THE LEARNING OF PARAMETERS FOR GENERATING COMPOUND CHARACTERIZERS FOR PATTERN RECOGNITION

by

Leonard Uhr and Sara Jordan
University of Wisconsin
Madison, Wisconsin U.S.A.

Abstract

This paper presents and describes a pattern recognition program with a relatively simple and general basic structure upon which has been superimposed a rather wide variety of techniques for learning, or self-organization. The program attempts to generalize n-tuple approaches to pattern recognition, in which an n-tuple is a set of individual cells or small pieces of patterns, and each n-tuple is said to characterize an input pattern when these pieces match it, as specified.

The program allows n-tuples to match when only _some_ of their parts match, and it allows these parts to match even though they are _not_ precisely positioned (See Uhr, 1969b, for some simple example programs). It further learns, in a variety of ways: It searches for good weights on its characterizers' implications, byre-weighting as a function of feedback. It generates and discovers new characterizers (and can therefore begin with no characterizers at all), and discards characterizers that prove to be poor (See Uhr and Vossler, 1961, and Prather and Uhr, 1964). It also uses a set of characterizers of characterizers, to search for good parameter values that newly-generated characterizers should have.

A detailed flow-chart-like "precis" description of the program is given, along with an actual listing. It is thus possible to examine _exactly_ what the program does, and how it does it, and therefore to see how a wide variety of learning mechanisms have been implemented in a single pattern recognition program. But because it was coded in a "high-level" pattern-matching and list-processing language the program runs too slowly for extensive tests to be practicable. Therefore only a brief listing of output is given, to show that the program, works and begins to learn.

Descriptors: Learning, self-organization, induction, discovery, pattern recognition, learning to learn, n-tuple recognition, characterizing characterizers.

Introduction

Programs that have used n-tuples as their characterizers appear to perform with the very best of pattern recognition programs (for discussions, see Uhr, 1963, 1969a; for a good recent example, see Andrews, Atrubin, and Hu, 1968). This is not surprising, for n-tuples are easily handled by the digital computer. And although they may appear simple, any possible characterizer can be described as a sufficiently complex and detailed n-tuple. What we don't know is whether the n-tuple description of sufficiently powerful characterizers would avoid being overly cumbersome and ridiculously wasteful of storage space and processing time.

Programs that use n-tuples either have them designed by human beings and pre-programmed in (e.g., Andrews, Atrubin, and Hu, 1968), or randomly generate a fixed set of fixed-n-size n-tuples (e.g., Bledsoe and Browning, 1959). An interestingly simple generalization of this is the following: Let the program begin with no characterizers, but have it generate new characterizers that are as simple as possible, and only when needed. Thus the program might start by generating one 1-tuple, continue generating more 1-tuples as it finds itself continuing to choose wrong names to assign to input patterns, and at some point begin generating n+1-tuples. It further should be assessing how well each characterizer is working, by in effect conducting a running experiment that examines its successes and failures. This information should be used a) to weight the importance of this characterizer's implications in combining them for the decisions as to names to choose, b) to decide whether a characterizer is good and should therefore be used, or is bad and should therefore be discarded, to be replaced by another, and c) to gather information about general _types_ of characterizers, so that new characterizers are generated that are _similar_ in important parameter values to characterizers that have proved themselves good.

This paper describes a program that is a first approximation to this simple, but hazy, scheme of generating as few characterizers as needed, keeping them as simple as possible, but using what has been learned about characterizers to direct the generation of new characterizers, so that they will be similar in their characteristics to good characterizers that have been generated in the past.

The program has a second general purpose - to puch deeper into techniques for learning characterizers.

The basic structure of this program seems to us extremely simple - the generation, when needed, of the best new specific n-tuple of the best general type possible, and the learning of as much as possible. But when the program is described or given in detail, as in the following pages, it inevitably sounds more complex - for indeed it is more complex when forced to the level of code for a discrete digital computer. In order to get flexibility into our n-tuples so that they need not be precisely positioned and can be considered to match even though all parts do not always match, extra details must be added to the code. These in turn suggest additional learning mechanisms that will search for good values for this allowed wobbling and threshold matching.

There are also several points at which we simply evade quite subtle decisions that should be made by the program: Should the program spend more time adjusting the weights of its present set of characterizers, or should it generate one or more new characterizers? This we handle by having the program generate one new characterizer per pattern, up to a fixed maximum (also discarding characterizers found to be bad, to make room for more). When the program generates a new characterizer, should it be of the same size n, or of size n + 1? This we handle by treating n-size as Just another parameter, so that, as described below, the program will choose the n for a new tuple as a function of the goodness of the tuples of different n-size that have been generated so far. Thus n is initialized to equal 1; the program will keep tabs on the goodness of each n-size and will generate tuples with an n-size that reflects this goodness, but with some probability will occasionally generate a new tuple of size n + 1. This procedure is used for all parameters of characterizers.

Precursors

As an introduction to the structure of our program, let us consider the Bledsoe-Browning pattern recognition program (1959), which was among the first to use n-tuples randomly selected from the input grid to recognize typed or handwritten characters. For each n-tuple, the possible pattern names having the same state as the unknown input pattern are added into a comparison tally. After all tuples are considered, the name that matches the unknown pattern most closely (i.e., having the highest sum of same-state n-tuples) is chosen as the name of the input pattern.

Using the string manipulation language SNOBOL, Uhr (1969b) coded a somewhat extended version of the Bledsoe-Browning program. Uhr's short program uses weighted implications, rather than merely tallying them, and it allows varying sizes for the n-tuples and for the individual pieces of the n-tuples.

There are several weaknesses in this type of program: It does not learn, so its performance remains only as good as the n-tuple characterizers it starts with. N-tuples are rigidly positioned, and must match exactly and entirely.

A Basic N-Tuple Pattern Recognition and Learning Program

Let's try to generalize the basic n-tuple program. For example, the characterizer tuples will be looked for one part at a time, instead of all at once. Each tuple piece of the characterizer n-tuple will contain pertinent information about its expected location within the pattern grid, its size, and the specific configuration that should be found. Optionally, a tuple part will have no particular position specified, signaling the program to look anywhere (presently meaning from its current position on down) for this tuple part. If a characterizer is matched, its implied pattern names are put on a list of found implications. When the same name is implied by several characterizers, the separate weights of implications are added together. The tuples are allowed to be non-exclusive, so that grid points in important locations (such as, perhaps, the left edge of the grid) may reappear in several characterizers.

