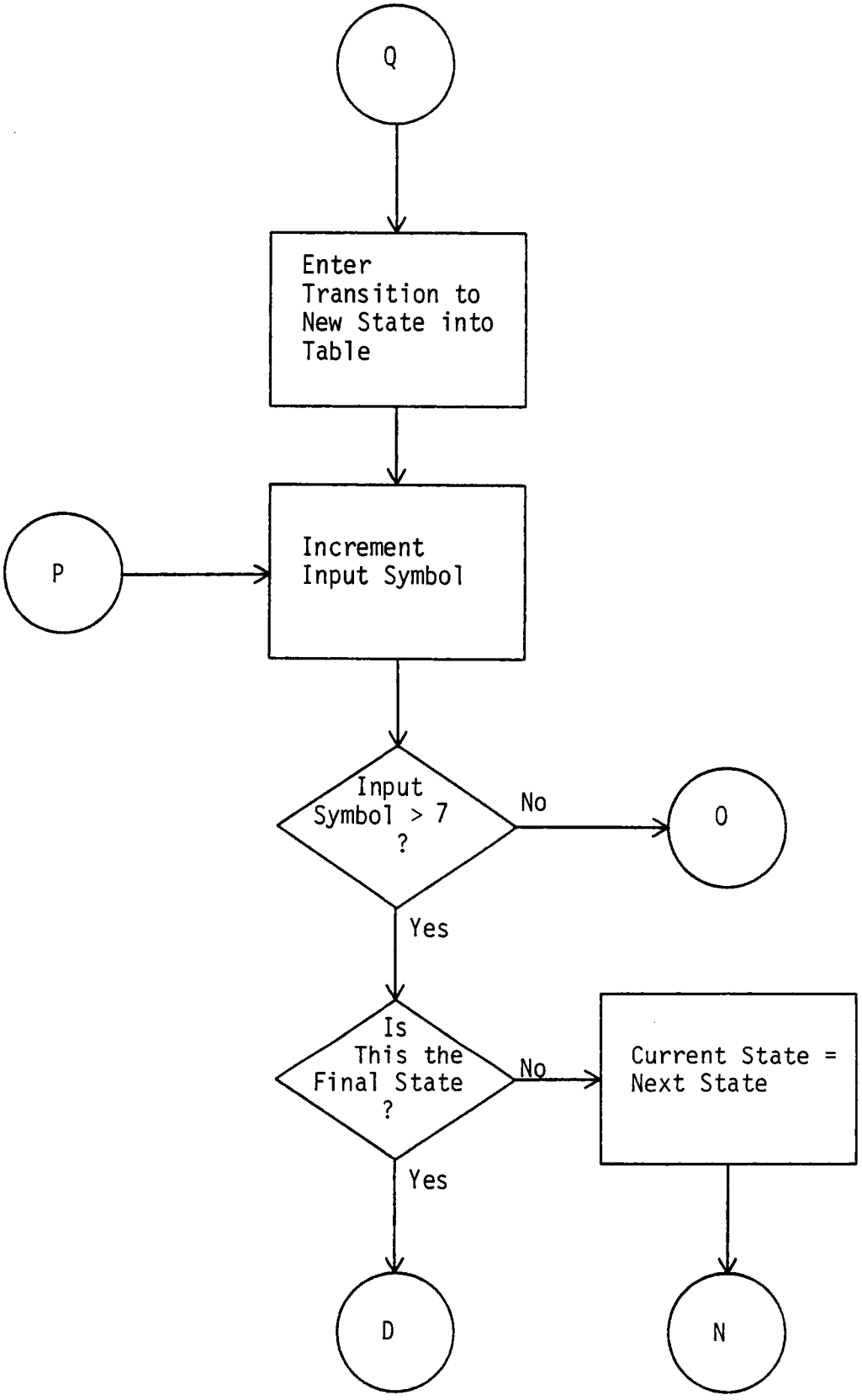


Figure 17. (Continued)

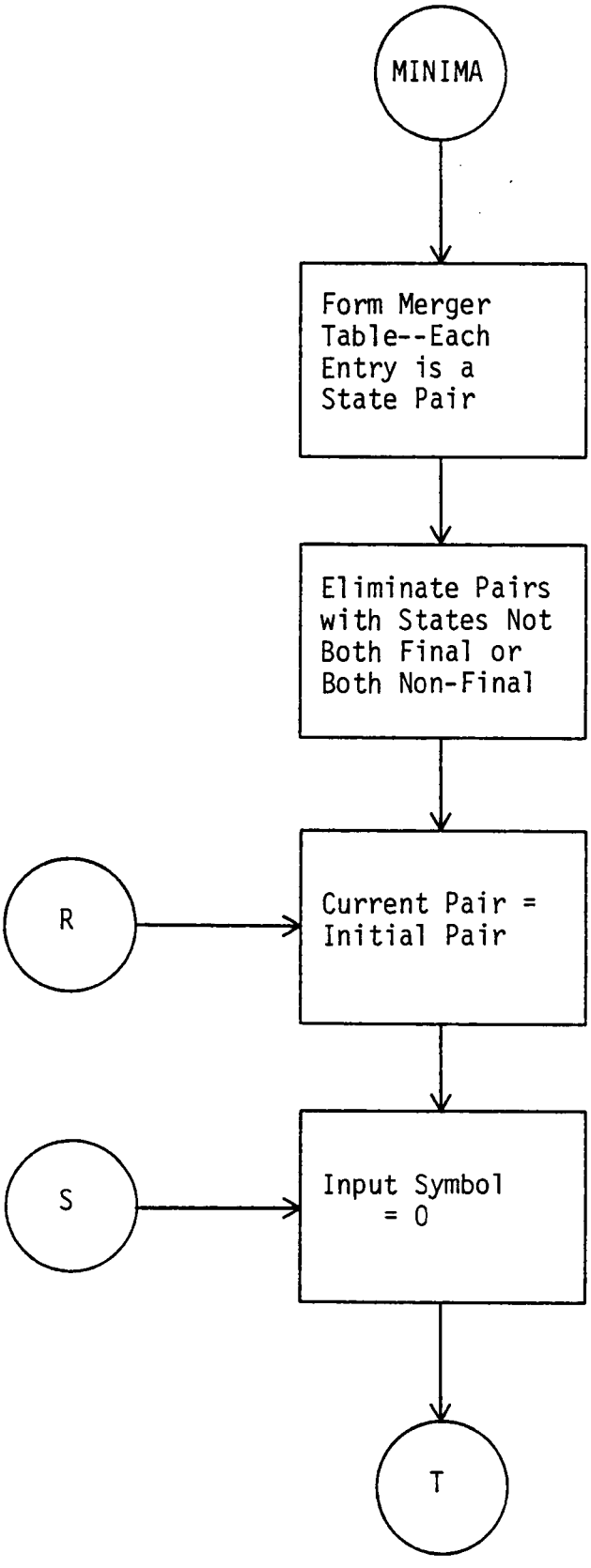


Figure 17. (Continued)

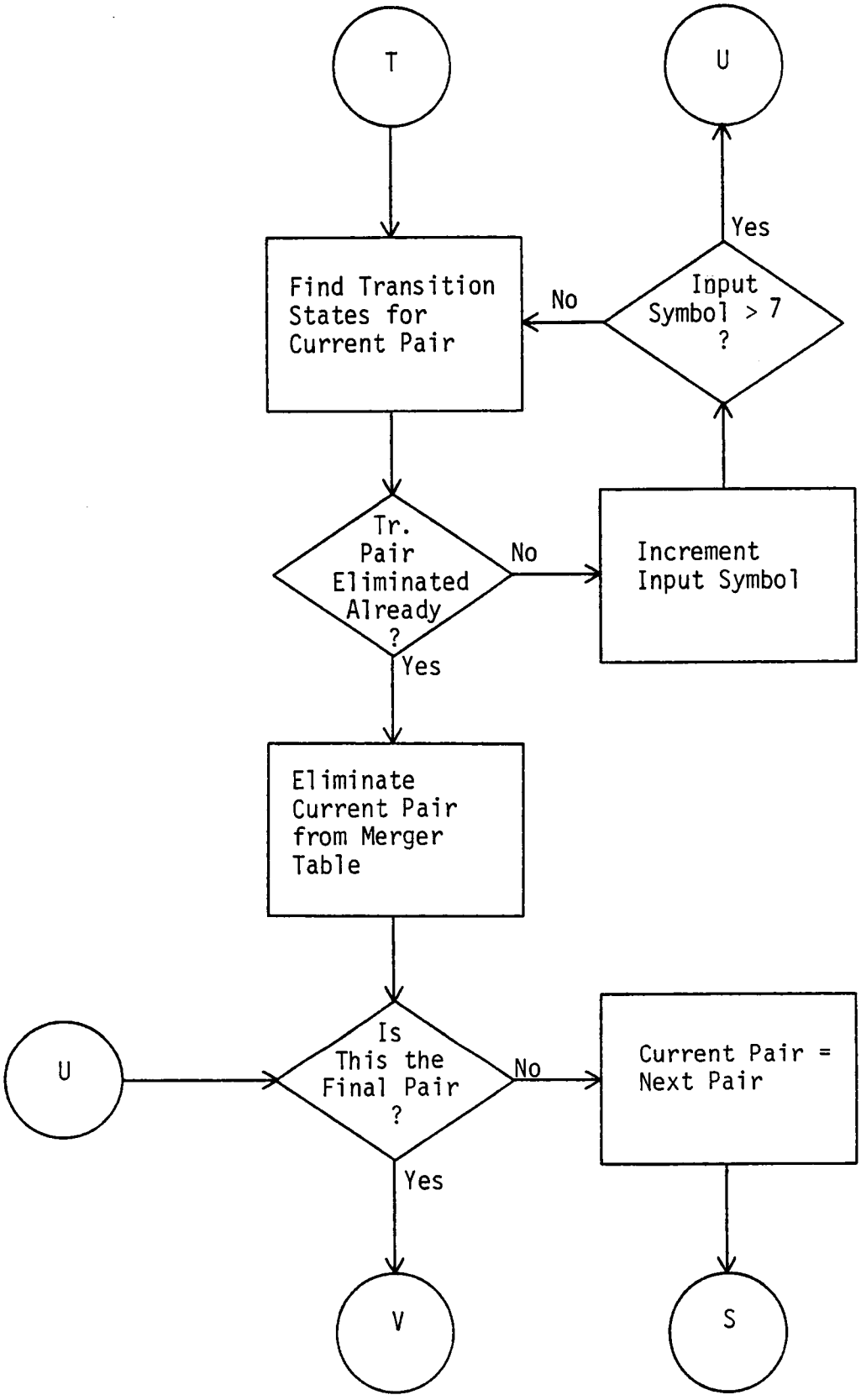


Figure 17. (Continued)

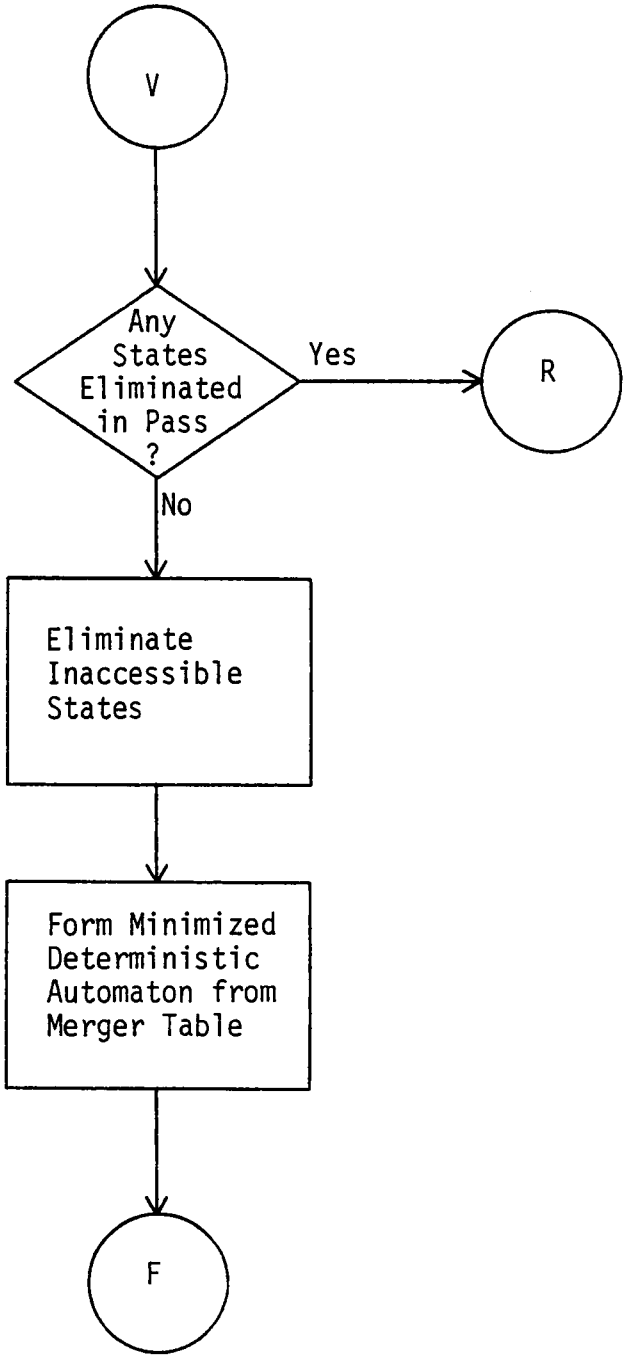


Figure 17. (Continued)

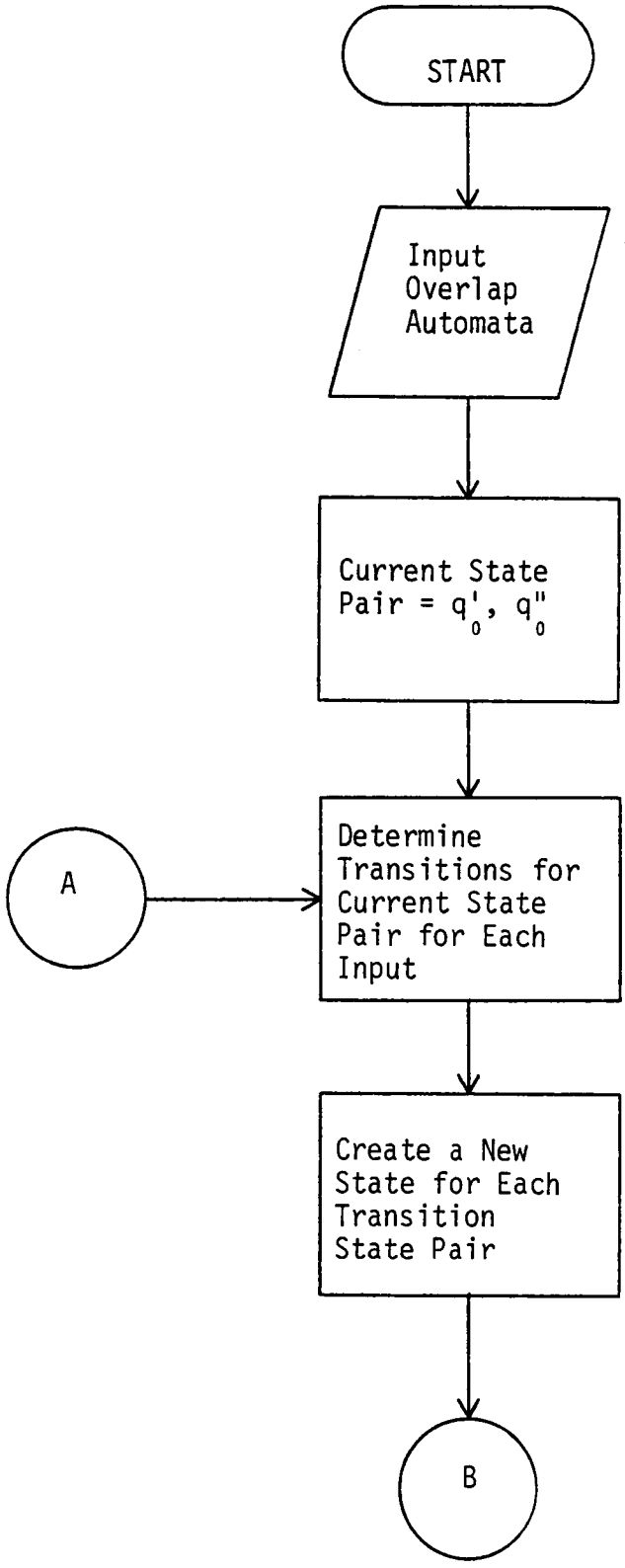


Figure 18. Simplified Flowchart of Program SEPAR.