This basic program will also have the ability to learn from its experience, by comparing its chosen answer with the feedback giving the correct pattern name. If the program gave the right answer, the memory is left as is, since it produced satisfactory results. But a wrong answer calls for reweighting of implications in the characterizers whose tuple configurations were matched. The weights of implications of the wrongly chosen name are decreased, and implications of the feedback name are increased. If the answer was wrong the program will also generate a new characterizer using this wrongly named input pattern. To do this, a random n-tuple is extracted (n is chosen to reflect the distribution of weights attached to the generated values of n) from the input and assembled into a characterizer which implies the correct feedback name. Each run has an upper limit to the number of characterizers generated, to prevent saturation of memory or unnecessary slowing of processing time. Poor characterizers are discarded, making room for new ones, when the weights of all their implications fall below a minimal acceptable level.

Characterization Over Variations

Presented with only standard, non-varying instances (in a single type font, perhaps) of its repertoire of patterns, it is no great problem for a pattern recognition program to learn to recognize a set of characters. But if patterns can vary, even slightly, in position or shape from time to time, then problems mushroom. Our program tries to handle this in several ways.

Wobbly Patterns

Each part of a tuple is allowed to wobble a given horizontal distance to either side. (A somewhat more limited capability for handling vertical wobbling is the "anywhere" search mentioned previously, plus the fact that all tuple part addresses are given relative to the last position, wherever that may be. Uhr (1969b) presents programs that also allow vertical wobble.) In each characterizer tuple part there is an explicitly given wobble which tells the program just how big a hunk of the grid row it can look in for the desired configuration. This allowable wobble may vary from tuple piece to piece, as learning has indicated was needed for good performance. Thus if a desired configuration was not found within the specified wobble, but would have been found were the wobble slightly larger, then the program remembers how it almost found this characterizer. When feedback shows it chose the wrong name, if the program finds that this almost-matched characterizer would have implied the right answer, it increases the wobble allowance to improve performance .

Threshold Characterizers that Can Partially Match

Suppose that three parts of a 4-tuple were found, but the other part was not. We would like to allow use of the implications of this nearly matched characterizer even though the program did not find a perfect match. In order to do this the program uses threshold matching, where each part of a tuple has its own weight to add into the tuple's sum of "foundness." Each implication of the characterizer is preceded by a threshold requirement which must be met by the tuple sum before the implication may be merged into the list of possible pattern names. Thus one implication may require all but one part of the tuple's configuration to be found, where another implication of the same characterizer might require a perfect match of all parts.

Compound Characterizers

Besides having a primitive sort of tuple consisting of a set of 0-1 configurations to be looked for at certain points on the pattern grid, our program can also use compound characterizers, where one or more of the tuple parts is itself the name of another characterizer. The program looks in the stated position (or else 'anywhere") for the name of the desired component characterizer and treats this tuple part just as any other. Now the program must add the names of found characterizers to the input that it is processing.

Compound characterizers are currently generated from primitive characterizers that are on the list of characterizers found for this input. There must be two or more such component characterizers in order to generate a compound characterizer. When the program decides to generate a compound characterizer, it chooses the maximum number of parts to give the tuple. Then the parts are pulled off the list of characterizers found in this input, and the characterizer is assembled as initially implying only the feedback.

These compound characterizers are more general than primitive characterizers in that a more sophisticated set of pattern characteristics can be represented by one tuple. Indeed, with compound characterizers we approach a method for learning stroke or feature recognition, where primitive characterizers might represent the various primary curves and lines, and the compound characterizers could form the desired combinations of strokes to imply the various patterns. For example, if CHAR1 is the tuple describing a small open-left curve, CHAR2 is a long vertical line, and CHAR3 is a large open-left curve, then CHAR4 compounding CHAR2 and CHAR1 could imply the pattern "P" and CHAR5 coupling CHAR2 and CHAR3 could imply "D".

Parameters That Characterize Characterizers

An important part of learning in humans is generalization. In order to enable our program to, in effect, "generalize" on what it has learned and thus perform better, we have given it an expandable set of parameters or characterizer traits. For each trait (such as the number of parts, or their closeness, or their maximum horizontal spread), a value can be computed for every characterizer. With every characterizer there is associated a list of this characterizer's value for each trait. In addition we keep a common traits list of all traits and all values that have been generated and used for them. A weight is associated with each value for each trait. For example, suppose the program gives a wrong name for an input pattern on the basis of found characterizer N. Then after the implication weight of the wrong name in CHAR N is decreased, the program goes through the trait list of CHAR N and, for every trait, downweights CHAR N's value for that trait in the common trait list. In particular, if CHAR N's value for VERTSPRED (vertical point spread) is "1", then our program will look for the value "1" under the trait VERTSPRED in the general characterizer traits list, and decrease the "goodness weight" of the value "1".

When upweighting a good characterizer's trait values, the program also enters (if not already there) on each trait value list a slightly larger parameter value. In this way it broadens the range of parameter values that will be used to generate new characterizers. The value and goodness weight information in the general traits list is used when a new characterizer is generated. The program tries to generate the new characterizer tuple within the framework of what the program has already learned; currently it uses the three traits necessary to control the basic generation (tuple size, piece size, wobble) plus a fourth chosen randomly from the other possible traits (currently, these are horizontal spread, vertical spread, average closeness of parts, number of parts on the edge of the grid, and compound or primitive). A desired value for the new characterizer is chosen with a probability that reflects the weights associated with the various possible values of the trait. The program tries several times to find a randomly positioned tuple which will have this same value. Thus the program generalizes on what it has learned, in that if a value of "6" for VERTSPRED has been upweighted several times, the program may decide that this is a good value to try for in a new characterizer. (For further details, see functions TRAITWT and PROBCHOOSE and the section labeled PRIMITIVE in the precis and the code.)

The Complete Program

The preceding sections describe independent features, any or all of which could be added to a basic learning program to create a complete program. The final program containing all the features is described at the end of this section. As might be expected, the characterizers for this final program have become fairly complex. As an example,

```
CHARO = "D=0.1.1*0-2,5.3.4*01-1,/"
        "I=3*I.2,1*T.1,/"
        "P=CHAR4,/T=TRO/L=5.4/"
```

means the following:
"Description=at row 0, column 1 look in the next 1 position for the string "0", adding 2 to the tuple sum of weights on success; 5 rows down and 3 cols, over look in the next 4 positions for the string "01", adding 1 to the tuple sum on success/Implications=if sum $\geq$ 3 then imply I with weight 2; if sum $\geq$ 1 imply T with weight l/CHARO is £art of the compound CHAR4/Trait list name=TRO/Last tuple part's absolute address is row 5, col. 4/".

An outline of the program's operation follows.

Detailed Description of Program

*PRECIS FOR NTUPLE LEARNING PROGRAM.	Statement Number
INITIZE	17-31
Initialize MEMORY (can be null), any CHARacterizers and their TRait lists, and the general CHaracterizerTRAITS value list,	
PAR	32--38
Initialize PARAMeterS (values for INCrement, DECrement, INItial THRESHold, INITial WeighT, GOOD, BAD, PROBability of COMPOUND characterizer generation), EDGEDOTS to allow "wobbling" room around input pattern.	
IN	39--50
READ in the matrix, ROW by ROW, putting an EDGE on each side to allow for the maximum current WOBbLe, and maintaining the current COLumnSIZE.	
READ in the FeedBacK (marked by '***') and any PARAMeter CHANGE (marked by '$'), if given. If no more cards, go to END.	
RECOGNIZE	51--56
Initialize FOUND implication list, FOUNDCHARacterizerS list, a copy of MEMory, ROWSIZE,GRIDSIZE,	
RI	57
Blank out IMPLIST of implications whose threshold requirements are met.	
Get the next CHARacterizer and its CNumber from MEM. If no more characterizers, go to DENY.	58--59
Get the DESCRiption of CHAR, its IMPLIED patterns, and the COMPOUNDS that CHAR is part of.	60--62
R3	63--65
Pick off THIS piece, its WeighT, and its POSition from DESCR. If no more parts, go to R2.	
RR2	66--69
If POS includes relativeDROW and DCOL numbers and a MASK size, compute the absolute DROW location and go to RR6 to look for THIS positioned piece.	
RR3	70--72
Otherwise look 'ANYWHERE' for THIS, starting at the current Begin ROW. If find THIS, go to R4R.	

	Statement Number
R3R	73
If the ANYWHERE search fails, make the next tuple part apply ANYWHERE, too (by erasing its position). Go to R3.	
R4R	74
Set this Row as the Begin ROW for future search.	
If THIS is followed by BCOLumn and WeiGhT information at the RIGHT, we are working with a compound characterizer; go to RR4.	75
Otherwise compute the BeginCOL for future search and go to R3.	76
RR4	77
If the WeiGhT of THIS piece meets the threshold WeighT requirement for this compound characterizer part, add the WeiGhT to the SUM of implications. Go to R3.	
RR6	78-80
Finish computing the absolute DCOLumn position to look at for THIS piece. Look first for THIS as a compound tuple part with its DColumn and WeiGhT information; if don't find it, go to RR7.	
If DC was within the WobBle allowance, reset BROW and BCOL, add WGT to SUM, and go to R3. Otherwise, if a little bigger WobBLe allowance would have given a match, record this NEARMISS. Go to RR8.	81-84
RR7	85-91
Look for THIS positioned primitive tuple part. If no match, see if a little more wobble gives a NEAR-MISS.	
RR8	92-93
Set new BROW and BCOL for next search and go to R3.	
R2	94-95
Make IMPLIST of implications whose THReshold weight requirements were met.	
R5A	96-97
Record NEARMISSes on ALMOSTFOUND characterizers list.	
R5	98
If IMPLIST is not empty put on FOUNDCHARacterizerS this CHAR and its information.	
If CHAR is part of a larger compound, mark it found in the input matrix and put the COMPOUNDS on MEM to look at later. Go to RI.	99-104

	Statement Number
DENY	105-106
Erase from FOUNDCHARS any characterizers whose implications are DENIED by a compound characterizer.	
IMPLY	107-111
Merge all the remaining implications from FOUNDCHARS into a FOUND list.	
CHOOSE	112-116
Choose as HINAME the NAME on FOUND with the Highest WeighT.	
OUT	117-122
PRINT out HINAME.	
If there was no FeedBacK or if HINAME was right, go to IN. Otherwise, answer was wrong.	
REWEIGHT	123-124
Pick off the next CHARacterizer from the FCopy of FOUNDCHARS. If no more CHARs, go to ADJust wobbles.	
If the wrong HINAME is IMPLIED by this CHAR, downweight the implication or erase it if the reduced weight is BAD.	125-130
C7	131
For each trait, give this bad CHARacterizer's trait value a DECrement on the main CHaracter TRAITS list.	
If all implications are erased for this CHAR, erase it from MEMORY and go to REWEIGHT.	132-134
C4	135-136
If FeedBacK was not IMPLIED by this found CHAR, add it to implications and go to REWEIGHT. Otherwise,	
C5	137-138
upweight the implication of FBK in CHAR. For each trait, give this good CHAR'S value an INCrement on the main CHTRAITS list. Go to REWEIGHT.	
ADJUST	139-140
Pick off the next CHARacterizer which was ALMOSTFOUND. If no more, go to GENERATE a new characterizer.	
If this CHARacterizer IMPUED FBK but not the wrong HINAME, enlarge the mask for the parts WHICH would have given a match.	141-150
Make sure this CHARacterizer*s TRait list contains the maximum wobble value of its parts.	151-156