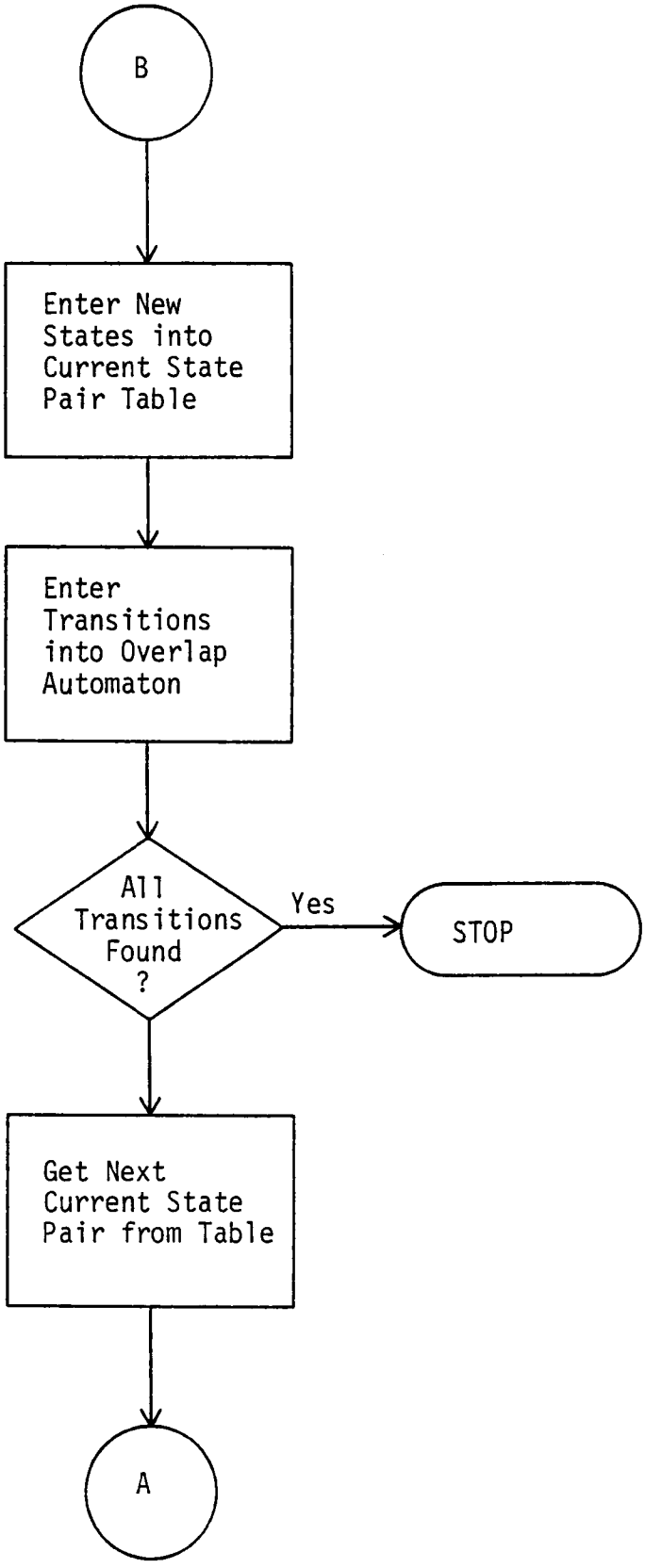


Figure 18. (Continued)

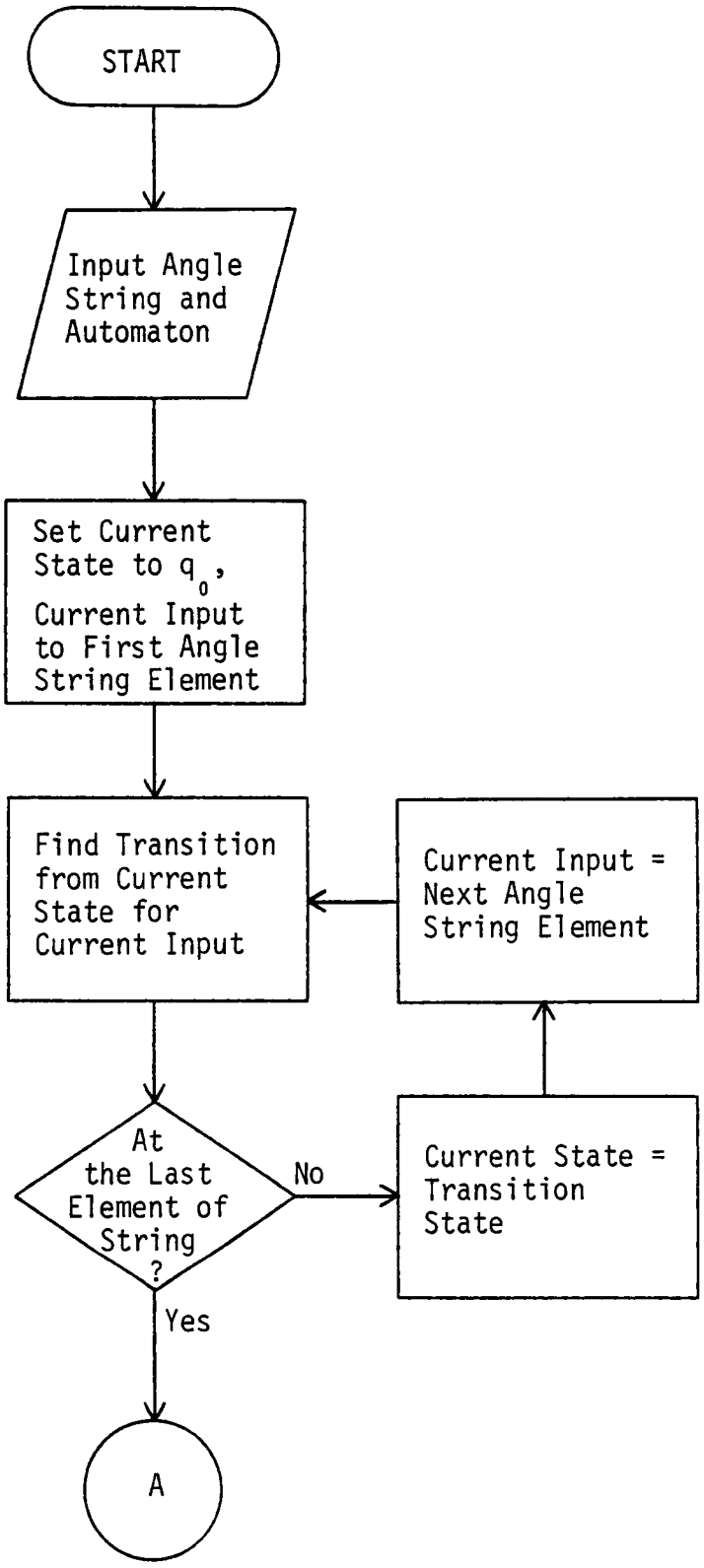


Figure 19. Simplified Flowchart of Program RECOG.

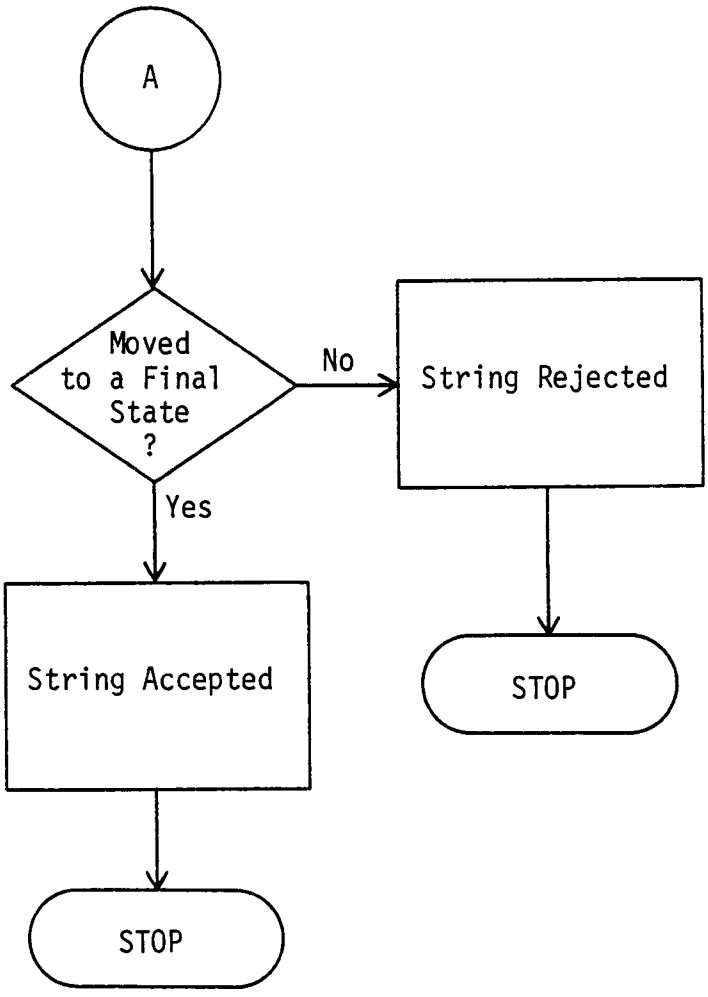


Figure 19. (Continued)

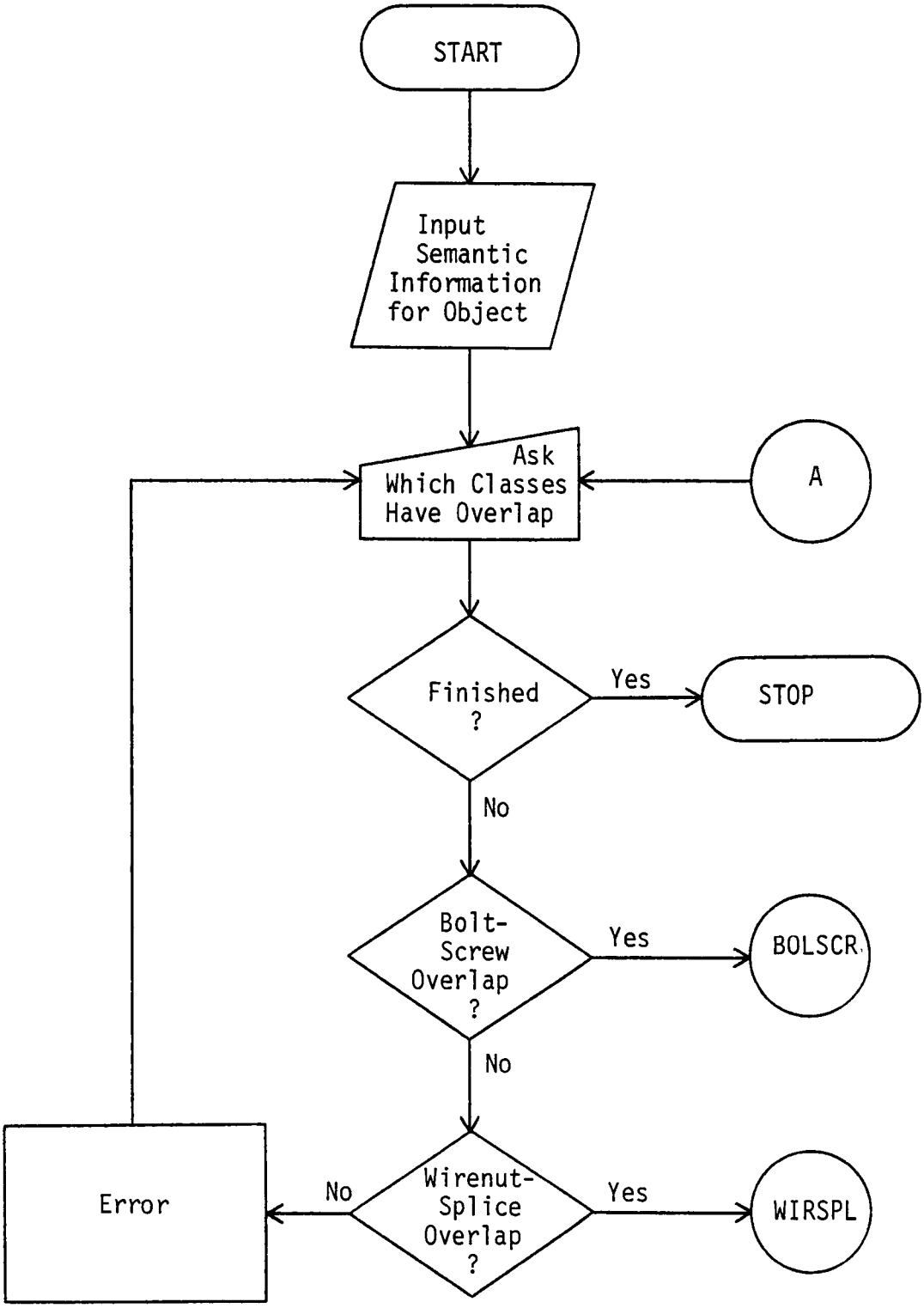


Figure 20. Simplified Flowchart of Program SEMAN.

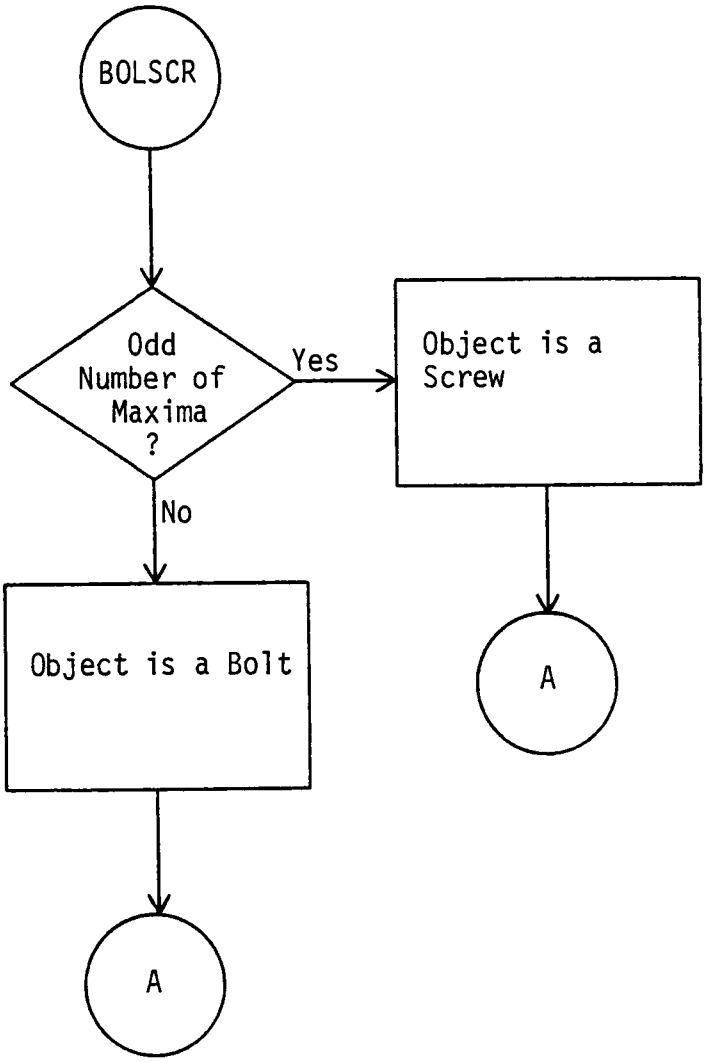


Figure 20. (Continued)