	Statement Number
GENERATE	157-158
If there are already enough characterizes (i.e. no more TOGENerate), go to IN.	
Get the number of the new characterizer and decide if it should be compound. If not, go to PRIMITIVE generation.	159-160
COMPOUND	161-169
Choose the number of PARTS the new characterizer should have. Make an FCopy of FOUNDCHARS to choose from.	
CGI	170-171
Get the next found CHaracterizer from FC. If no more, go to CG2. Keep TUP count of how many parts are got from FC.	
Get the LASTPART location of CH and insert it in order into the DESCRiption of the new compound characterizer.	172-173
Add to the new IMPlication LIST the denial of this component CHaracterizer's implications.	174
Keep a list of the PRIMitive PARTS in order to later insert this new characterizer in their COMPOUNDS lists.	175
Keep the largest WoBble value of any part of the compound stored as WBL.	176-177
Keep the largest Piece size of any part of the compound stored as PC.	178-180
CG2	181-182
If there were less than 2 components in FC, go to PRIMITIVE.	
CG3	183-191
Assemble the final DESCRiption with relative positions, noting the LASTPART.	
Initialize TRait list for new compound characterizer. Assemble the characterizer.	192-193
CG5	194-196
Mark each component characterizer as part of this compound characterizer. Go to G6.	
PRIMITIVE	197-199
Make a rearranged COPYTRAITS of CHTRAITS so as to cycle through traits used to Influence primitive tuple generation.	
According to the value probabilities in COPYTRAITS, choose the necessary values for characterizer generation (TUPle size, PieCe size, WobBLe).	200-202
Initialize the new TRait list with these values.	203
Get a desired TRYVALue for another TRait. Will TRY to generate a tuple with the same VALue.	204-207
GENTUP	208-210
Create a relatively-ordered random TUP-tuple ordered by rows, and calculate its VALue for the chosen TRait.	
If VAL equals the TRYVAL, or if 5 TRYs failed, go to G4. Otherwise go to GENTUP and TRY again.	
G4	211-212
Assemble the primitive characterizer.	
G6	213-216
Complete the new characterizer's TRait list, computing values for all other traits.	
Add the new characterizer to MEMORY. Go to IN.	217-218
♦Begin routines to calculate the various trait values for the relatively-addressed tuple in DESCR.	
HOROSPRED	219-227
calculates the maximum number of columns between the leftmost and rightmost parts of the tuple.	
VERTSPRED	228-231
calculates the maximum number of rows between the topmost and bottommost parts of the tuple.	
GRIDEDGE	232-242
calculates the number of parts in the tuple which lie on the edge of the input pattern. VAL is normalized over 10.	
PROXIM	243-256
calculates the sum of absolute differences between corresponding digits in all possible pairs of tuple parts.	
COMPOUND	257-259
returns 'YES' or 'NO', according to whether the tuple is a compound or primitive characterizer.	

Discussion

This paper briefly describes the various features of-our program. It then gives a detailed flow-chart-like "precis" that refers by number to the actual program statements being described. The program itself is given in the Appendix. Thus the reader can examine exactly what has been done to implement any of the aspects of the program about which he is curious. This seems to us of crucial importance: if the program can be used to document itself there is no need for lengthy and usually misleading descriptions and discussions.

The program listing is too long and complex to be followed with ease, even by someone who knows SNOBOL; but it should give an idea of what's going on to the casual observer, and those parts in which the reader is interested enough to make some effort should become understandable. SNOBOL is a very simple language in its basic conception, for its programs are built up from sets of production and replacement statements (of the sort "Let A = B; Look for C on A and, if it's found, replace it by B), tied together by labels and gotos. A brief description of SNOBOL is given in the Appendix.

This program was written to examine whether a wide variety of learning methods could be implemented together in a single pattern recognition program. Using the language SNOBOL allowed us to code a relatively powerful, yet short, program. However, the program runs too slowly to make extensive tests of its abilities to learn and achieve interesting asymptotic performance levels. We therefore give only a brief listing of a short run, to indicate that the program works, and that it at least begins to learn. The program will be recoded in a faster language if we decide to make more extensive tests.

Further developments might be to have the program try to learn good weights of characterizer tuple parts and the thresholds required to imply a pattern name. We would also like it to generate new parameters with which to characterize its characterizers (see Uhr, 1969b).

Summary

The program described in this paper attempts to combine a very simple basic pattern recognition scheme with a wide variety of powerful learning mechanisms. The program attempts 1) to generate its own n-tuple characterizers as needed, and to adjust their weights as a function of feedback, 2) to decide what type of characterizer to generate, and 3) to learn what are good general characteristics of characterizers. It can further decide 4) whether and how to modify any particular characterizer that it is evaluating. These decisions are all made within the framework of a program that tries to recognize patterns with as small a set of characterizers that are as simple as possible. It therefore starts out with no characterizers, and generates other characterizers which are as simple as it has been able to get away with and which fall within the range of what the program conjectures to be optimal values for the characteristics of characterizers. In terms of characterizer size, this means the program starts out generating 1-tuples and then, to the extent that feedback indicates that it must improve upon its performance, 2-tuples, 3-tuples, and n+1-tuples.

Acknowledgements

This research has been supported in part by NIH grant MH-12266 and NSF grant GP-7069.

Bibliography

1. Andrews, D.R., Atrubin, A.J., and Hu, K. C, The IBM 1975 optical page reader: Part III: Recognition logic development. IBM I. Research and Development, 1968, 12, 364-372.
2. Bledsoe, W.W. and Browning, I., Pattern recognition and reading by machine. Proc. Eastern Joint Comp. Conf., 1959, 225-232.
3. Prather, Rebecca and Uhr, L., Discovery and learning techniques for pattern recognition. Proc. 19th Annual Meeting of the ACM, 1964.
4. Uhr, L., Pattern recognition computers as models for form perception. Psychol Bull., 1963, 60., 40-73.
5. Uhr, L. & Vossler, C, A pattern recognition program that generates, evaluates, and adjusts its own operators. Proc. Western Joint Computer Conf., 1961, 555-569.
6. Uhr, L., A tutorial description of pattern recognition programs. (Submitted for publication, 1969a).
7. Uhr, L,, Pattern Recognition, Problem-Solving and Learning. 1969b (In preparation) .

Appendix

A Brief Description of SNOBOL

SNOBOL is a "pattern matching" language that turns out to be quite convenient for handling list structures and networks of information, using push-down stacks, indirection, and recursive programming. Its syntax is extremely simple, as follows:

SNOBOL programs are built up of two basic types of statements:

1) Assignment statements that assign a name to a pattern of strings,

```
e.g.    DESCRIPTION = '001100'
        CHARACTERIZER = DESCRIPTION ' = '
          IMPliEDS  '/'
```

2) Replacement statements that find patterns on a string and (if they are found) replace them by another pattern,

e.g. **FOUND THIS '-' *SUM* ',' =**
THIS '-' SUM + '1' ','

These statements have several components: a) the "name" of the string to be processed, b) the "pattern" which is a sequence of 1) "names" (e.g. IMPUEDS, FOUND, THIS) which refer to and stand for their contents, 2) "literals (e.g. "'='") which stand for themselves, and 3) "variable names" (e.g. "*SUM*"), which are assigned contents during the execution of the statement, if the program succeeds in matching the pattern somewhere in the named string. A variable name can be subscripted with a number that fixes its length (e.g. ***THIS/'2'* or** ♦THIS/SIZE* where SIZE contains an integer).

In the two examples of assignment statements above, DESCRIPTION is made the name of the string whose literal contents are '001100', and then 001100 is put at the beginning of the string named CHARACTERIZER, since the name DESCRIPTION refers to its <u>contents</u>. If another assignment statement, **001100 = 'EDGE ',** were coded, then the indirect reference symbol dollar-sign ($) preceding the name $ DESCRIPTION would put EDGE, not 001100, on CHARACTERIZER.