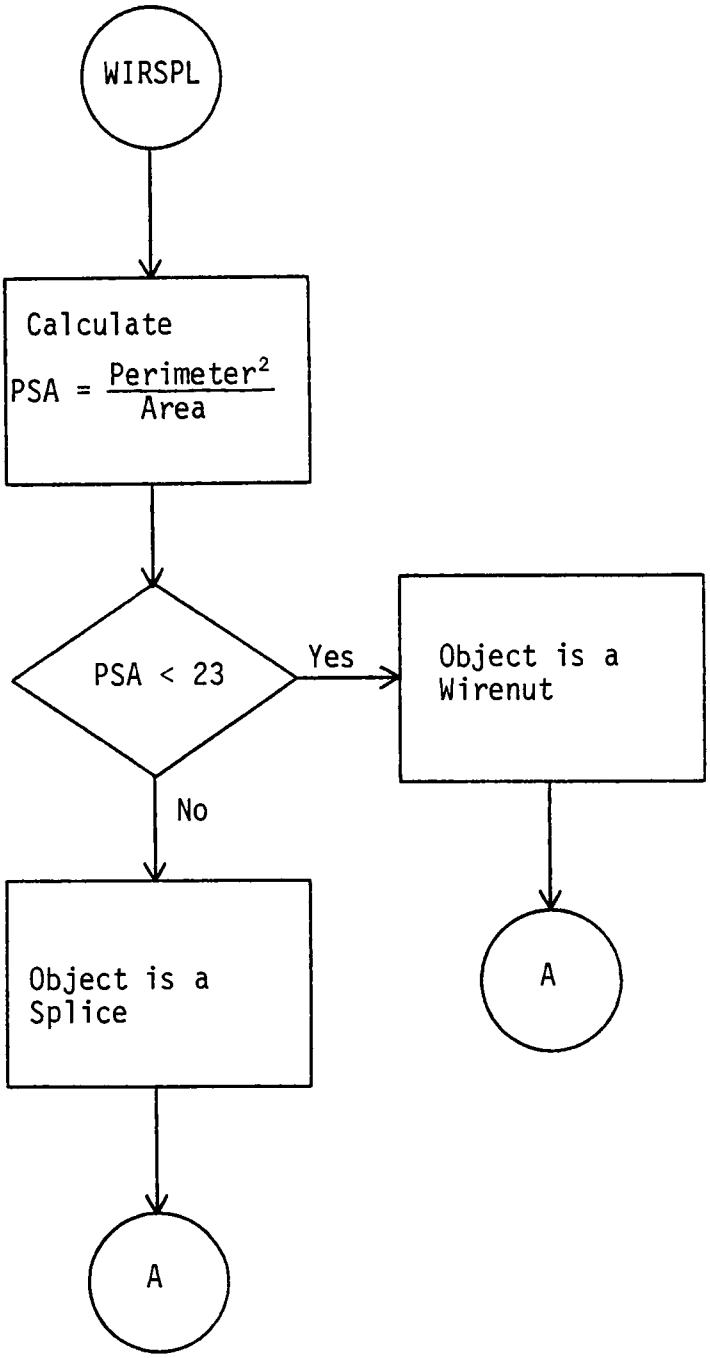


Figure 20. (Continued)

APPENDIX B

FORTRAN PROGRAMS

STRING

```
      PROGRAM STRING
C
C PROGRAM NAME - STRING
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 26, 1982
C PURPOSE      - SEGMENT THE BOUNDARY OF AN INPUT IMAGE AND
C              - DERIVE THE STRING OF ANGLES BETWEEN THE
C              - SEGMENTS (256 X 256 IMAGE) AND SEMANTIC
C              - INFORMATION REGARDING THE IMAGE
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK, FNAME, ANSWER, STRNAM(36)
      LOGICAL*1 SEMNAM(36)
      DIMENSION WRK(512), IWRK(256)
      COMMON BUF(1280), SEGCHN(251), ANGCHN(251), FNAME(36)
      EQUIVALENCE(WRK(1), BUF(1)), (WRK(1), IWRK(1))
C
      DATA ITHR/128/
C
C OPEN PICTURE FILE FOR UNFORMATTED DIRECT ACCESS WITH BLOCK
C SIZE = 512 BYTES
      TYPE 210
      CALL GETFNM('IMAGE FILE FOR INPUT?', 'PIC', FNAME,
     $            NCHAR)
      OPEN(UNIT=1, NAME=FNAME, TYPE='OLD', ACCESS='DIRECT',
     $     FORM='UNFORMATTED', RECORDSIZE=128)
C
      TYPE 200
200   FORMAT('0THE ANGLE STRING OUTPUT CAN EITHER BE',
     $       ' APPENDED TO ANGCHN.DAT FOR LATER USE IN', /,
     $       ' CONSTRUCTING THE AUTOMATON OR WRITTEN TO',
     $       ' ANOTHER FILE FOR LATER USE AS', /,
     $       ' INPUT TO THE AUTOMATON', //,
     $       ' DO YOU WANT THE ANGLES OUTPUT TO ANGCHN.DAT?'
     $       , //, ' ANSWER YES OR NO')
      ACCEPT 210, ANSWER
210   FORMAT(1A1)
      IF(ANSWER .EQ. 'Y') GO TO 220
      TYPE 210
      CALL GETFNM('OUTPUT FILE FOR ANGLE STRING?', 'DAT',
     $            STRNAM, NCHAR)
C
C GET THE NAME FOR THE OUTPUT FILE FOR THE SEGMENTED IMAGE
220   TYPE 210
```

```
      CALL GETFNM('OUTPUT FILE FOR SEGMENTED IMAGE?', 'SEG',
     $            FNAME, NCHAR)
C
C GET THE NAME FOR THE SEMANTIC INFORMATION OUTPUT FILE
      TYPE 210
      CALL GETFNM('OUTPUT FILE FOR SEMANTIC INFORMATION?',
     $            'SEM', SEMNAM, NCHAR)
C
C ASK FOR VECTOR LENGTH
10    TYPE 15
15    FORMAT('0WHAT VECTOR LENGTH DO YOU WANT?',/,
     $       ' ENTER IN I2 FORMAT')
      ACCEPT 20, LENGTH
20    FORMAT(I2)
C
C GET HEADER INFORMATION
      CALL FFICRD(1, WRK, 1, 1)
      CALL RDHDR(WRK,,,IDSTBK,NDBLK,,,,,IFLAG)
C
C COORDINATES OF FIRST OBJECT PIXEL, OBJECT AREA, AND
C CENTROID ARE STORED IN HEADER IF PICTURE HAS
C ALREADY BEEN PROCESSED
      INICOL = IWRK(13)
      INIROW = IWRK(14)
      AREA = IWRK(15)
      XCNTR = IWRK(16)
      YCNTR = IWRK(17)
C
C IF IMAGE HAS ALREADY BEEN PROCESSED, SKIP TO SEGMENTATION
C (WE HAVE THE INITIAL OBJECT PIXEL COORDINATES, THE AREA,
C AND THE CENTROID)
      IF(IFLAG .EQ. 1) GO TO 1000
C
C OBTAIN COORDINATES OF INITIAL OBJECT (BLACK) PIXEL, OBJECT
C AREA, AND CENTROID
      TYPE 1
1     FORMAT('0FIND INITIAL OBJECT PIXEL, OBJECT AREA, AND',
     $       ' CENTROID')
      CALL INIPIX(ITHR, IDSTBK, NDBLK, INICOL, INIROW, AREA,
     $            XCNTR, YCNTR)
1000  CONTINUE
C
C SEGMENT EDGES INTO FREEMAN CHAIN CODE AND COUNT THE
C MAXIMUM BOUNDARY POINTS AND PERIMETER
      TYPE 2
2     FORMAT('0BEGIN SEGMENTATION')
      CALL SEGMNT(ITHR, IDSTBK, NDBLK, INICOL, INIROW,
     $            LENGTH, PERIM, XCNTR, YCNTR, MAXIMA)
C
C IF WE HAVE EXCEEDED 250 SEGMENTS, GIVE HIM ANOTHER CHANCE
      IF(SEGCHN(1) .GT. 250) GO TO 10
C
```

```
C CLOSE THE PICTURE FILE
      CLOSE(UNIT=1, DISPOSE='SAVE')
C
C CREATE A FILE 'FNAME.PIC' CONTAINING THE SEGMENTED
C BOUNDARY IMAGE
      TYPE 4
4     FORMAT ('0WRITE SEGMENTED IMAGE TO DISK')
      CALL SEGPRT(INICOL, INIROW, LENGTH, IDSTBK, NDBLK)
C
C DERIVE THE STRING OF ANGLES BETWEEN THE EDGE VECTORS
      TYPE 5
5     FORMAT('0DERIVE ANGLE STRING')
      CALL ANGLE
C
C OPEN THE CORRECT OUTPUT FILE FOR THE ANGLE STRING
      IF(ANSWER .EQ. 'Y') GO TO 70
      OPEN(UNIT=4, NAME=STRNAM, TYPE='NEW', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
      GO TO 80
C
70    OPEN(UNIT=4, NAME='ANGCHN.DAT', TYPE='UNKNOWN',
     $     ACCESS='APPEND', FORM='FORMATTED')
C
C WRITE OUT THE ANGLE STRING TO DISK
80    TYPE 6
6     FORMAT('0WRITE ANGLES TO DISK')
      WRITE(4,90) ANGCHN(1)
90    FORMAT(1X, I3)
      WRITE(4,100) (ANGCHN(I), I = 2, ANGCHN(1)+1)
100   FORMAT(10(1X, I1))
C
      CLOSE(UNIT=4, DISPOSE='SAVE')
C
C WRITE OUT THE SEMANTIC INFORMATION FILE
      TYPE 7
7     FORMAT('0WRITE SEMANTIC INFORMATION TO DISK')
      OPEN(UNIT=4, NAME=SEMNAM, TYPE='NEW', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
      WRITE(4,110) AREA, PERIM, XCNTR, YCNTR, MAXIMA
110   FORMAT(1X, I5)
      CLOSE(UNIT=4, DISPOSE='SAVE')
C
      STOP
      END
```

```
      SUBROUTINE INIPIX(ITHR, IDSTBK, NDBLK, INICOL, INIROW,
     $                  AREA, XCNTR, YCNTR)
C
C PROGRAM NAME - INIPIX
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 26, 1982
C PURPOSE      - OBTAIN COORDINATES OF INITIAL OBJECT
C              - (BLACK) PIXEL (256 X 256 IMAGE)
C              - ALSO, OBTAIN AREA (PIXEL COUNT) OF THE
C              - OBJECT AND ITS CENTROID
C
      IMPLICIT INTEGER (A-Z)
      REAL RXCNTR, RYCNTR, RSUM
      LOGICAL*1 BUF, WRK
      DIMENSION WRK(512), IWRK(256)
      COMMON BUF(1280), SEGCHN(251), ANGCHN(251)
      EQUIVALENCE(WRK(1), BUF(1)), (WRK(1), IWRK(1))
      DATA RXCNTR, RYCNTR /2*0.0/
C
C READ IN IMAGE ONE ROW AT A TIME
      DO 20 I = IDSTBK, IDSTBK+NDBLK-1
      CALL FPICRD(1, WRK, I, 1)
C
C SEARCH FOR THE INITIAL OBJECT PIXEL
      DO 10 J = 1, 512
      IF(INTPX(WRK(J)) .GE. ITHR) GO TO 10
C
C THIS IS THE INITIAL OBJECT (BLACK) PIXEL, OBTAIN ITS
C COORDINATES
      IMROW = I
      INICOL = J
      INIROW = (I - IDSTBK) * 2 + 1
      IF(J .LE. 256) GO TO 30
      INICOL = INICOL - 256
      INIROW = INIROW + 1
      GO TO 30
C
10    CONTINUE
20    CONTINUE
C
C OBTAIN THE AREA (PIXEL COUNT) OF THE IMAGE AND THE
C CENTROID
30    AREA = 0
C
      DO 50 I = IMROW, IDSTBK+NDBLK-1
      OLDAR = AREA
      CALL FPICRD(1, WRK, I, 1)
C
C ODD-NUMBERED ROW
      RSUM = 0.0
      DO 40 J = 1, 256
      IF(INTPX(WRK(J)) .GE. ITHR)  GO TO 40
```

```
      AREA = AREA + 1
      RSUM = RSUM + 1.0
      RYCNTR = RYCNTR + FLOAT(J)
40    CONTINUE
      RXCNTR = RXCNTR + RSUM * FLOAT((I - IDSTBK) * 2 + 1)
C
C EVEN-NUMBERED ROW
      RSUM = 0.0
      DO 45 J = 257, 512
      JJ = J - 256
      IF(INTFX(WRK(J)) .GE. ITHR) GO TO 45
      AREA = AREA + 1
      RSUM = RSUM + 1.0
      RYCNTR = RYCNTR + FLOAT(JJ)
45    CONTINUE
      RXCNTR = RXCNTR + RSUM * FLOAT((I - IDSTBK) * 2 + 2)
C
      IF(AREA .EQ. OLDAR) GO TO 60
50    CONTINUE
C
C CALCULATE THE CENTROID
60    XCNTR = IFIX(RXCNTR / FLOAT(AREA))
      YCNTR = IFIX(RYCNTR / FLOAT(AREA))
C
C FLAG THIS PICTURE FILE AS HAVING BEEN PROCESSED
C STORE COORDINATES OF INITIAL OBJECT PIXEL, AREA, AND
C CENTROID IN PICTURE HEADER
      CALL FPICRD(1, WRK, 1, 1)
      IWRK(13) = INICOL
      IWRK(14) = INIROW
      IWRK(15) = AREA
      IWRK(16) = XCNTR
      IWRK(17) = YCNTR
      WRK(511) = 1
      CALL FPICWT(1, WRK, 1, 1)
C
      RETURN
      END
```

```
      SUBROUTINE SEGMNT(ITHR, IDSTBK, NDBLK, INICOL, INIROW,
     $                  LENGTH, PERIM, XCNTR, YCNTR, MAXIMA)
C
C PROGRAM NAME - SEGMNT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 26, 1982
C PURPOSE      - SEGMENT OBJECT EDGES INTO FREEMAN CHAIN
C              - CODE (256 X 256 IMAGE)
C              - ALSO, OBTAIN COUNT OF THE PERIMETER PIXELS
C              - AND COUNT BOUNDARY MAXIMA POINTS
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF1, BUF2, IMAGE, EVEN, ODD
      DIMENSION EVEN(1024), ODD(1024)
      COMMON BUF1(256), IMAGE(256,3), BUF2(256), SEGCHN(251)
      EQUIVALENCE(EVEN(1), IMAGE(1,1)), (ODD(1), BUF1(1))
      DATA INCCNT, DECCNT, INCFLG, DECFLG, COUNTR /5*0/
      DATA INIFLG /0/
C
      LSTCOL = INICOL - 1
      LSTROW = INIROW - 1
      NOWCOL = INICOL
      NOWROW = INIROW
      LNKCOL = INICOL
      LNKROW = INIROW
      SEGCHN(1) = 0
      PERIM = 0
      MAXIMA = 0
C
C CALCULATE DISTANCE FROM CENTROID TO INITIAL BOUNDARY PIXEL
      DISTSQ = ((XCNTR - NOWROW) ** 2) +
     $         ((YCNTR - NOWCOL) ** 2)
C
C READ IN THE ROW CONTAINING THE INITIAL PIXEL, THE ONE
C BEFORE IT, AND THE ONE AFTER IT
      BUFROW = (NOWROW - 2) / 2 + IDSTBK
      IF(MOD(NOWROW,2) .EQ. 0) GO TO 10
C
C FIRST ROW IS ODD-NUMBERED
      CALL FPICRD(1, ODD, BUFROW, 2)
      GO TO 50
C
C FIRST ROW IS EVEN-NUMBERED
10    CALL FPICRD(1, EVEN, BUFROW, 2)
      GO TO 50
C
20    CONTINUE
      BUFROW = (NOWROW - 2) / 2 + IDSTBK
C
C IF WE ARE STILL IN THE SAME ROW, NO NEED TO READ IN NEW
C ROWS
      IF(NOWROW - LSTROW) 30, 50, 30
```

```
30    IF(MOD(NOWROW, 2) .EQ. 0) GO TO 40
C
C CURRENT ROW IS ODD-NUMBERED
      CALL FPICRD(1, ODD, BUFROW, 2)
      GO TO 50
C
C CURRENT ROW IS EVEN-NEMBERED
40    CALL FPICRD(1, EVEN, BUFROW, 2)
C
50    CONTINUE
C
C FIND NEXT EDGE PIXEL AND INCREMENT THE PERIMETER PIXEL
C COUNT
      CALL NEXT(ITHR, NOWCOL, NOWROW, LSTCOL, LSTROW)
      PERIM = PERIM + 1
C
C CALCULATE DISTANCE FROM CENTROID TO CURRENT BOUNDARY PIXEL
      DISOLD = DISTSQ
      DISTSQ = ((XCNTR - NOWROW) ** 2) +
     $         ((YCNTR - NOWCOL) ** 2)
C
C HAS DISTANCE FROM THE CENTROID INCREASED OR DECREASED?
      IF(DISTSQ - DISOLD) 70, 80, 60
C
C DISTANCE HAS INCREASED
60    INCCNT = INCCNT + 1
      INCFLG = 0
      IF(INCCNT .GE. 2) INCFLG = 1
      IF(INCCNT .GE. 2) DECCNT = 0
      GO TO 80
C
C DISTANCE HAS DECREASED
70    DECCNT = DECCNT + 1
      DECFLG = 0
      IF(DECCNT .GE. 2) DECFLG = 1
      IF(DECCNT .GE. 2) INCCNT = 0
C
C IF DISTANCED HAS INCREASED FOR TWO PIXELS, THEN DECREASED
C FOR TWO PIXELS, COUNT A MAXIMUM POINT
      IF((INCFLG .NE. 1) .OR. (DECFLG .NE. 1)) GO TO 80
      MAXIMA = MAXIMA + 1
      INCFLG = 0
      DECFLG = 0
      GO TO 80
C
C WERE THE FIRST TWO DISTANCES DECREASING?
80    COUNTR = COUNTR + 1
      IF(COUNTR .GT. 2) GO TO 90
      IF(DECCNT .NE. 0) INIFLG = INIFLG + 1
C
C CAN WE CONNECT TO THE NEW EDGE PIXEL WITH A VECTOR?
90    CALL CONECT(LENGTH, LNKCOL, LNKROW, NOWCOL, NOWROW,
```

```
     $              NEWVEC)
      IF(SEGCHN(1) .GT. 250) GO TO 99999
C
C IF THE NEW VECTOR CONNECTS TO THE STARTING PIXEL, THE
C SEGMENTED BOUNDARY
C IS FINISHED
      IF(NEWVEC .NE. 1) GO TO 20
      IF((LNKCOL .NE. INICOL) .OR. (LNKROW .NE. INIROW))
     $  GO TO 20
C
C FINISH COUNTING THE PERIMETER PIXELS
100   IF((NOWROW .EQ. INIROW) .AND. (NOWCOL .EQ. INICOL))
     $  GO TO 130
      CALL NEXT(ITHR, NOWCOL, NOWROW, LSTCOL, LSTROW)
      PERIM = PERIM + 1
C
C CALCULATE DISTANCE FROM CENTROID TO CURRENT BOUNDARY PIXEL
      DISOLD = DISTSQ
      DISTSQ = ((XCNTR - NOWROW) ** 2) +
     $         ((YCNTR - NOWCOL) ** 2)
C
C HAS DISTANCE FROM THE CENTROID INCREASED OR DECREASED?
      IF(DISTSQ - DISOLD) 120, 100, 110
C
C DISTANCE HAS INCREASED
110   INCCNT = INCCNT + 1
      INCFLG = 0
      IF(INCCNT .GE. 2) INCFLG = 1
      IF(INCCNT .GE. 2) DECCNT = 0
      GO TO 100
C
C DISTANCE HAS DECREASED
120   DECCNT = DECCNT + 1
      DECFLG = 0
      IF(DECCNT .GE. 2) DECFLG = 1
      IF(DECCNT .GE. 2) INCCNT = 0
C
C IF DISTANCE HAS INCREASED FOR TWO PIXELS, THEN DECREASED
C FOR TWO PIXELS, COUNT A MAXIMUM POINT
      IF((INCFLG .NE. 1) .OR. (DECFLG .NE. 1)) GO TO 100
      MAXIMA = MAXIMA + 1
      INCFLG = 0
      DECFLG = 0
      GO TO 100
C
C IF A MAXIMUM POINT WAS NEAR THE INITIAL PIXEL, MAKE SURE
C IT WAS FOUND
130   IF(((INCFLG .EQ. 1) .AND. (DECFLG .NE. 1)) .AND.
     $   ((DECCNT .EQ. 1) .OR. (INIFLG .EQ. 2)))
     $  MAXIMA = MAXIMA + 1
C
99999 RETURN
      END
```

```
      SUBROUTINE CONECT(LENGTH, LNKCOL, LNKROW, NOWCOL,
     $                  NOWROW, NEWVEC)
C
C PROGRAM NAME - CONECT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 26, 1982
C PURPOSE      - CONNECT PRESENT PIXEL TO PREVIOUS VECTOR
C              - TIP WITH A NEW VECTOR, IF POSSIBLE
C              - (256 X 256 IMAGE)
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK
      DIMENSION WRK(512)
      COMMON BUF(1280), SEGCHN(251), ANGCHN(251)
      EQUIVALENCE(WRK(1), BUF(1))
C
      LHALF = LENGTH / 2
      LCOL = LNKCOL
      LROW = LNKROW
      NEWVEC = 0
C
C CALCULATE COLUMN AND ROW INCREMENTS TO THIS PIXEL FROM
C LAST VECTOR TIP
      INCCOL = NOWCOL - LNKCOL
      INCROW = NOWROW - LNKROW
C
C ARE WE AT VECTOR RADIUS FROM LAST TIP?
      IF(IABS(INCCOL) .EQ. LENGTH) GO TO 10
      IF(IABS(INCROW) .EQ. LENGTH) GO TO 20
      GO TO 99999
C
C FIND BEST APPROXIMATION VECTOR - PIXEL IS IN A BOUNDARY
C COLUMN
10    LNKCOL = NOWCOL
      IF(INCROW .LT. -LHALF) LNKROW = LNKROW - LENGTH
      IF(INCROW .GT. LHALF) LNKROW = LNKROW + LENGTH
      GO TO 30
C
C FIND BEST APPROXIMATION VECTOR - PIXEL IS IN A BOUNDARY
C ROW
20    LNKROW = NOWROW
      IF(INCCOL .LT. -LHALF) LNKCOL = LNKCOL - LENGTH
      IF(INCCOL .GT. LHALF) LNKCOL = LNKCOL + LENGTH
C
C WE HAVE TIP OF NEW VECTOR, FIND VECTOR NUMBER
30    JCOL = LNKCOL - LCOL
      JROW = LNKROW - LROW
      SEGCHN(1) = SEGCHN(1) + 1
C
C WE ARE LIMITED TO 250 SEGMENTS
      IF(SEGCHN(1) .LE. 250) GO TO 50
      TYPE 40
```

```
40     FORMAT(' SEGMENTED IMAGE HAS EXCEEDED 250 SEGMENTS',
     $        /,
     $        ' TRY AGAIN WITH A LONGER SEGMENTATION VECTOR')
       GO TO 99999
C
50     LOC = SEGCHN(1) + 1
       IF((JCOL .EQ. LENGTH ) .AND. (JROW .EQ. 0       ))
     $   SEGCHN(LOC) = 0
       IF((JCOL .EQ. LENGTH ) .AND. (JROW .EQ. -LENGTH))
     $   SEGCHN(LOC) = 1
       IF((JCOL .EQ. 0      ) .AND. (JROW .EQ. -LENGTH))
     $   SEGCHN(LOC) = 2
       IF((JCOL .EQ. -LENGTH) .AND. (JROW .EQ. -LENGTH))
     $   SEGCHN(LOC) = 3
       IF((JCOL .EQ. -LENGTH) .AND. (JROW .EQ. 0       ))
     $   SEGCHN(LOC) = 4
       IF((JCOL .EQ. -LENGTH) .AND. (JROW .EQ. LENGTH ))
     $   SEGCHN(LOC) = 5
       IF((JCOL .EQ. 0      ) .AND. (JROW .EQ. LENGTH ))
     $   SEGCHN(LOC) = 6
       IF((JCOL .EQ. LENGTH ) .AND. (JROW .EQ. LENGTH ))
     $   SEGCHN(LOC) = 7
C
C WE HAVE A NEW VECTOR
       NEWVEC = 1
C
99999 RETURN
       END
```

```
      SUBROUTINE NEXT(ITHR, NOWCOL, NOWROW, LSTCOL, LSTROW)
C
C PROGRAM NAME - NEXT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 26, 1982
C PURPOSE      - FIND NEXT EDGE PIXEL BY FOLLOWING THE
C              - OBJECT BORDER IN A CLOCKWISE DIRECTION
C              - (256 X 256 IMAGE)
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF1, BUF2, IMAGE, EVEN, ODD
      DIMENSION EVEN(1024), ODD(1024)
      COMMON BUF1(256), IMAGE(256,3), BUF2(256), SEGCHN(251)
      EQUIVALENCE(EVEN(1), IMAGE(1,1)), (ODD(1), BUF1(1))
C
      IFLAG = 0
C
C DETERMINE AT WHICH PIXEL TO START SEARCH
      IVECT = (NOWCOL - LSTCOL) + 3 * (NOWROW - LSTROW) + 5
C
C THE LAST PIXEL WILL BE WHAT IS NOW THE PRESENT PIXEL
      LSTCOL = NOWCOL
      LSTROW = NOWROW
      GO TO (10, 30, 30, 10, 99999, 50, 70, 70, 50), IVECT
C
C SEARCH SURROUNDING PIXELS FOR THE NEXT EDGE PIXEL
10    IF(INTPX(IMAGE(LSTCOL-1,3)) .GE. ITHR) GO TO 20
      NOWCOL = LSTCOL - 1
      NOWROW = LSTROW + 1
      GO TO 99999
C
20    IF(INTPX(IMAGE(LSTCOL-1,2)) .GE. ITHR) GO TO 30
      NOWCOL = LSTCOL - 1
      NOWROW = LSTROW
      GO TO 99999
C
30    IF(INTPX(IMAGE(LSTCOL-1,1)) .GE. ITHR) GO TO 40
      NOWCOL = LSTCOL - 1
      NOWROW = LSTROW - 1
      GO TO 99999
C
40    IF(INTPX(IMAGE(LSTCOL,1)) .GE. ITHR) GO TO 50
      NOWCOL = LSTCOL
      NOWROW = LSTROW - 1
      GO TO 99999
C
C CHECK FOR ISOLATED INITIAL PIXEL
50    IFLAG = IFLAG + 1
      IF(IFLAG .GT. 1) GO TO 90
      IF(INTPX(IMAGE(LSTCOL+1,1)) .GE. ITHR) GO TO 60
      NOWCOL = LSTCOL + 1
      NOWROW = LSTROW - 1
```

```
      GO TO 99999
C
60    IF(INTFX(IMAGE(LSTCOL+1,2)) .GE. ITHR) GO TO 70
      NOWCOL = LSTCOL + 1
      NOWROW = LSTROW
      GO TO 99999
C
70    IF(INTFX(IMAGE(LSTCOL+1,3)) .GE. ITHR) GO TO 80
      NOWCOL = LSTCOL + 1
      NOWROW = LSTROW + 1
      GO TO 99999
C
80    IF(INTFX(IMAGE(LSTCOL,3)) .GE. ITHR) GO TO 10
      NOWCOL = LSTCOL
      NOWROW = LSTROW + 1
      GO TO 99999
C
C FOUND ISOLATED INITIAL PIXEL
90    TYPE 100, LSTCOL, LSTROW
100   FORMAT(' INITIAL PIXEL IS AN ISOLATED PIXEL AT',
     $       ' COLUMN ', I3, ', ROW ', I3)
      STOP
C
99999 RETURN
      END
```

```
      SUBROUTINE SEGPRT(INICOL, INIROW, LENGTH, IDSTBK,
     $                  NDBLK)
C
C PROGRAM NAME - SEGPRT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 27, 1982
C PURPOSE      - CREATE A FILE 'SEGPRT.PIC' CONTAINING THE
C              - SEGMENTED BOUNDARY IMAGE (256 X 256 IMAGE)
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK, WRK2, FNAME
      DIMENSION WRK(512), WRK2(512)
      COMMON BUF(1280), SEGCHN(251), ANGCHN(251), FNAME(36)
      EQUIVALENCE(WRK(1), BUF(513)), (WRK2(1), BUF(257))
C
      LNKCOL = INICOL
      LNKROW = INIROW
C
C OPEN FILE OUTPUT FILE IN WHICH THE SEGMENTED BOUNDARY
C IMAGE WILL BE STORED
      OPEN(UNIT=3, NAME=FNAME, TYPE='NEW', ACCESS='DIRECT',
     $     FORM='UNFORMATTED', RECORDSIZE=128)
      CALL NEWHDR(WRK,,'M',IDSTBK,NDBLK,FNAME)
      WRITE(3'1) WRK
C
C CLEAR THE NEW IMAGE
      DO 10 J = 1, 512
      WRK(J) = 255
      WRK2(J) = 255
10    CONTINUE
      DO 20 I = IDSTBK, IDSTBK+NDBLK-1
      WRITE(3'I) WRK
20    CONTINUE
C
C SET UP FOR FIRST ROW
      IOLD = INIROW
C
C BLACK OUT EACH VECTOR
      DO 190 ISEG = 2, SEGCHN(1)+1
      IVECT = SEGCHN(ISEG) + 1
      GO TO (30, 50, 70, 90, 110, 130, 150, 170), IVECT
C
C VECTOR = 0
30    CALL IMREAD(LNKROW, IOLD, IDSTBK)
      DO 40 K = 1, LENGTH
      WRK(LNKCOL+K-1) = 0
40    CONTINUE
      CALL IMWRIT(LNKROW, IDSTBK)
      LNKCOL = LNKCOL + LENGTH
      GO TO 190
C
C VECTOR = 1
```

```
50      DO 60 K = 1, LENGTH
        IROW = LNKROW - K + 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL+K-1) = 0
        CALL IMWRIT(IROW, IDSTBK)
60      CONTINUE
        LNKCOL = LNKCOL + LENGTH
        LNKROW = LNKROW - LENGTH
        GO TO 190
C
C VECTOR = 2
70      DO 80 K = 1, LENGTH
        IROW = LNKROW - K + 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL) = 0
        CALL IMWRIT(IROW, IDSTBK)
80      CONTINUE
        LNKROW = LNKROW - LENGTH
        GO TO 190
C
C VECTOR = 3
90      DO 100 K = 1, LENGTH
        IROW = LNKROW - K + 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL-K+1) = 0
        CALL IMWRIT(IROW, IDSTBK)
100     CONTINUE
        LNKCOL = LNKCOL - LENGTH
        LNKROW = LNKROW - LENGTH
        GO TO 190
C
C VECTOR = 4
110     CALL IMREAD(LNKROW, IOLD, IDSTBK)
        DO 120 K = 1, LENGTH
        WRK(LNKCOL-K+1) = 0
120     CONTINUE
        CALL IMWRIT(LNKROW, IDSTBK)
        LNKCOL = LNKCOL - LENGTH
        GO TO 190
C
C VECTOR = 5
130     DO 140 K = 1, LENGTH
        IROW = LNKROW + K - 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL-K+1) = 0
        CALL IMWRIT(IROW, IDSTBK)
140     CONTINUE
        LNKCOL = LNKCOL - LENGTH
        LNKROW = LNKROW + LENGTH
        GO TO 190
C
C VECTOR = 6
```

```
150     DO 160 K = 1, LENGTH
        IROW = LNKROW + K - 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL) = 0
        CALL IMWRIT(IROW, IDSTBK)
160     CONTINUE
        LNKROW = LNKROW + LENGTH
        GO TO 190
C
C VECTOR = 7
170     DO 180 K = 1, LENGTH
        IROW = LNKROW + K - 1
        CALL IMREAD(IROW, IOLD, IDSTBK)
        WRK(LNKCOL+K-1) = 0
        CALL IMWRIT(IROW, IDSTBK)
180     CONTINUE
        LNKCOL = LNKCOL + LENGTH
        LNKROW = LNKROW + LENGTH
C
190     CONTINUE
C
C CLOSE THE SEGMENTED BOUNDARY IMAGE FILE
        CLOSE(UNIT=3, DISPOSE='SAVE')
C
        RETURN
        END
```

```
      SUBROUTINE IMREAD(IROW, IOLD, IDSTBK)
C
C PROGRAM NAME - IMREAD
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - JUNE 5, 1983
C PURPOSE      - READ A ROW OF A 256 X 256 IMAGE INTO THE
C              - ARRAY WRK
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK, WRK2
      DIMENSION WRK(512), WRK2(512)
      COMMON BUF(1280)
      EQUIVALENCE(WRK(1), BUF(513)), (WRK2(1), BUF(257))
C
C SEE WHERE CURRENT ROW IS IN RELATION TO LAST ROW
      IF(IROW - IOLD) 10, 99999, 40
C
C CURRENT ROW IS BEFORE LAST ROW
10    IF(MOD(IROW, 2) .EQ. 1) GO TO 20
C
C CURRENT ROW IS EVEN-NUMBERED
      CALL FPICRD(3, WRK2, (IROW/2 - 1 + IDSTBK), 1)
      GO TO 70
C
C CURRENT ROW IS ODD-NUMBERED
20    DO 30 I = 1, 512
      IBACK = 512 - I + 1
      WRK(IBACK) = WRK2(IBACK)
30    CONTINUE
      GO TO 70
C
C CURRENT ROW IS AFTER LAST ROW
40    IF(MOD(IROW, 2) .EQ. 1) GO TO 60
C
C CURRENT ROW IS EVEN-NUMBERED
      DO 50 I = 1, 512
      WRK2(I) = WRK(I)
50    CONTINUE
      GO TO 70
C
C CURRENT ROW IS ODD-NUMBERED
60    CALL FPICRD(3, WRK, (IROW/2 + IDSTBK), 1)
C
C SET IOLD TO CURRENT ROW
70    IOLD = IROW
C
99999 RETURN
      END
```

```
      SUBROUTINE IMWRIT(IROW, IDSTBK)
C
C PROGRAM NAME - IMWRIT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - JUNE 5, 1983
C PURPOSE      - WRITE A ROW TO A 256 X 256 IMAGE
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK, WRK2
      DIMENSION WRK(512), WRK2(512)
      COMMON BUF(1280)
      EQUIVALENCE(WRK(1), BUF(513)), (WRK2(1), BUF(257))
C
      IF(MOD(IROW, 2) .EQ. 1) GO TO 10
C
C EVEN-NUMBERED ROW
      CALL FPICWT(3, WRK2, (IROW/2 - 1 + IDSTBK), 1)
      GO TO 99999
C
C ODD-NUMBERED ROW
10    CALL FPICWT(3, WRK, (IROW/2 + IDSTBK), 1)
C
99999 RETURN
      END
```

```
      SUBROUTINE ANGLE
C
C PROGRAM NAME - ANGLE
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - NOVEMBER 27, 1982
C PURPOSE      - DERIVE THE STRING OF ANGLES BETWEEN THE
C              - EDGE VECTORS, STORE THIS STRING IN ARRAY
C              - ANGCHN (256 X 256 IMAGE)
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 BUF, WRK
      DIMENSION WRK(512)
      COMMON BUF(1280), SEGCHN(251), ANGCHN(251)
      EQUIVALENCE(WRK(1), BUF(1))
C
C THE NUMBER OF ANGLES EQUALS THE NUMBER OF VECTORS
      ANGCHN(1) = SEGCHN(1)
C
C FIND THE ANGLE BETWEEN EACH PAIR OF VECTORS
      DO 10 I = 2, SEGCHN(1)
C
C WITH ANGLES MEASURED COUNTERCLOCKWISE, WE HAVE
C IANGLE = ANGLE / 45 DEGREES
      IANGLE = SEGCHN(I+1) - SEGCHN(I)
C
C ALSO, FOR ALPHA>0 AND BETA<0, WE HAVE
C ALPHA = 360 DEGREES + BETA
C THIS EQUALITY IN STRING ANGLES IS
C IANGLE = 8 + IANGLE
      IF(IANGLE .LT. 0) IANGLE = 8 + IANGLE
      ANGCHN(I) = IANGLE
10    CONTINUE
C
C TAKE CARE OF LAST VECTOR, WHICH CONNECTS TO THE FIRST ONE
      IANGLE = SEGCHN(2) - SEGCHN(SEGCHN(1)+1)
      IF(IANGLE .LT. 0) IANGLE = 8 + IANGLE
      ANGCHN(SEGCHN(1)+1) = IANGLE
C
      RETURN
      END
```