The end of the pattern-to-be-matched is marked by the equal sign **(=)** <u>without</u> quotes around it, and this also marks the beginning of the replacement pattern. The first string of a statement is always the name; all subsequent strings up to the equal sign form the pattern-to-be-matched, and all strings after the equal sign form the replacement pattern (if the left-hand pattern succeeded).

A statement can be surrounded by "labels" and 'tjotos" which control the flow of the program. A 'label" is a string that always begins in column 1. A "goto" comes after the statement, is signaled by a slash, and is of the form /(INPUT) or /S(INPUT) or /F(INPUT) or /S(INPUT)F(PROCESS), where S means transfer on success, F means transfer on failure, and no letter after the slash means unconditional transfer. (The goto must, of course, always refer to a label.) When there are no gotos, the program goes to the next statement in sequence.

Arithmetic is performed within these statements by using **+,-,*,** and / (and ** for exponentiation). Numbers must be referred to either as literals or as contents of lists (as in SUM + '1' above, which will add one to the number stored in SUM). A number of built-in functions can be used to test for inequalities: .GT(A,B), .LT(A,B),.GE(A,B), .LE(A,B), .EQ(A,B) and EQUALS(A,B) (which is string-matching equality). The command ".READ" will read in one data card, and ".PRINT =" will print out the pattern that follows.

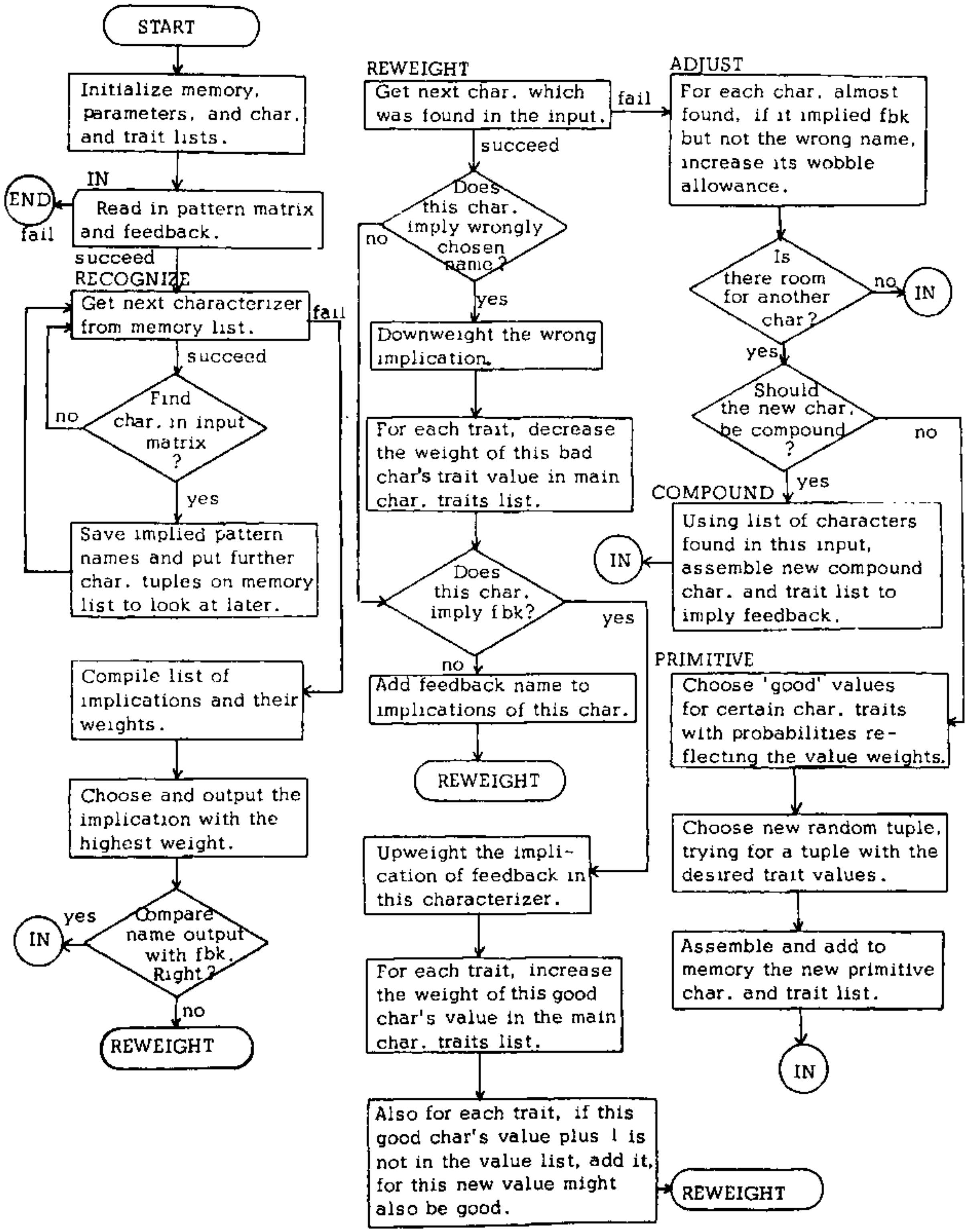
An asterisk (*) in column 1 denotes a comment card, which the compiler will ignore. A period (.) in column 1 indicates that this card continues the statement on the preceding card (statements can use only 72 columns, whereas the data cards that follow the program can use all 80 columns). A program ends with an END card (END starts in column 1) that also contains the label of the first statement to be executed.

The basic pattern match goes from left to right. The compiler looks for the next match of each element of the pattern (ignoring variable names, which will be assigned to the strings that lie between the matched elements - the literals and names with contents). If no match is found, it backtracks to break the assignment of the previously matched element, and looks for its next match, continuing this until either the last element matches or the first element fails. Success or failure in the gotos is contingent upon either this match or one of the functions. The programmer can define and code his own functions, and do a number of other powerful things not discussed here.

A simple program for information retrieval follows.

```
♦EXAMPLE PROGRAM.  A SIMPLE PROGRAM TO
♦DO 'INFORMATION RETRIEVAL" FOLLOWS:
GO     DOCUMENTS = 'RIVERS=D1,D3,D8,/'     MI
       'LAKES=D3, D5, /SPAIN= D3, D8, DI 1,'
       •D17,/'
IN     .READ *QUERY* '    7F(END)           1
ASK    QUERY ♦DESCRIPTORS ','=/F(IN)        2
       DOCUMENTS DESCRIPTOR '='             3
       ♦PERTINENT*  '/'  /F(ASK)
       .PRINT = DESCRIPTOR ' IS '           4
       'DISCUSSED IN ' PERTINENT/(ASK)
END    GO
RIVERS, SPAIN, MOUNTAINS,                   DI
((Query to be input, on data card))
♦PRECIS - AN ENGLISH DESCRIPTION OF
♦ABOVE INFORMATION RETRIEVAL PROGRAM.
GO     Let DOCUMENTS contain the de-        MI
       scriptors, followed by pertinent
       documents.
IN     READ in the next QUERY (which is     1
       a list of descriptors)
ASK    Get the next DESCRIPTOR from the     2
       QUERY.  (If no more, Fail to ASK.)
       From DOCUMENTS, get PERTINENT        3
       ones if the DESCRIPTOR is found.
       PRINT out the DESCRIPTOR and the     4
       PERTINENT documents.
END    GO
```