KTAIL

```
      PROGRAM KTAIL
C
C PROGRAM NAME - KTAIL
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - CREATE A DETERMINISTIC FINITE-STATE
C              - AUTOMATON FROM ONE OR MORE INPUT STRINGS
C              - USING THE K-TAIL METHOD OF GRAMMATICAL
C              - INFERENCE
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
C ARRAYS:
C Z(Z ,SYMBOL) - ARRAY OF Z STRINGS FORMED FROM THE INPUT
C                STRINGS
C KTAIL(Z ,SYMBOL) - ARRAY OF KTAIL STRINGS FORMED FROM THE
C                    INPUT STRINGS
C                    (THESE KTAIL STRINGS CORRESPOND TO THE
C                    ZSTRINGS IN ARRAY Z)
C Q(Z ,INDEX) - STATE INFORMATION ARRAY
C               Z   = 1  -  PHI STATE
C               Z   = 2  -  INITIAL STATE
C               INDEX = 1  -  LENGTH OF Z STRING
C               INDEX = 2  -  LENGTH OF KTAIL STRING
C               INDEX = 3  -  ZSTRING NUMBER AFTER ZSTRINGS
C                             HAVE BEEN EQUIVALENCED
C               INDEX = 4  -  KTAIL NUMBER AFTER KTAILS HAVE
C                             BEEN EQUIVALENCED
C               INDEX = 5  -  STATE NUMBER AFTER STATES HAVE
C                             BEEN EQUIVALENCED
C               INDEX = 6  -  = 1 IF Z STRING IS THE FIRST
C                                 (SMALLEST) Z STRING
C                                 FORMED FROM THE INPUT
C                                 STRING
C ZKTAIL(Z ,KTAIL ) -  1 <= KTAIL  <= 10  -  LISTING OF
C                                            KTAILS
C                                            ASSOCIATED WITH
C                                            DISTINCT Z
C                                            STRINGS
C                      KTAIL  = 11  -  NUMBER OF KTAILS FOR
C                                      THIS Z STRING
C                      KTAIL  = 12  -  = 1 IF THIS IS A
C                                          FINAL STATE
C
C FORM THE Z STRINGS AND KTAILS FROM THE INPUT STRINGS
      TYPE 10
10    FORMAT(//,' FORM THE Z STRINGS AND KTAILS FROM THE',
```

```
     $       ' INPUT STRINGS', /)
      CALL ZSTRNG
C
C FIND EQUIVALENT KTAILS
      TYPE 20
20    FORMAT(' FIND EQUIVALENT KTAILS',/)
      CALL KEQUIV
C
C FIND EQUIVALENT Z STRINGS
C BUILD LIST OF KTAILS ASSOCIATED WITH EACH DISTINCT Z
C STRING
      TYPE 30
30    FORMAT(' FIND EQUIVALENT Z STRINGS AND', /,
     $       ' BUILD LIST OF KTAILS ASSOCIATED WITH EACH Z',
     $       ' STRING',/)
      CALL ZEQUIV
C
C DETERMINE EQUIVALENT STATES BY COMPARING KTAIL LISTS
      TYPE 40
40    FORMAT(' DETERMINE EQUIVALENT STATES BY COMPARING',
     $       ' KTAIL LISTS',/)
      CALL STATEQ
C
C OBTAIN (NONDETERMINISTIC) TRANSITION TABLE DELTA FROM THE
C STATES
      TYPE 50
50    FORMAT(' OBTAIN NONDETERMINISTIC TRANSITION TABLE',/)
      CALL TRANSI
C
C OBTAIN THE DETERMINISTIC AUTOMATON FROM THE
C NONDETERMINISTIC TRANSITION TABLE
      TYPE 60
60    FORMAT(' OBTAIN THE DETERMINISTIC AUTOMATON',/)
      CALL DETERM
C
C ELIMINATE INACCESSIBLE STATES FROM THE DETERMINISTIC
C AUTOMATON
      TYPE 70
70    FORMAT(' ELIMINATE INACCESSIBLE STATES FROM THE',
     $       ' DETERMINISTIC AUTOMATON',/)
      CALL STELIM
C
C MINIMIZE THE DETERMINISTIC AUTOMATON BY COMBINING
C EQUIVALENT STATES
      TYPE 80
80    FORMAT(' MINIMIZE THE DETERMINISTIC AUTOMATON BY',
     $       ' COMBINING EQUIVALENT STATES',/)
      CALL MINIMA
C
C WRITE THE DETERMINISTIC AUTOMATON TO DISK
90    TYPE 100
100   FORMAT(' WRITE THE DETERMINISTIC AUTOMATON TO DISK',/)
```

```
      CALL STORE
C
      STOP
      END
```

```
      BLOCK DATA
C
C PROGRAM NAME - BLOCK DATA
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 8, 1983
C PURPOSE      - INITIALIZE COMMON
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
      DATA KTAIL/2500*0/, Z/2500*0/, Q/300*0/, ZKTAIL/600*0/
C
      END
```

```
      SUBROUTINE ZSTRNG
C
C PROGRAM NAME - ZSTRNG
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 8, 1983
C PURPOSE      - FORM Z STRINGS AND KTAILS FROM THE INPUT
C              - STRINGS
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 ANSWER
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      DIMENSION ANGCHN(250)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
      EQUIVALENCE (ANGCHN(1), ZKTAIL(1,1))
      DATA MAXWRD/50/, ZNUM/2/
C
C CHECK THAT INPUT FILE IS IN CORRECT FORMAT
      TYPE 10
10    FORMAT(' THE INPUT FILE OF ANGLE STRINGS SHOULD BE',
     $       ' NAMED ANGCHN.DAT',/,
     $       ' IS ANGCHN.DAT PREPARED CORRECTLY?',/,
     $       ' ENTER YES OR NO')
C
      ACCEPT 20,ANSWER
20    FORMAT(1A1)
      IF(ANSWER .EQ. 'Y') GO TO 40
C
C INPUT FILE NOT CORRECT
      TYPE 30
30    FORMAT(///,' CORRECT ANGCHN.DAT AND TRY AGAIN')
      STOP
C
C INPUT FILE IS CORRECT - OPEN THE INPUT FILE
40    OPEN(UNIT=4, NAME='ANGCHN.DAT', TYPE='OLD', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
C
C READ IN VALUE OF K
50    TYPE 60
60    FORMAT(///,' WHAT VALUE OF K DO YOU WANT?',/,
     $       ' ENTER IN I2 FORMAT')
      ACCEPT 70,K
70    FORMAT(I2)
      TYPE 75
75    FORMAT(///)
      IF((K .LE. 50) .AND. (K .GE. 0)) GO TO 90
      TYPE 80
80    FORMAT(' K MUST BE GREATER THAN OR EQUAL TO 0', /,
     $       ' AND LESS THAN OR EQUAL TO 50')
      GO TO 50
C
C READ IN ANGLE STRINGS
```

```
90     READ(4, 100, END=230) LENGTH
100    FORMAT(1X, I3)
       READ(4,110) (ANGCHN(I), I=1,LENGTH)
110    FORMAT(10(1X,I1))
C
C FORM K+1 Z'S AND KTAILS FROM EACH INPUT
       KN = LENGTH - K
C
C Z CAN NOT BE LESS THAN 0 SYMBOLS LONG
       IF(KN .LE. 0) KN = 0
       KINDEX = KN
       ZNUM = ZNUM + 1
       IF(ZNUM .GT. 50) GO TO 250
C
C FLAG THIS Z STRING AS BEING THE FIRST ONE FORMATTED FROM
C THIS INPUT STRING
       Q(ZNUM, 6) = 1
C
C TRUNCATE THE INPUT STRING TO FORM FIRST Z FOR THIS INPUT
       WORD = 0
130    CONTINUE
       I1 = (5 * WORD) + 1
       I2 = 5 * (WORD + 1)
C
C IF ALL KN SYMBOLS HAVE BEEN USED, THE FIRST Z FOR THIS
C INPUT HAS BEEN FORMED, SO GO FORM ITS KTAIL
       IF(KN .LT. I2) I2 = KN
       IF(KN .LT. I1) GO TO 180
       WORD = WORD + 1
C
C FORM THE Z STRING
       DO 140 I = I1, I2
       Z(ZNUM, WORD) = Z(ZNUM, WORD) +
      $                ANGCHN(I) * (8 ** (I - I1))
140    CONTINUE
C
C LOOP TO COMPLETE THE Z STRING
       GO TO 130
C
C FORM ADDITIONAL Z STRINGS USING THE PREVIOUS ONE AS A BASE
C (FOR K > 0)
150    DO 220 KINDEX = KN+1, LENGTH
       ZNUM = ZNUM + 1
       IF(ZNUM .GT. 50) GO TO 250
       DO 160 I = 1, WORD
       Z(ZNUM, I) = Z(ZNUM-1, I)
160    CONTINUE
C
C FIND THE LOCATION IN THE Z STRING AFTER THE LAST ONE USED
C ALSO, ALLOW FOR THE PREVIOUS Z STRING HAVING ZERO LENGTH
       POWER = MOD(I2-1, 5)
       IF((POWER .EQ. 0) .AND. (I2 .NE. 1)) WORD = WORD + 1
```

```
C
C APPEND THE NEXT INPUT SYMBOL
170   Z(ZNUM, WORD) = Z(ZNUM, WORD) +
     $                ANGCHN(I2) * (8 ** POWER)
C
C NOW FORM THE KTAIL FOR THIS Z STRING
C
C IF THE KTAIL IS EMPTY, NO NEED TO FORM IT
180   IF(KINDEX .EQ. LENGTH) GO TO 210
C
C FIND THE SYMBOL IN THE INPUT STRING AFTER THE LAST ONE
C USED
      I2 = I2 + 1
      SAVE = I2
      JWORD = 0
190   CONTINUE
      JWORD = JWORD + 1
      J1 = 5 * (JWORD - 1) + 1
      J2 = 5 * JWORD
      KLENTH = LENGTH - KINDEX
C
C IF ALL THE SYMBOLS IN THE INPUT STRING HAVE BEEN USED,
C THE KTAIL HAS BEEN FORMED
      IF(J1 .GT. KLENTH) GO TO 210
      IF(J2 .GT. KLENTH) J2 = KLENTH
      DO 200 J = J1, J2
      KTAIL(ZNUM, JWORD) = KTAIL(ZNUM, JWORD) +
     $                     ANGCHN(SAVE) * (8 ** (J - J1))
      SAVE = SAVE + 1
200   CONTINUE
C
C LOOP TO COMPLETE THE KTAIL
      GO TO 190
C
C STORE THE STATE INFORMATION
210   Q(ZNUM, 1) = KINDEX
      Q(ZNUM, 2) = LENGTH - KINDEX
      Q(ZNUM, 3) = ZNUM
      Q(ZNUM, 4) = ZNUM
      Q(ZNUM, 5) = ZNUM
C
C IS THIS AN INITIAL STATE?
      IF(KINDEX .EQ. 0) Q(ZNUM, 5) = 1
C
C IF K=0, THEN THIS IS THE ONLY Z FOR THIS INPUT
C GO BACK TO READ IN NEXT INPUT STRING
      IF(K .EQ. 0) GO TO 90
C
C IF THIS WAS THE FIRST Z FOR THIS INPUT, GO BACK AND FORM
C THE OTHERS (ENTER THE Z-BUILDING LOOP)
      IF(KINDEX .EQ. KN) GO TO 150
C
```

```
C LOOP BACK TO FORM NEXT Z FROM THIS INPUT STRING
220   CONTINUE
C
C GO BACK TO READ IN NEXT INPUT STRING
      GO TO 90
C
C ALL INPUT STRINGS HAVE BEEN PROCESSED
230   CONTINUE
      MAXSTA = ZNUM
C
C CLEAR THE ANGCHN ARRAY
      DO 240 I = 1, 250
      ANGCHN(I) = 0
240   CONTINUE
C
C STORE THE PHI STATE INFORMATION (ONLY USED IN DELBLD)
      Q(2,5) = 2
C
C CLOSE THE INPUT FILE
      CLOSE(UNIT=4, DISPOSE='SAVE')
      GO TO 99999
C
C EXCEEDED 50 Z STRINGS
250   TYPE 260
260   FORMAT(' EXCEEDED 50 Z STRINGS')
      STOP
C
99999 RETURN
      END
```

```
      SUBROUTINE KEQUIV
C
C PROGRAM NAME - KEQUIV
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 8, 1983
C PURPOSE      - FIND EQUIVALENT KTAILS
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
C CONSIDER EACH KTAIL STRING
      DO 30 I = 3, MAXSTA-1
      MAXWRD = ((Q(I,2) - 1) / 5) + 1
C
C COMPARE IT TO THE REST OF THE KTAIL STRINGS
      DO 30 J = I+1, MAXSTA
C
C TO BE THE SAME, THE KTAILS MUST HAVE THE SAME LENGTH
      IF(Q(I,2) .NE. Q(J,2)) GO TO 30
C
C IF THEY ARE BOTH FINAL STATES, THE KTAILS ARE EQUIVALENT
C (EMPTY)
      IF(Q(I,2) .NE. 0) GO TO 10
      Q(J,4) = Q(I,4)
      GO TO 30
C
C COMPARE THE KTAILS
10    DO 20 K = 1, MAXWRD
      IF(KTAIL(I,K) .NE. KTAIL(J,K)) GO TO 30
20    CONTINUE
C
C THE KTAILS MATCH, TAG THEM AS BEING EQUIVALENT
      Q(J,4) = Q(I,4)
30    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE ZEQUIV
C
C PROGRAM NAME - ZEQUIV
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 8, 1983
C PURPOSE      - FIND EQUIVALENT Z STRINGS
C              - BUILD LIST OF KTAILS ASSOCIATED WITH EACH
C              - DISTINCT Z
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
C CONSIDER EACH Z STRING
      DO 40 I = 3, MAXSTA-1
C
C IF THIS Z STRING HAS ALREADY BEEN EQUIVALENCED, SKIP IT
      IF(Q(I,3) .NE. I) GO TO 40
C
C ENTER ITS OWN KTAIL INTO THE ZKTAIL TABLE
      CALL ZKT(I, I)
C
C COMPARE IT TO THE REST OF THE Z STRINGS
      MAXWRD = ((Q(I,1) - 1) / 5) + 1
      DO 30 J = I+1, MAXSTA
C
C TO BE THE SAME, THE Z STRINGS MUST HAVE THE SAME LENGTH
      IF(Q(I,1) .NE. Q(J,1)) GO TO 30
C
C IF THEY ARE BOTH INITIAL STATES, THE Z STRINGS ARE
C EQUIVALENT (EMPTY)
      IF(Q(I,1) .EQ. 0) GO TO 20
C
C COMPARE THE Z STRINGS
      DO 10 K = 1, MAXWRD
      IF(Z(I,K) .NE. Z(J,K)) GO TO 30
10    CONTINUE
C
C THE Z STRINGS MATCH, TAG THEM AS BEING EQUIVALENT
20    Q(J,3) = Q(I,3)
      Q(J,5) = Q(I,5)
      CALL ZKT(I, J)
30    CONTINUE
40    CONTINUE
C
C IF THE LAST Z STRING HAS NOT BEEN EQUIVALENCED, ENTER ITS
C OWN KTAIL INTO THE ZKTAIL TABLE
      IF(Q(MAXSTA,3) .EQ. MAXSTA) CALL ZKT(MAXSTA, MAXSTA)
C
C ZKTAIL TABLE HAS NOW BEEN BUILT, SO SORT IT INTO ASCENDING
C ORDER
```

```
      DO 50 I = 3, MAXSTA
      IF(ZKTAIL(I,11) .LT. 2) GO TO 50
      DO 50 K = 2, ZKTAIL(I,11)
      KBACK = ZKTAIL(I,11) - K + 2
      DO 50 J = 2, KBACK
      IF(ZKTAIL(I,J-1) .LE. ZKTAIL(I,J)) GO TO 50
      SAVE = ZKTAIL(I,J-1)
      ZKTAIL(I,J-1) = ZKTAIL(I,J)
      ZKTAIL(I,J) = SAVE
50    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE ZKT(I,J)
C
C PROGRAM NAME - ZKT
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - ADD ENTRY TO LIST OF KTAILS ASSOCIATED WITH
C              - EACH DISTINCT Z
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
C FIND EQUIVALENT Z ASSOCIATED WITH Z NUMBER I
      IZ = Q(I,3)
C
C FIND EQUIVALENT KTAIL ASSOCIATED WITH KTAIL NUMBER I
      JKTAIL = Q(J,4)
C
C APPEND THIS KTAIL TO THE KTAILS LISTED FOR Z NUMBER I
      LOC = ZKTAIL(IZ,11) + 1
C
C CHECK IF THIS KTAIL HAS ALREADY BEEN LISTED
      IF(LOC .EQ. 1) GO TO 20
      DO 10 K = 1, LOC-1
      IF(ZKTAIL(IZ,K) .EQ. JKTAIL) GO TO 99999
10    CONTINUE
C
C THE KTAIL IS NOT IN THE LIST, SO ADD IT
20    ZKTAIL(IZ,LOC) = JKTAIL
C
C IF THIS KTAIL CORRESPONDS TO A FINAL STATE, FLAG IT
      IF(Q(J,2) .EQ. 0) ZKTAIL(IZ,12) = 1
C
C UPDATE THE COUNT OF KTAILS FOR THIS Z
      ZKTAIL(IZ,11) = LOC
C
C IF WE HAVE EXCEEDED THE MAXIMUM PERMISSIBLE NUMBER OF
C KTAILS FOR THIS Z, ALARM AND STOP
      IF(LOC .LE. 10) GO TO 99999
      TYPE 30, IZ
30    FORMAT(' TOO MANY KTAILS WERE FOUND FOR Z  ', I2)
      STOP
C
99999 RETURN
      END
```

```
      SUBROUTINE STATEQ
C
C PROGRAM NAME - STATEQ
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - DETERMINE EQUIVALENT STATES BY COMPARING
C              - KTAIL SETS
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION KTAIL(50,50), Z(50,50), Q(50,6)
      DIMENSION ZKTAIL(50,12), SPARE(7300)
      COMMON MAXSTA, KTAIL, SPARE, Z, Q, ZKTAIL
C
C CONSIDER EACH STATE
      DO 40 I = 3, MAXSTA-1
C
C COMPARE ITS LIST OF KTAILS TO THE LISTS OF THE OTHER
C STATES
      DO 30 J = I+1, MAXSTA
C
C IF THESE STATES HAVE ALREADY BEEN EQUIVALENCED, TRY THE
C NEXT ONE
      IF(Q(I,5) .EQ. Q(J,5)) GO TO 30
C
C TO BE EQUIVALENT, THEY MUST HAVE THE SAME NUMBER OF KTAILS
      IF(ZKTAIL(I,11) .NE. ZKTAIL(J,11)) GO TO 30
C
C IF THE KTAIL LISTS ARE THE SAME, THEY ARE EQUIVALENT
      DO 10 K = 1, ZKTAIL(I,11)
      IF(ZKTAIL(I,K) .NE. ZKTAIL(J,K)) GO TO 30
10    CONTINUE
C
C THEY ARE EQUIVALENT - ASSIGN STATE I TO STATE J
C (UNLESS ONE IS AN INITIAL STATE, THEN MAKE THEM BOTH
C INITIAL STATES)
      IF(Q(J,5) .EQ. 1) Q(I,5) = 1
      Q(J,5) = Q(I,5)
C
C LOOP BACK FOR NEXT COMPARISON STATE
30    CONTINUE
C
C LOOP BACK FOR NEXT STATE
40    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE TRANSI
C
C PROGRAM NAME - TRANSI
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - OBTAIN THE (NONDETERMINISTIC) TRANSITION
C              - TABLE DELTA FROM THE STATES
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DELTA(100,8,12), FINAL(100), MAP(100)
      DIMENSION Z(50,50), Q(50,6), ZKTAIL(50,12)
      COMMON MAXSTA, DELTA, FINAL, MAP, Z, Q, ZKTAIL
      DATA DELMAX/12/, MAXSIZ/100/
C
C CLEAR THE DELTA, FINAL, AND MAP ARRAYS
      DO 10 I = 1, MAXSIZ
      FINAL(I) = 0
      MAP(I) = 0
      DO 10 J = 1, 8
      DO 10 K = 1, DELMAX
      DELTA(I, J, K) = 0
10    CONTINUE
C
C INITIALIZE PTRSTA AND MAP (USED IN DELBLD)
      PTRSTA = 2
      MAP(1) = 1
      MAP(2) = 2
C
C ENTER THE PHI-TO-PHI TRANSITIONS INTO THE TRANSITION TABLE
      DO 15 I = 1, 8
      DELTA(2,I,1) = 2
15    CONTINUE
C
C CONSIDER EACH Z STRING
      DO 60 I = 3, MAXSTA
C
C SKIP THIS Z STRING IF IT IS A REPEAT
      IF(Q(I,3) .NE. I) GO TO 60
C
C FOR EACH POSSIBLE INPUT SYMBOL, APPEND THE SYMBOL TO THE Z
C STRING, THEN SEE IF THIS NEW STRING MATCHES ANY STATE'S
C Z STRING
C
C FIND THE LOCATION AFTER THE LAST SYMBOL IN THE Z STRING
      WORD = 1 + (Q(I,1) / 5)
      LOC = 1 + Q(I,1) - (5 * (WORD - 1))
C
C SAVE THE WORD WE ARE CHANGING SO THAT IT CAN LATER BE
C RESTORED
      WRDSAV = Z(I, WORD)
C
C FIND THE NUMBER OF THE POSITION OF THE SYMBOL TO BE
```

```
C APPENDED
      LENTST = Q(I,1) + 1
C
C ONE AT A TIME, APPEND THE INPUT SYMBOLS
      DO 50 J = 1, 8
      INPUT = J - 1
      IF(J .EQ. 1) GO TO 20
C
C APPEND THE INPUT SYMBOL
      Z(I,WORD) = WRDSAV + (INPUT * (8 ** (LOC - 1)))
C
C COMPARE THE NEW Z STRING TO THE OTHER STATES' Z STRINGS -
C SEARCH FOR A MATCH
20    DO 40 K = 3, MAXSTA
C
C THE Z STRINGS MUST BE OF THE SAME LENGTH
      IF(LENTST .NE. Q(K,1)) GO TO 40
C
C COMPARE THE Z STRINGS
      DO 30 L = 1, WORD
      IF(Z(I,L) .NE. Z(K,L)) GO TO 40
30    CONTINUE
C
C THE Z STRINGS MATCH - ADD THIS INFORMATION TO THE
C TRANSITION TABLE AND QUIT LOOKING FOR A MATCH
      CALL DELBLD(I, K, INPUT, PTRSTA)
      GO TO 50
C
C LOOP BACK FOR NEXT COMPARISON STRING
40    CONTINUE
C
C NO MATCH FOUND - ADD THIS INFORMATION TO THE TRANSITION
C TABLE
      CALL DELBLD(I, 2, INPUT, PTRSTA)
C
C LOOP BACK FOR NEXT INPUT SYMBOL TO BE APPENDED
50    CONTINUE
C
C RESTORE THE LAST Z STRING WORD
      Z(I,WORD) = WRDSAV
C
C LOOP BACK FOR NEXT Z STRING TO BE APPENDED
60    CONTINUE
C
C NOW CHECK THAT ALL THE TRANSITIONS FROM THE PHI STATE TO
C OTHER STATES ARE PUT INTO THE TRANSITION TABLE
C
C FIND THE SMALLEST Z STRING FORMED FROM EACH INPUT STRING
      DO 100 I = 3, MAXSTA
      IF(Q(I,6) .NE. 1) GO TO 100
C
C IF THE Z STRING IS EMPTY, DON'T BOTHER
```

```
      IF(Q(I,1) .EQ. 0) GO TO 100
C
C FIND THE LOCATION OF THE LAST SYMBOL IN THE Z STRING
      WORD = 1 + ((Q(I,1) - 1) / 5)
      LOC = Q(I,1) - (5 * (WORD - 1))
      LENTST = Q(I,1) - 1
C
C SAVE THE LAST WORD IN THE Z STRING
      WRDSAV = Z(I,WORD)
C
C WHAT INPUT SYMBOL WILL BE REMOVED?
      INPUT = Z(I,WORD) / (8 ** (LOC - 1))
C
C REMOVE THE LAST SYMBOL IN THE Z STRING
      Z(I,WORD) = Z(I,WORD) - (INPUT * (8 ** (LOC - 1)))
C
C SEARCH FOR A Z STRING WHICH MATCHES THE TRUNCATED STRING
      DO 80 J = 3, MAXSTA
C
C Z STRINGS MUST BE THE SAME LENGTH
      IF(LENTST .NE. Q(J,1)) GO TO 80
C
C COMPARE THE Z STRINGS
      DO 70 K = 1, WORD
      IF(Z(I,K) .NE. Z(J,K)) GO TO 80
70    CONTINUE
C
C THE Z STRINGS MATCH, THEREFORE THIS TRANSITION WAS TAKEN
C CARE OF IN THE FIRST PART OF THIS SUBROUTINE - GO TO THE
C NEXT STATE
      GO TO 90
C
C NO MATCH, SO COMPARE TO THE NEXT Z STRING
80    CONTINUE
C
C NO MATCHES FOUND, THIS IS A TRANSITION FROM THE PHI STATE,
C ADD THIS INFORMATION TO THE TRANSITION TABLE
      CALL DELBLD(2, I, INPUT, PTRSTA)
C
C RESTORE THE Z STRING
90    Z(I,WORD) = WRDSAV
C
C GET NEXT Z STRING TO CHECK FOR POSSIBLE TRANSITION FROM
C THE PHI STATE
100   CONTINUE
C
C NOW THAT THE TRANSITION TABLE IS BUILT, ANY INPUT NOT
C CAUSING A TRANSITION FROM THE INITIAL STATE TO A VALID
C STATE SHOULD CAUSE A TRANSITION TO THE PHI STATE
      DO 110 I = 1, 8
      IF(DELTA(1,I,1) .EQ. 0) DELTA(1,I,1) = 2
110   CONTINUE
```

```
C
C SAVE THE HIGHEST MAPPED STATE
      MAXSTA = PTRSTA
C
      RETURN
      END
```

```
      SUBROUTINE DELBLD(I1, I2, INPUT, PTRSTA)
C
C PROGRAM NAME - DELBLD
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - ADD ENTRY INTO TRANSITION TABLE FROM STATE
C              - OF Z STRING I1 TO STATE OF Z STRING I2 VIA
C              - INPUT SYMBOL INPUT,
C              - REMOVE BLANK ROWS FROM THE TRANSITION TABLE
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DELTA(100,8,12), FINAL(100), MAP(100)
      DIMENSION Z(50,50), Q(50,6), ZKTAIL(50,12)
      COMMON MAXSTA, DELTA, FINAL, MAP, Z, Q, ZKTAIL
      DATA DELMAX/12/, MAXSIZ/100/
C
C WHICH STATES ARE INVOLVED IN THE TRANSITION?
      FROM = Q(I1,5)
      TO = Q(I2,5)
C
C WHAT ARE THE NEW STATES TO WHICH THESE ARE MAPPED?
C
C HAS THIS FROM STATE BEEN MAPPED?
      IF(MAP(FROM) .NE. 0) GO TO 10
C
C MAP THE FROM STATE INTO A NEW STATE
      PTRSTA = PTRSTA + 1
      IF(PTRSTA .LE. MAXSIZ) GO TO 8
      TYPE 5, MAXSIZ
5     FORMAT(' EXCEEDED ', I3, ' STATES WHEN BUILDING THE',
     $       ' NONDETERMINISTIC TRANSITION TABLE')
      STOP
C
8     MAP(FROM) = PTRSTA
C
C IF THE OLD STATE IS A FINAL STATE, FLAG THE NEW STATE AS
C SUCH
10    IF(ZKTAIL(I1,12) .EQ. 1) FINAL(MAP(FROM)) = 1
      FROM = MAP(FROM)
C
C HAS THIS TO STATE BEEN MAPPED?
      IF(MAP(TO) .NE. 0) GO TO 20
C
C MAP THE TO STATE INTO A NEW STATE
      PTRSTA = PTRSTA + 1
      IF(PTRSTA .LE. MAXSIZ) GO TO 18
      TYPE 5, MAXSIZ
      STOP
C
18    MAP(TO) = PTRSTA
C
C IF THE OLD STATE IS A FINAL STATE, FLAG THE NEW STATE AS
```

```
C SUCH
20     IF(ZKTAIL(I2,12) .EQ. 1) FINAL(MAP(TO)) = 1
       TO = MAP(TO)
C
C HAS THIS TRANSITION ALREADY BEEN ENTERED INTO THE
C TRANSITION TABLE?
       DO 30 I = 1, DELMAX
       IF(DELTA(FROM, INPUT+1, I) .EQ. TO) GO TO 99999
C
C ARE WE AT THE END OF THE TABLE?
       IF(DELTA(FROM, INPUT+1, I) .NE. 0) GO TO 30
C
C MAKE THE NEW TRANSITION TABLE ENTRY
       DELTA(FROM, INPUT+1, I) = TO
       GO TO 99999
30     CONTINUE
C
C NO ROOM LEFT IN DELTA
       TYPE 40, DELMAX, I1, I2, INPUT
40     FORMAT(' EXCEEDED ', I2, ' MULTIPLE STATES FROM Z  ',
      $        I2, ' TO Z  ',I2, ' WITH INPUT SYMBOL ', I1)
C
99999 RETURN
       END
```

```
      SUBROUTINE DETERM
C
C PROGRAM NAME - DETERM
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - OBTAIN THE DETERMINISTIC AUTOMATON FROM THE
C              - TRANSITION TABLE
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DELTA(100,8,12), FINAL(100), TRANSI(100,8)
      DIMENSION MAP(100,3)
      COMMON MAXSTA, DELTA, FINAL, TRANSI, MAP
      DATA DELMAX/12/, MAXSIZ/100/
C
C INISTA = INITIAL STATE OF THE CURRENT PASS
C MAXOLD = MAXIMUM STATE OF THE CURRENT PASS
C INIMAX = MAXIMUM STATE OF THE FIRST PASS
C MAXSTA = MAXIMUM STATE (INCLUDING NEWLY CREATED MULTIPLE
C          STATES)
C
C ZERO TRANSI AND MAP ARRAYS
      DO 20 I = 1, MAXSIZ
      DO 10 J = 1, 3
      TRANSI(I, J) = 0
      MAP(I, J) = 0
10    CONTINUE
      DO 20 J = 4, 8
      TRANSI(I, J) = 0
20    CONTINUE
C
C THE INITIAL STATE FOR THE FIRST PASS IS THE FIRST STATE
      INISTA = 1
C
C SAVE THE MAXIMUM STATE FROM THE INITIAL PASS
      MAXOLD = MAXSTA
      INIMAX = MAXSTA
C
C COPY THE DELTA ARRAY TO THE TRANSI ARRAY
C ANY MULTIPLE STATES WILL BE ASSIGNED NEW STATES
C
C CONSIDER EACH STATE AND EACH INPUT
25    DO 90 I = INISTA, MAXOLD
      DO 80 J = 1, 8
C
C IS THIS TRANSITION TO A MULTIPLE STATE?
      IF(DELTA(I,J,2) .NE. 0) GO TO 30
C
C TRANSITION IS TO A SINGLE STATE - COPY THE TRANSITION AND
C MOVE TO NEXT ENTRY
      TRANSI(I,J) = DELTA(I,J,1)
      GO TO 80
C
```

```
C TRANSITION IS TO A MULTIPLE STATE
C
C HOW MANY STATES ARE THERE?
C ARE ANY OF THEM A FINAL STATE?
30     FINFLG = 0
       DO 40 K = 1, DELMAX
       MULTI = DELTA(I,J,K)
       IF(FINAL(MULTI) .EQ. 1) FINFLG = 1
       IF(MULTI .NE. 0) NUMBER = K
40     CONTINUE
C
C DOES THIS MATCH ANY PREVIOUS MULTIPLE STATE?
C (OBVIOUSLY CAN NOT ON THE FIRST PASS)
       IF(MAXSTA .EQ. INIMAX) GO TO 70
       DO 60 K = INIMAX+1, MAXOLD
C
C MUST HAVE SAME NUMBER OF STATES TO MATCH
       IF(NUMBER .NE. MAP(K,3)) GO TO 60
C
C ARE THE STATES EQUIVALENT?
       II = MAP(K,1)
       JJ = MAP(K,2)
       DO 50 L = 1, NUMBER
       IF(DELTA(I,J,L) .NE. DELTA(II,JJ,L)) GO TO 60
50     CONTINUE
C
C THEY MATCH - USE THE EXISTING MULTIPLE STATE
       TRANSI(I,J) = K
       GO TO 80
C
C LOOP BACK FOR NEXT MULTIPLE STATE TO TEST FOR MATCH
60     CONTINUE
C
C NO MATCHES - CREATE A NEW MULTIPLE STATE
70     MAXSTA = MAXSTA + 1
       TRANSI(I,J) = MAXSTA
C
C CHECK IF WE HAVE EXCEEDED THE ARRAY SPACE
       IF(MAXSTA .LE. MAXSIZ) GO TO 78
       TYPE 76, I, MAXSTA, J-1
76     FORMAT(' TOO MANY NEW STATES HAVE BEEN CREATED (IN',
      $        ' DETERM)', /, ' TRANSITION FROM STATE ', I3,
      $        ' TO NEW STATE ', I3, ' WITH INPUT SYMBOL ',
      $        I1)
       STOP
C
C SAVE THE NEW-STATE-TO-MULTIPLE-STATE MAPPING AND FINAL
C STATE STATUS
78     MAP(MAXSTA,1) = I
       MAP(MAXSTA,2) = J
       MAP(MAXSTA,3) = NUMBER
       FINAL(MAXSTA) = FINFLG
```

```
C
C LOOP FOR NEXT INPUT
80    CONTINUE
C
C LOOP FOR NEXT STATE
90    CONTINUE
C
C WHICH ARE THE NEW STATES?
      INISTA = MAXOLD + 1
      MAXOLD = MAXSTA
C
C WERE ANY NEW STATES CREATED? IF NOT, WE ARE DONE
      IF(INISTA .GT. MAXSTA) GO TO 99999
C
C DETERMINE THE TRANSITIONS FROM THE NEW STATES, PUT THEM
C INTO THE DELTA ARRAY
      DO 190 I = INISTA, MAXSTA
C
C FIND WHERE WE WOULD GO FROM EACH COMPONENT OF THIS
C MULTIPLE STATE AND STORE THESE TRANSITIONS IN THE DELTA
C ARRAY
C
C FIND THE COMPONENTS
      II = MAP(I,1)
      JJ = MAP(I,2)
      DO 130 K = 1, DELMAX
      FROM = DELTA(II,JJ,K)
      IF(FROM .EQ. 0) GO TO 130
C
C CONSIDER EACH INPUT FOR THIS COMPONENT
      DO 120 J = 1, 8
C
C FOR THIS INPUT, CONSIDER EACH TRANSITION
      DO 115 M = 1, DELMAX
      TO = DELTA(FROM, J, M)
      IF(TO .EQ. 0) GO TO 115
C
C HAS THIS TRANSITION ALREADY BEEN ENTERED INTO THE DELTA
C ARRAY?
      DO 100 L = 1, DELMAX
      IF(DELTA(I,J,L) .EQ. TO) GO TO 115
C
C ARE WE AT THE END OF THE TABLE?
      IF(DELTA(I,J,L) .EQ. 0) GO TO 110
100   CONTINUE
C
C NO MORE ROOM IN DELTA ARRAY
      TYPE 105, I, TO, J-1
105   FORMAT(' NO ROOM IN DELTA FOR TRANSITION FROM STATE',
     $       I3, ' TO STATE', I3, ' WITH INPUT SYMBOL ',I1)
C
C ENTER THIS TRANSITION INTO DELTA
```

```
110   DELTA(I,J,L) = TO
C
C LOOP BACK FOR NEXT TRANSITION
115   CONTINUE
C
C LOOP BACK FOR NEXT INPUT
120   CONTINUE
C
C LOOP BACK FOR NEXT COMPONENT
130   CONTINUE
C
C NOW, IF ANY MULTIPLE STATES HAVE BEEN CREATED, CHECK THAT
C THE STATES ARE IN ASCENDING ORDER
C
C LOOK FOR TRANSITIONS TO MULTIPLE STATES
      DO 180 J = 1, 8
      IF(DELTA(I,J,2) .EQ. 0) GO TO 180
C
C A MULTIPLE STATE - SORT THE STATES INTO ASCENDING ORDER
      DO 140 KK = 2, DELMAX
      KKK = DELMAX - KK + 2
      DO 140 K = 2, KKK
      IF(DELTA(I,J,K) .EQ. 0) GO TO 140
      IF(DELTA(I,J,K) .GT. DELTA(I,J,K-1)) GO TO 140
      SAVE = DELTA(I,J,K)
      DELTA(I,J,K) = DELTA(I,J,K-1)
      DELTA(I,J,K-1) = SAVE
140   CONTINUE
C
C LOOP BACK FOR NEXT INPUT, THEN SORT ANY MULTIPLE STATES
C FOUND
180   CONTINUE
C
C LOOP BACK TO DETERMINE THE DELTA TRANSITIONS FOR THE NEXT
C NEW STATE
190   CONTINUE
C
C NOW GO BACK AND COPY THE DELTA ARRAY TO THE TRANSI ARRAY,
C CHECKING FOR ANY REMAINING MULTIPLE STATES
      GO TO 25
C
99999 RETURN
      END
```

```
      SUBROUTINE STELIM
C
C PROGRAM NAME - STELIM
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 17, 1983
C PURPOSE      - ELIMINATE INACCESSIBLE STATES FROM THE
C              - DETERMINISTIC AUTOMATON
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DELTA(100,8,12), FINAL(100), TRANSI(100,8)
      DIMENSION MAP(100,3)
      COMMON MAXSTA, DELTA, FINAL, TRANSI, MAP
C
C INITIALIZE THE MAP ARRAY
      DO 10 I = 1, MAXSTA
      MAP(I,1) = 1
10    CONTINUE
C
C INITIALIZE MAP POINTERS
      OLDPTR = 2
      NEWPTR = 1
C
C SWITCH NEWPTR AND OLDPTR
20    SAVE = NEWPTR
      NEWPTR = OLDPTR
      OLDPTR = SAVE
C
C ZERO THE NEW PART OF MAP
      DO 30 I = 1, MAXSTA
      MAP(I,NEWPTR) = 0
30    CONTINUE
C
C FOR EVERY STATE FLAGGED IN THE OLD PART OF MAP, CONSIDER
C THE TRANSI TRANSITIONS
C IN THE NEW PART OF MAP, FLAG EACH STATE THAT CAN BE
C REACHED FROM THE ABBREVIATED TRANSI LIST
      DO 40 I = 1, MAXSTA
      DO 40 J = 1, 8
      IF(MAP(I,OLDPTR) .EQ. 0) GO TO 40
      INDEX = TRANSI(I,J)
      MAP(INDEX,NEWPTR) = 1
40    CONTINUE
C
C WE CAN ALWAYS GET TO THE INITIAL STATE
      MAP(1,NEWPTR) = 1
C
C IF THE NEW PART OF MAP MATCHES THE OLD PART OF MAP, WE
C HAVE FOUND ALL THE INACCESSIBLE STATES
      FLAG = 0
      DO 50 I = 1, MAXSTA
      IF(MAP(I,NEWPTR) .NE. MAP(I,OLDPTR)) GO TO 20
      IF(MAP(I,NEWPTR) .EQ. 0) FLAG = 1
```

```
50    CONTINUE
C
C THE NEW AND OLD PARTS MATCH - WE HAVE FOUND ALL THE
C INACCESSIBLE STATES
C
C IF NO STATES HAVE BEEN ELIMINATED, WE ARE DONE
      IF(FLAG .EQ. 0) GO TO 99999
C
C REMOVE THE INACCESSIBLE STATES FROM TRANSI
      POINTR = 0
      DO 80 I = 1, MAXSTA
C
C INCREMENT POINTR TO THE NEXT ACCESSIBLE STATE
60    POINTR = POINTR + 1
      IF(POINTR .GT. MAXSTA) GO TO 90
      IF(MAP(POINTR,NEWPTR) .EQ. 0) GO TO 60
C
C REPLACE THE CURRENT STATE WITH THE NEXT ACCESSIBLE ONE
      FINAL(I) = FINAL(POINTR)
      DO 70 J = 1, 8
      TRANSI(I,J) = TRANSI(POINTR,J)
70    CONTINUE
C
C SAVE THE NEW-STATE-TO-OLD-STATE MAPPING
      MAP(POINTR,3) = I
C
80    CONTINUE
C
C THE INACCESSIBLE STATES HAVE BEEN REMOVED
C HOW MANY STATES ARE LEFT?
90    MAXSTA = I - 1
C
C CHANGE THE TRANSITIONS TO SHOW THE REARRANGED STATES
      DO 100 I = 1, MAXSTA
      DO 100 J = 1, 8
      INDEX = TRANSI(I,J)
      TRANSI(I,J) = MAP(INDEX,3)
100   CONTINUE
C
99999 RETURN
      END
```

```
      SUBROUTINE MINIMA
C
C PROGRAM NAME - MINIMA
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 24, 1983
C PURPOSE      - MINIMIZE THE DETERMINISTIC AUTOMATON BY
C              - COMBINING EQUIVALENT STATES
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION TABLE(100,10), EXTRA(8600), FINAL(100)
      DIMENSION TRANSI(100,8), MAP(100,3), ZERO(1000)
      COMMON MAXSTA, TABLE, EXTRA, FINAL, TRANSI, MAP
      EQUIVALENCE (ZERO(1), TABLE(1,1))
C
C ZERO THE TABLE ARRAY
      DO 10 I = 1, 1000
      ZERO(I) = 0
10    CONTINUE
C
C IF ALL THE TRANSITIONS FROM THE INITIAL STATE ARE TO THE
C PHI STATE, THEN THESE TWO STATES ARE EQUIVALENT
      DO 12 I = 1, 8
      IF(TRANSI(1,I) .NE. 2) GO TO 16
12    CONTINUE
C
C THE INITIAL STATE AND THE PHI STATE ARE EQUIVALENT - MAKE
C THE INITIAL STATE TRANSITIONS THE SAME AS THE PHI STATE
C TRANSITIONS SO THIS SUBROUTINE WILL COMBINE THEM
      DO 14 I = 1, 8
      TRANSI(1,I) = TRANSI(2,I)
14    CONTINUE
C
C X OUT ALL PAIRS OF STATES THAT ARE NOT EITHER BOTH FINAL
C STATES OR BOTH NON-FINAL STATES
16    DO 20 I = 1, MAXSTA-1
      DO 20 J = I+1, MAXSTA
      IF(FINAL(I) .EQ. FINAL(J)) GO TO 20
      CALL X(I, J)
20    CONTINUE
C
C RESET THE X'D-OUT-STATE FLAG
      XFLAG = 0
C
C FOR EACH PAIR OF STATES, EXAMINE THE TRANSITION STATES FOR
C EACH INPUT. IF THE PAIR OF TRANSITION STATES FOR ANY INPUT
C HAS BEEN DETERMINED TO BE NOT EQUIVALENT (X'D OUT), X OUT
C THESE STATES (THEY CAN NOT BE EQUIVALENT)
30    DO 80 I = 1, MAXSTA-1
      DO 70 J = I+1, MAXSTA
C
C IF THIS PAIR OF STATES IS ALREADY X'D OUT, SKIP IT
      CALL STATUS(I, J, S)
```

```
      IF(S .EQ. 1) GO TO 70
C
C GET THE TRANSITION STATES FOR THIS PAIR FOR EACH INPUT
      DO 60 K = 1, 8
      ITEST = TRANSI(I,K)
      JTEST = TRANSI(J,K)
C
C IF THE TRANSITION STATES ARE THE SAME, NO NEED TO TEST
C THEM.  OTHERWISE, ITEST MUST BE LESS THAN JTEST
      IF(JTEST - ITEST) 40, 60, 50
C
C ITEST > JTEST, SO SWITCH THEM
40    SAVE = ITEST
      ITEST = JTEST
      JTEST = SAVE
C
C HAVE THESE TWO TRANSITIONS BEEN X'D OUT?
50    CALL STATUS(ITEST, JTEST, S)
C
C IF SO, X OUT THIS PAIR OF STATES
      IF(S .EQ. 0) GO TO 60
      CALL X(I, J)
      XFLAG = 1
      GO TO 70
60    CONTINUE
70    CONTINUE
80    CONTINUE
C
C IF WE HAVE NOT X'D OUT ANY NEW PAIRS OF STATES, WE ARE
C DONE WITH THIS PART
      IF(XFLAG .EQ. 0) GO TO 90
C
C REPEAT UNTIL NO MORE STATES CAN BE X'D OUT
      XFLAG = 0
      GO TO 30
C
C IN THE TRANSITION TABLE, FOR EACH EQUIVALENT PAIR USE ONLY
C THE LOWEST NUMBERED STATE
90    DO 110 I = 1, MAXSTA-1
      IBACK = MAXSTA - I
      DO 110 J = IBACK+1, MAXSTA
      JBACK = MAXSTA - J + IBACK + 1
C
C IS THIS PAIR EQUIVALENT?
      CALL STATUS(IBACK, JBACK, S)
      IF(S .EQ. 1) GO TO 110
C
C THEY ARE EQUIVALENT - IN THE TRANSITION TABLE, REPLACE ALL
C OCCURENCES OF JBACK WITH IBACK
      DO 100 STATE = 1, MAXSTA
      DO 100 INPUT = 1, 8
      IF(TRANSI(STATE,INPUT) .EQ. JBACK)
```

```
     $  TRANSI(STATE,INPUT) = IBACK
100   CONTINUE
C
C GO BACK TO CHECK NEXT PAIR FOR EQUIVALENCE
110   CONTINUE
C
C NOW, ELIMINATE ALL INACCESSIBLE STATES (THESE ARE NOT
C NEEDED BECAUSE THEY ARE EQUIVALENT TO SOME OTHER STATE)
      CALL STELIM
C
      RETURN
      END
```

```
      SUBROUTINE X(I, J)
C
C PROGRAM NAME - X
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 24, 1983
C PURPOSE      - FLAG TWO STATES AS NOT BEING EQUIVALENT
C              - (X THEM OUT)
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION TABLE(100,10)
      COMMON MAXSTA, TABLE
C
C SINCE TABLE IS SET UP FOR 10 STATE PAIRS PER ELEMENT, FIND
C THE LOCATION FOR THIS PAIR
      WORD = ((J - 1) / 10) + 1
      POWER = MOD(J, 10)
      IF(POWER .EQ. 0) POWER = 10
      ENTRY = 2 ** POWER
C
C FLAG (X OUT) THIS STATE PAIR
      TABLE(I,WORD) = TABLE(I,WORD) + ENTRY
C
      RETURN
      END
```

```
      SUBROUTINE STATUS(I, J, S)
C
C PROGRAM NAME - STATUS
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 24, 1983
C PURPOSE      - RETURN EQUIVALENCE STATUS OF A STATE PAIR
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION TABLE(100,10)
      COMMON MAXSTA, TABLE
C
C FIND THIS PAIR'S LOCATION IN TABLE
      WORD = ((J - 1) / 10) + 1
      POWER = MOD(J, 10)
      IF(POWER .EQ. 0) POWER = 10
      ENTRY = 2 ** POWER
C
C HAS THIS PAIR BEEN FLAGGED?
      TEST = TABLE(I,WORD) / ENTRY
      S = MOD(TEST, 2)
C
      RETURN
      END
```

```
      SUBROUTINE STORE
C
C PROGRAM NAME - STORE
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - WRITE THE DETERMINISTIC AUTOMATON TO DISK
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION DELTA(100,8,12), FINAL(100), TRANSI(100,8)
      DIMENSION MAP(100,3)
      COMMON MAXSTA, DELTA, FINAL, TRANSI, MAP
C
C OPEN THE OUTPUT FILE
      OPEN(UNIT=4, NAME='AUTOMA.DAT', TYPE='NEW', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
C
C WRITE THE NUMBER OF STATES
      WRITE(4,100) MAXSTA
100   FORMAT(1X, I3)
C
C WRITE THE TRANSITIONS
      DO 30 I = 1, MAXSTA
      WRITE(4,200) (TRANSI(I,J), J = 1,8)
30    CONTINUE
200   FORMAT(8(1X, I3))
C
C WRITE THE FINAL ARRAY
      DO 40 I = 1, MAXSTA
      WRITE(4,100) FINAL(I)
40    CONTINUE
      CLOSE(UNIT=4, DISPOSE='SAVE')
C
      RETURN
      END
```

SEPAR

```
      PROGRAM SEPAR
C
C PROGRAM NAME - SEPAR
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - APRIL 6, 1983
C PURPOSE      - CHECK FOR SEPATABILITY OF THE CLASSES
C              - RECOGNIZED BY TWO DETERMINISTIC
C              - FINITE-STATE AUTOMATA BY CREATING AN
C              - AUTOMATON WHICH RECOGNIZES ONLY THOSE
C              - STRINGS WHICH CAN BE RECOGNIZED BY BOTH
C              - ORIGINAL AUTOMATA
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 FNAME(36)
      DIMENSION AUT1(100,8), AUT2(100,8), AUTCMB(100,8)
      DIMENSION FINAL1(100), FINAL2(100), FINCMB(100)
      DIMENSION STPAIR(100,2)
C
C READ IN FIRST AUTOMATON
      TYPE 1
1     FORMAT(1I1)
      CALL GETFNM('FILE CONTAINING FIRST AUTOMATON?', 'DAT',
     $            FNAME, NCHAR)
      OPEN(UNIT=1, NAME=FNAME, TYPE='OLD', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
      READ(1,10) MAX1
10    FORMAT(1X, I3)
      DO 20 I = 1, MAX1
      READ(1,30) (AUT1(I,J), J = 1, 8)
20    CONTINUE
30    FORMAT(8(1X, I3))
      DO 40 I = 1, MAX1
      READ(1,10) FINAL1(I)
40    CONTINUE
      CLOSE(UNIT=1, DISPOSE='SAVE')
C
C READ IN SECOND AUTOMATON
      TYPE 1
      CALL GETFNM('FILE CONTAINING SECOND AUTOMATON?',
     $            'DAT', FNAME, NCHAR)
      OPEN(UNIT=1, NAME=FNAME, TYPE='OLD', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
      READ(1,10) MAX2
      DO 50 I = 1, MAX2
      READ(1,30) (AUT2(I,J), J = 1, 8)
50    CONTINUE
      DO 60 I = 1, MAX2
      READ(1,10) FINAL2(I)
60    CONTINUE
      CLOSE(UNIT=1, DISPOSE='SAVE')
```

```
C
C GET NAME FOR OUTPUT FILE
      TYPE 1
      CALL GETFNM('OUTPUT FILE FOR COMBINED AUTOMATON?',
     $            'DAT', FNAME, NCHAR)
C
C INITIALIZE CURRENT STATE POINTER, MAXIMUM STATE POINTER,
C AND FIRST STATE PAIR
      CURSTA = 0
      MAXSTA = 1
      STPAIR(1,1) = 1
      STPAIR(1,2) = 1
      FINCMB(1) = 0
      IF((FINAL1(1) .EQ. 1) .AND. (FINAL2(1) .EQ. 1))
     $  FINCMB(1) = 1
C
C INCREMENT CURRENT STATE POINTER
70    CURSTA = CURSTA + 1
C
C DETERMINE THE TRANSITIONS FROM THE CURRENT STATE PAIR FOR
C EACH INPUT
      DO 100 I = 1, 8
      NEXT1 = AUT1(STPAIR(CURSTA,1), I)
      NEXT2 = AUT2(STPAIR(CURSTA,2), I)
C
C HAS THIS NEW STATE PAIR ALREADY BEEN USED?
      DO 80 J = 1, MAXSTA
      IF((STPAIR(J,1) .NE. NEXT1) .OR.
     $   (STPAIR(J,2) .NE. NEXT2)) GO TO 80
C
C THIS PAIR HAS BEEN USED BEFORE, SO USE THE EXISTING
C COMBINED STATE FOR THIS PAIR
      NXTCMB = J
      GO TO 90
80    CONTINUE
C
C THIS PAIR HAS NOT BEEN USED BEFORE, SO CREATE A NEW STATE
      MAXSTA = MAXSTA + 1
      NXTCMB = MAXSTA
      STPAIR(NXTCMB,1) = NEXT1
      STPAIR(NXTCMB,2) = NEXT2
      FINCMB(NXTCMB) = 0
      IF((FINAL1(NEXT1) .EQ. 1) .AND.
     $   (FINAL2(NEXT2) .EQ. 1)) FINCMB(NXTCMB) = 1
C
C ENTER THIS TRANSITION INTO THE COMBINED AUTOMATON
90    AUTCMB(CURSTA,I) = NXTCMB
C
C GO BACK TO FIND THE TRANSITION FOR THE NEXT INPUT
100   CONTINUE
C
C IF WE HAVE FOUND THE TRANSITIONS FROM ALL THE COMBINED
```

```
C AUTOMATON'S STATES, THE COMBINED AUTOMATON IS COMPLETE
      IF(CURSTA .NE. MAXSTA) GO TO 70
C
C WRITE OUT THE COMBINED STATE AUTOMATON
      OPEN(UNIT=1, NAME=FNAME, TYPE='NEW', ACCESS=
     $     'SEQUENTIAL', FORM='FORMATTED')
      WRITE(1,10) MAXSTA
      DO 110 I = 1, MAXSTA
      WRITE(1,30) (AUTCMB(I,J), J = 1, 8)
110   CONTINUE
      DO 120 I = 1, MAXSTA
      WRITE(1,10) FINCMB(I)
120   CONTINUE
      CLOSE(UNIT=1, DISPOSE='SAVE')
C
      STOP
      END
```