Figure 1. Program Flowchart



```
BEGIN    MODE(#DUMP#)                                                     1
         OUTPUT(#PRINT#,#61#)                                             2
         READ(#READ#,#60#)                                                3
         DEFINE(#ROWORDER(PT,LIST)#,#ROWORDER#,#ROW,COL,R,C,LEFT#)        4
         DEFINE(#TRAITWT(TN,UX)#,#T1#)                                    5
         DEFINE(#RANTUP(LIST)#,#RT#)                                      6
         DEFINE(#PROBCHOOSE(T)#,#PROBCH#)                                 7
         DEFINE(#ABSVAL(K)#,#ABSVAL#)                                     8
         DEFINE(#TUPLEDESCR(T,P)#,#D1#,#ROW,COL#)                         9
         DEFINE(#LARGEST(TR)#,#LARGEST#)                                 10
         DEFINE(#SECS(T)#,#SEC#,#H,M,S#)                                 11
         DEFINE(#PRINT75(C,S)#,#PRINT75#,#S1#)                           12
         READ *TLIM* #    #                                              13
         PRINT = # TLIM = # TLIM                                         14
         BEGINTIME = SECS(TIME())                                        15
         PRINT = # BEGINTIME = # BEGINTIME                               16
         READ *FLAG* #    # /F(END)S($FLAG)                              17
* READ CARDS WITH INFO SAVED FROM LAST RUN
READFILE READ *INP* /F(PAR4)                                             18
         INP #$$$# /S(PAR4)                                              19
         INP *NAME* #=# *INP*                                            20
         INPUT = INPUT INP                                               21
         INPUT *INP* #$$# /F(READFILE)                                   22
         BLANK *INPUT*                                                   23
         $NAME = NAME INP /(READFILE)                                    24
INITIZE  MEMORY =                                                        25
         N = #0#                                                         26
         CHTRAITS = #PIECE(,1-1,)/TOPSIZE(,2-1,)/$/HORUSPRED(,3-1,)/#    27
.        #VERTSPRED(,3-1,)/GRIDEDGE(,5-1,)/PROXIM(,4-1,)/#
.        #COMPOUND(,NO-1,)/WOBBLE(,0-1,)/#
         TOGEN = #2)#                                                    28
PAR4     PRINT = #INITIAL MEMORY#                                        29
PAR3     KN = .LT(KN,N) KN + #1# /F(PAR)                                 30
         PRINT75((#CHAR# KN),$(#CHAR# KN)) /(PAR3)                       31
PAR      PARAMS =              #INC(1)/DEC(-1)/INITHRESH(2)/INITWT(5)/#  32
.        #GOOD(10)/BAD(0)/PROBCOMPOUND(2)/#
* INITIALIZE PARAMS TO THEIR VALS
         COPY = PARAMS                                                   33
PAR1     COPY *X* #(# *Y* #)/# = /F(PAR2)                                34
         $X = Y /(PAR1)                                                  35
PAR2     EDGEDOTS = #..........#    /(IN)                                36
PCH      PARAMCHANGE *PAR* #=# *X*                                       37
         $PAR = X                                                        38
IN       COLSIZE = #0#                                                   39
         PRINT = #INPUT PATTERN#                                         40
* IF ALMOST TO END OF TIME LIMIT SAVE INFO LEARNED
         X = SECS(TIME()) - BEGINTIME                                    41
         .GE(X,TLIM) /S(KEEPFILE)                                        42
* FIND LARGEST POSSIBLE WOBBLE TO PUT ON EDGES
         WOBL = LARGEST(#WOBBLE#)                                        43
         EDGEDOTS *EDGE/WOBL*                                            44
I1       READ *INROW* #    # /F(KEEPFILE)                                45
         INROW #****# *FRK* /S(RECOGNIZE)                                46
* CAN READ IN NEW PARAM VALUE WITH STMT OF FORM )PARAMNAME=#)NEWVALUE=#
         INROW #)# *PARAMCHANGE* /S(PCH)                                 47
         $(#ROW# COLSIZE) = EDGE INROW EDGE                              48
```

```
          PRINT = #      # $(#ROW# COLSIZE)                                          49
          COLSIZE = COLSIZE + #1# /(I1)                                              50
RECOGNIZE FOUND = #.UNKNOWN=0.#                                                      51
          PRINT = #FEEDBACK = # FBK                                                  52
          MEM = MEMORY                                                               53
          BLANK *FOUNDCHARS*                                                         54
          ROWSIZE = SIZE(ROW1) - #2# * SIZE(EDGE)                                    55
          GRIDSIZE = ROWSIZE * COLSIZE                                               56
R1        BLANK *IMPLIST*                                                            57
          MEM *X/#4#* *CN* #.#                                                       58
          MEM *CHAR* #.# = /F(DENY)                                                  59
          $CHAR #D=# *DESCR* #/I=# *IMPLIED* #/P=# *COMPOUNDS* #/#                   60
* ACCUMULATE SUM OF IMPLICS. OF EACH PART
          SUM = #0#                                                                  61
          BLANK *BROW* *BCOL* *NEARMISS* *WHICH*                                     62
R3        DESCR *POS* #*# *THIS* #-# *WT* #.# = /F(R2)                               63
          BLANK *LS* *RS*                                                            64
          WHICH = WHICH + #1#                                                        65
RR2       POS *DROW* #.# *DCOL* #.# *MASK* /S(RR5)                                   66
          CROW = #ANYWHERE#  /(RA5)                                                  67
RR5       CROW = DROW + BROW /S(RR6)                                                 68
* ADDITION FAILS WHEN DROW = #ANYWHERE#.  START LOOKING IN ROW BROW
RA5       R = BROW - #1#                                                             69
* #ANYWHERE# SEARCH
RR3       .LT(R,COLSIZE - #1#) /F(R3R)                                               70
          R = R + #1#                                                                71
          $(#ROW# R) *LEFT* THIS *RIGHT* /F(RR3)S(R4R)                               72
* SINCE THIS #ANYWHERE# SEARCH FAILED MAKE NEXT TUPLE PART APPLY ANYWHER
R3R       DESCR *ROWCOL* #*# = #*# /(R3)                                             73
* FOUND THIS PART. SET THIS ROW AS NEW BEGINROW FOR FUTURE SEARCH
R4R       BROW = R                                                                   74
          RIGHT ANCHOR() #/(# *BCOL* #-# *WGT* #)# /S(RR4)                           75
          BCOL = SIZE(LEFT) - SIZE(EDGE) /(R3)                                       76
* SEE IF WGT OF THIS FOUND PRIMITIVE CHARACTERIZER IS ABOVE THE
* THRESHOLD NEEDED AS PART OF THE COMPOUND CHAR
RR4       SUM = .GE(WGT,WT) SUM + WGT /(R3)                                          77
* LOOK FOR (RELATIVELY) POSITIONED THIS PIECE
RR6       CCOL = DCOL + BCOL                                                         78
* FIND CURRENT WOBBLE SIZE FOR THIS CHARACTERIZER
          $(#TR# CN) #WOBBLE(# *WBL* #)#                                             79
* LOOK FOR COMPOUND TUPLE PART WITH ITS INFO
          $(#ROW# DROW) THIS #/(# *DC* #-# *WGT* #)# /F(RR7)                         80
          CCOL = .GE(WBL,ABSVAL(DC - DCOL)) DC /S(RR9)                               81
* SEE IF MORE WOBBLE WOULD GIVE MATCH
          NEARMISS = .GE(WBL + #1#, ABSVAL(DC - DCOL)) NEARMISS WHICH #.#            82
.         /(RR8)
* SET THIS AS NEW STARTING PT. FOR FUTURE SEARCH
RR9       BROW = DROW                                                                83
          BCOL = DCOL /(RR4)                                                         84
* IF WONT GO OFF EDGES OF GRID. LOOK AT LEFT + RIGHT SIDES OF MASK TOO
RR7       LS = .GT(DCOL + SIZE(EDGE),#0#) #1#                                        85
          RS = .LT(DCOL + MASK + SIZE(EDGE),SIZE(ROW1)) #1#                          86
* LOOK FOR POSITIONED PRIMITIVE TUPLE PART
RR11      $(#ROW# DROW) ANCHOR() *LEFT/(DCOL + SIZE(EDGE) - LS)*                     87
.         *L/LS* *GLANCE/MASK* *R/RS*  /F(RR8)