RECOG

```
      PROGRAM RECOG
C
C PROGRAM NAME - RECOG
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - MARCH 9, 1983
C PURPOSE      - ATTEMPT RECOGNITION OF AN INPUT STRING
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 ANSWER, FNAME(36)
      DIMENSION ANGCHN(250), TRANSI(100,8), FINAL(100)
C
C GET THE NAME OF THE ANGLE STRING FILE
      CALL GETFNM('ANGLE STRING FILE?', 'DAT', FNAME, NCHAR)
C
C CHECK THAT THE INPUT FILE FOR THE AUTOMATON IS IN CORRECT
C FORMAT
      TYPE 40
40    FORMAT(///,' THE INPUT FILE CONTAINING THE AUTOMATON',
     $        ' SHOULD BE NAMED AUTOMA.DAT',/,
     $        ' IS AUTOMA.DAT PREPARED CORRECTLY?',/,
     $        ' ENTER YES OR NO')
C
      ACCEPT 50,ANSWER
50    FORMAT(1A1)
      IF(ANSWER .EQ. 'Y') GO TO 70
C
C INPUT FILE NOT CORRECT
      TYPE 60
60    FORMAT(///,' CORRECT AUTOMA.DAT AND TRY AGAIN')
      STOP
C
C ANGLE STRING FILE IS CORRECT - OPEN THE FILE
70    OPEN(UNIT=4, NAME=FNAME, TYPE='OLD',
     $     ACCESS='SEQUENTIAL', FORM='FORMATTED')
C
C READ IN ANGLE STRINGS
      READ(4,90) LENGTH
90    FORMAT(1X, I3)
      READ(4,100) (ANGCHN(I), I = 1, LENGTH)
100   FORMAT(10(1X, I1))
C
C CLOSE THE INPUT FILE FOR THE ANGLE STRING
      CLOSE(UNIT=4, DISPOSE='SAVE')
C
C OPEN THE INPUT FILE FOR THE AUTOMATON
      OPEN(UNIT=4, NAME='AUTOMA.DAT', TYPE='OLD',
     $     ACCESS='SEQUENTIAL', FORM='FORMATTED')
C
C READ IN THE AUTOMATON
      READ(4,90) MAXSTA
```

```
      DO 110 I = 1, MAXSTA
      READ(4,120) (TRANSI(I,J), J = 1, 8)
110   CONTINUE
120   FORMAT(8(1X, I3))
      DO 130 I = 1, MAXSTA
      READ(4,90) FINAL(I)
130   CONTINUE
C
C CLOSE THE INPUT FILE FOR THE AUTOMATON
      CLOSE(UNIT=4, DISPOSE='SAVE')
C
C NOW RUN THE INPUT STRING THROUGH THE AUTOMATON
      STATE = 1
      DO 140 I = 1, LENGTH
      INPUT = ANGCHN(I) + 1
      STATE = TRANSI(STATE, INPUT)
140   CONTINUE
C
C DID WE END IN A FINAL STATE?
      IF(FINAL(STATE) .EQ. 0) GO TO 160
C
C YES - ENDED IN A FINAL STATE
      TYPE 150
150   FORMAT(///, ' ACCEPTED', /,
     $        ' THIS INPUT STRING BELONGS TO THIS CLASS')
      GO TO 99999
C
C NO - DID NOT END IN A FINAL STATE
160   TYPE 170
170   FORMAT(///, ' NOT ACCEPTED', /,
     $        ' THIS INPUT STRING DOES NOT BELONG TO THIS',
     $        ' CLASS')
C
99999 STOP
      END
```

SEMAN

```
      PROGRAM SEMAN
C
C PROGRAM NAME - SEMAN
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - JUNE 11, 1983
C PURPOSE      - PERFORM SEMANTIC RECOGNITION ON THE
C              - SEMANTIC INFORMATION FILE FOR AN OBJECT
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 FNAME(36)
C
C OPEN AND READ THE SEMANTIC INFORMATION FILE
      CALL GETFNM('NAME OF THE SEMANTIC INFORMATION FILE?',
     $            'SEM', FNAME, NCHAR)
      OPEN(UNIT=1, NAME=FNAME, TYPE='OLD',
     $     ACCESS='SEQUENTIAL', FORM='FORMATTED')
      READ(1, 1000) AREA, PERIM, XCNTR, YCNTR, MAXIMA
1000  FORMAT(1X, I5)
C
C WHICH CLASSES HAVE THE OVERLAP?
10    TYPE 1010
1010  FORMAT('0INTO WHICH TWO CLASSES DOES',
     $       ' THIS OBJECT OVERLAP?', /,
     $       ' CLASS 1 - BOLT', /, ' CLASS 2 - SCREW', /,
     $       ' CLASS 3 - WIRENUT', /, ' CLASS 4 - SPLICE',
     $       /, ' ENTER AS TWO I1 FORMAT CLASS NUMBERS', /,
     $       ' (<CR> TO EXIT)')
      ACCEPT 1020, CLASS1, CLASS2
1020  FORMAT(2I1)
      IF(CLASS1 .EQ. 0) GO TO 99999
C
C ARRANGE THE CLASSES IN ASCENDING ORDER
      IF(CLASS1 .LT. CLASS2) GO TO 20
      SAVE = CLASS1
      CLASS1 = CLASS2
      CLASS2 = SAVE
C
C IDENTIFY APPROPRIATE SEMANTIC RECOGNITION ROUTINE
20    IF((CLASS1 .EQ. 1) .AND. (CLASS2 .EQ. 2)) GO TO 30
      IF((CLASS1 .EQ. 3) .AND. (CLASS2 .EQ. 4)) GO TO 40
C
C*************************************************************
C TO ADD ANOTHER SEMANTIC RECOGNITION ROUTINE, ADD THE
C APPROPRIATE 'IF STATEMENT' TO THE ONES ABOVE
C*************************************************************
C
      TYPE 1030
1030  FORMAT('0THE SEMANTIC RECOGNITON ROUTINE',
     $       ' DOES NOT EXIST TO', /,
     $       ' SEPARATE THESE TWO CLASSES.', /,
```

```
      $        ' TRY A DIFFERENT PAIR OR ADD THE',
      $        ' REQUIRED RECOGNITION', /,
      $        ' ROUTINE TO THIS PROGRAM.')
       GO TO 10
C
C CALL APPROPRIATE SEMANTIC RECOGNITON ROUTINE
C
C BOLT-SCREW
30     CALL BOLSCR(MAXIMA, RESULT)
       GO TO 500
C
C WIRENUT-SPLICE
40     CALL WIRSPL(RESULT, AREA, PERIM)
       GO TO 500
C
C*************************************************************
C TO ADD ANOTHER SEMANTIC RECOGNITION ROUTINE,
C ADD THE APPROPRIATE "CALL STATEMENT"
C AND "GO TO 100" TO THE ONES ABOVE
C*************************************************************
C
C REPORT THE RESULTS OF THE SEMATIC RECOGNITON PROCESS
500    IF(RESULT .EQ. 0) GO TO 510
       TYPE 1040, RESULT
1040   FORMAT('0THE OBJECT BELONGS TO CLASS ', I1)
       GO TO 10
C
C NO DECISION COULD BE REACHED
510    TYPE 1050, CLASS1, CLASS2
1050   FORMAT('0THE SEMANTIC ALGORITHM COULD',
      $        ' NOT RESOLVE THE CLASS', /,
      $        ' OVERLAP FOR THIS OBJECT BETWEEN CLASSES',
      $        I1, ' AND ', I1, '.', /,
      $        ' TRY MODIFYING THE SEMANTIC',
      $        ' RECOGNITON ALGORITHM', /,
      $        ' OR OBTAINING A DIFFERENT REPRESENTATION',
      $        ' FOR THIS OBJECT.')
       GO TO 10
C
99999  STOP
       END
```

```
      SUBROUTINE BOLSCR(MAXIMA, RESULT)
C
C PROGRAM NAME - BOLSCR
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - JUNE 11, 1983
C PURPOSE      - USE SEMANTIC RECOGNITON TECHNIQUES TO
C              - ASSIGN AN OBJECT TO EITHER THE BOLT CLASS
C              - (1) OR THE SCREW CLASS (2)
C
      IMPLICIT INTEGER (A-Z)
C
C MAXIMA IS THE NUMBER OF POINTS ON THE OBJECT PERIMETER
C WHICH ARE LOCAL MAXIMA OF THE FUNCTION OF DISTANCE FROM
C THE OBJECT CENTROID TO BOUNDARY PIXELS
C
C DUE TO LENGTHWISE SYMMETRY, BOLTS HAVE AN EVEN NUMBER OF
C MAXIMA POINTS WHILE SCREWS HAVE AN ODD NUMBER (ONLY ONE
C MAXIMUM POINT AT THE TIP)
      RESULT = 1
      IF(MOD(MAXIMA, 2) .EQ. 1) RESULT = 2
C
      RETURN
      END
```

```
      SUBROUTINE WIRSPL(RESULT, AREA, PERIM)
C
C PROGRAM NAME - WIRSPL
C AUTHOR       - DAVID J. BRAUNEGG
C DATE         - JUNE 11, 1983
C PURPOSE      - USE SEMANTIC RECOGNITON TECHNIQUES TO
C              - ASSIGN AN OBJECT TO EITHER THE WIRENUT
C              - CLASS (3) OR THE SPLICE CLASS (4)
C
      IMPLICIT INTEGER (A-Z)
      DATA SEP/23/
C
C CALCULATE THE PERIMETER SQUARED DIVIDED BY THE AREA AS A
C MEASURE OF THE 'COMPACTNESS' OF THE OBJECT (SPLICES ARE
C LONG AND THIN WHILE WIRENUTS ARE MORE NEARLY SQUARE)
      PSA = IFIX((FLOAT(PERIM) ** 2) / FLOAT(AREA))
C
C CHOOSE THE BEST FIT CLASS
      RESULT = 3
      IF(PSA .GE. SEP) RESULT = 4
C
      RETURN
      END
```

VITA

David Jerome Braunegg was born in Charleroi, Pennsylvania on July 4, 1958. He graduated as class valedictorian from Charleroi Area High School in June 1976. The following September he entered the University of Pittsburgh, and in April 1980 he graduated Summa Cum Laude with a Bachelor of Science in Electrical Engineering degree. In May 1980 he accepted a full-time position as an Electrical Engineer with the Aluminum Company of America in Alcoa, Tennessee. In September 1980 he entered the Graduate School of The University of Tennessee, Knoxville. He received the Master of Engineering degree with a major in Electrical Engineering in August 1983.

The author is a member of Phi Eta Sigma, Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi, the Institute of Electrical and Electronics Engineers, and the Association for Computing Machinery. Mr. Braunegg will be continuing his graduate study in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology beginning September 1983.