          GLANCE THIS /S(RR10)                                                       88
* SEE IF MORE WOBBLE WOULD GIVE MATCH
          (L GLANCE R) THIS /F(RR8)                                                  89
          NEARMISS = NEARMISS WHICH #.# /(RR8)                                       90
RR10      SUM = SUM + WT                                                             91
* SET NEW STARTING POINT FOR SEARCH (EVEN IF DID NOT FIND THIS PART)
RR8       BROW = .GE(DROW,#0#) DROW                                                  92
          BCOL = .GE(DCOL,#0#) DCOL /(R3)                                            93
* END OF THIS CHARACTERIZER
* MAKE IMPLIST OF IMPLICS. WHOSE THRESHOLD WT. REQUIREMENTS WERE MET
```

```
R2        IMPLIED *THR* #*# *IMP* #,# = /F(R5A)                              94
          IMPLIST = .GE(SUM,THR) IMPLIST IMP #,# /(R2)                       95
* SEE IF HAD ANY NEARMISSES
R5A       NEARMISS *X/#1#* /F(R5)                                            96
          ALMOSTFOUND = .GT(SIZE(IMPLIST),#1#) ALMOSTFOUND CHAR #*#          97
.         NEARMISS #*,# IMPLIST #/#
* IF ANY PART MATCHED (AND GOT ON IMPLIST) PUT ON FOUNDCHARS
* THIS CHAR AND ITS INFO
R5        FOUNDCHARS = .GT(SIZE(IMPLIST),#1#) FOUNDCHARS CHAR #,# CN         98
.         #*# SUM #=,# IMPLIST #/# /F(R1)
* IF THIS CHAR IS PART OF A LARGER COMPOUND, MARK IT FOUND IN
* CURRENT POSITION ROW OF INPUT
          COMPOUNDS *X/#1#* /F(R1)                                           99
          $(#ROW# BROW) = $(#ROW# BROW) CHAR #/(# BCOL #=# SUM #)#           100
* IF NOT ALREADY THERE ADD ONTO MEM THE CHARACTERIZERS OF WHICHTHIS CHAR
* IS A COMPOUND PART TO LOOK AT LATER
R4        COMPOUNDS *CH* #,# = /F(R1)                                        101
          MEM CH #,# /S(R4)                                                  102
          FOUNDCHARS CH #,# /S(R4)                                           103
          MEM = MEM CH #,# /(R4)                                             104
* ERASE ANY DENIED CHARS
DENY      FOUNDCHARS #$# *DENIED* #,# = /F(IMPLY)                            105
          FOUNDCHARS DENIED #,# *X* #/# = /(DENY)                            106
IMPLY     FC = FOUNDCHARS                                                    107
R6        FC *CH* #=,# *IMPLIED* #/# = /F(CHOOSE)                            108
R7        IMPLIED *NAME* #,# *WT* #,# = /F(R6)                               109
          FOUND #,# NAME #=# *SUM* #,# = #,# NAME #=# SUM + WT #,# /S(R7)     110
          FOUND = #,# NAME #=# WT FOUND /(R7)                                111
CHOOSE    FOUND #,# *HINAME* #=# *HIWT* #,# =                                112
C1        FOUND *NAME* #=# *WT* #,# = /F(OUT)                                113
          .GT(WT,HIWT) /F(C1)                                                114
          HINAME = NAME                                                      115
          HIWT = WT /(C1)                                                    116
OUT       PRINT = #IT IS # HINAME                                            117
* EVALUATE ANSWER
          FBK *X/#1#* /F(IN)                                                 118
          EQUALS(FBK,HINAME) /S(IN)                                          119
* ANSWER WAS WRONG
          FC = FOUNDCHARS                                                    120
          PRINT = #REWEIGHTED CHARACTERIZERS#                                121
          CHAR =                                                             122
REWEIGHT  .GT(SIZE(CHAR),#1#) PRINT75(CHAR,$CHAR)                            123
          FC *CHAR* #,# *CN* #*# *SUM* #=# *IMPLIED* #/# = /F(ADJ)           124
* SEE IF HINAME WAS AMONG IMPLICS
          IMPLIED #,# HINAME #,# *W1* #,# /F(C4)                             125
* DOWNWEIGHT THE IMPLIC. OF HINAME
          WT = .GT(WT,BAD + #1#) WT - #1# /S(C6)                             126
* ERASE HINAME, BECAUSE IMPLIC. WT. WOULD NOW BE $ BAD
          $CHAR #I=# *X* #*# HINAME #,# *WT* #,# = #I=#                      127
* IF THERE IS ANOTHER IMPLIC. IN X, PICK THRESH WT OF HINAME OFF END
* AND REPLACE X AFTER #I=#
          X .GT(SIZE(X),#3#) *LEFT/(SIZE(X) - #3#)* *Y* #,# *THR*            128
.         = LEFT Y #,# /F(C7)
          $CHAR #I=# = #I=# X /(C7)                                          129
* REPLACE HINAME IMPLIC. WT WITH REDUCED WT.
C6        $CHAR #I=# *X* #*# HINAME #,# *OLDWT* #,# =                        130
.         #I=# X #*# HINAME #,# WT #,#
* FOR EACH TRAIT, GIVE THIS BAD CHARACTERIZERS VALUE A DECREMENT
* ON THE MAIN EVALUATION LIST
C7        TRAITWT(CN,DEC)                                                    131
* IF NO MORE IMPLICS FOR THIS TUPLE ERASE CHAR FROM MEMORY
          $CHAR *X* #I=/# *Y* = /F(C4)                                       132
          TOGEN = TOGEN + #1#                                                133
```

```
         MEMORY CHAR ≠,≠ = /(REWEIGHT)                                    134
* IF FBK WAS NOT AMONG IMPLICS, ADD IT
C4       IMPLIED ≠.≠ FBK ≠.≠ /S(C5)                                        135
        $CHAR ≠I=≠ = ≠I=≠ INITHRESH ≠*≠ FBK ≠.≠ INITWT ≠;≠ /(REWEIGHT)    136
* UPWEIGHT THE IMPLIC. OF FBK
C5      $CHAR ≠I=≠ *X* FBK ≠.≠ *WT* ≠,≠ = ≠I=≠ X FBK ≠.≠ WT + ≠1≠ ≠,≠      137
* UPWEIGHT THIS CHARS VALUE IN MAIN CHARTRAITS LIST
                         TRAITWT(CN,INC) /(REWEIGHT)                       138
ADJ      PRINT = ≠WOBBLE-ADJUSTED CHARACTERIZERS≠                          139
* INCREASE ANY WOBBLE THAT WOULD HAVE IMPROVED PERFORMANCE
ADJUST   ALMOSTFOUND *CHAR* ≠-≠ *WHICH* ≠*≠ *IMPLIED* ≠/≠ = /F(GENERATE)   140
* SEE IF THIS CHAR IMPLIED FBK BUT NOT WRONG HINAME
         IMPLIED ≠,≠ FBK ≠-≠ /F(ADJUST)                                    141
         IMPLIED ≠,≠ HINAME ≠-≠ /S(ADJUST)                                 142
* INCREASE WBL FOR THE WHICH PARTS OF CHAR, SO WOULD HAVE SUCCEEDED
AD1      BLANK *LEFT*                                                      143
         MAXWBL = ≠0≠                                                      144
         $CHAR ≠D=≠ *DESCR* ≠/≠ *X* ≠/T=≠ *TR* ≠/≠                         145
AD3      WHICH *W* ≠.≠ = /F(REPLACE)                                       146
AD2      DESCR *PART* ≠.≠ =                                                147
         W = W - ≠1≠                                                       148
         LEFT = .GT(W,≠0≠) LEFT PART ≠,≠ /S(AD2)                           149
* INCREASE WOBBLE OF THIS PART BY ADDING 2 TO MASK SIZE
         PART *ROW* ≠.≠ *COL* ≠.≠ *MASK* ≠*≠ *THIS* ≠-≠ =                  150
.        ROW ≠.≠ COL - ≠1≠ ≠.≠ MASK + ≠2≠ ≠*≠ THIS ≠-≠
* KEEP LARGEST WOBBLE OF THIS CHAR IN MAXWBL
         THISWBL = (MASK + ≠2≠ - SIZE(THIS)) / ≠2≠                         151
         MAXWBL = .GT(THISWBL,MAXWBL) THISWBL                              152
         DESCR = LEFT PART ≠,≠ DESCR /(AD3)                                153
* PUT ADJUSTED DESCRIPTION BACK IN CHAR
REPLACE  $CHAR ≠D=≠ *OLDDESCR* ≠/≠ = ≠D=≠ DESCR ≠/≠                         154
* MAKE SURE TR CONTAINS MAX WOBBLE OF CHARS PARTS
         $TR ≠WOBBLE(≠  *WBL* ≠)≠ = .GT(MAXWBL,WBL) ≠WOBBLE(≠ MAXWBL ≠)≠   155
         PRINT75(CHAR,$CHAR) /(ADJUST)                                     156
GENERATE .GT(TOGEN,≠0≠) /F(IN)                                             157
         TOGEN = TOGEN - ≠1≠                                               158
* GET NUMBER OF NEW CHARACTERIZER
         N = N + ≠1≠                                                       159
* DECIDE WHETHER TO GENERATE COMPOUND OR PRIMITIVE CHARACTERIZER
         .LE(.RANF(≠10≠),PROBCOMPOUND) /F(PRIMITIVE)                       160
* COMPOUND GENERATION
COMPOWND COPYTRAITS = CHTRAITS                                             161
         COPYTRAITS ≠WOBBLE(≠ *X* ≠)/≠ =                                   162
         COPYTRAITS ≠PIECE(≠ *X* ≠)/≠ =                                    163
* ERASE ABOVE SO WONT TRY TO CALCULATE TRAIT VAL LATER
         PARTS = PROBCHOOSE(≠TUPSIZE≠)                                     164
         PARTS = .LE(PARTS,≠1≠) ≠2≠                                        165
         FC = FOUNDCHARS                                                   166
         BLANK *IMPLIST* *DESCR*      *TUP* *PC* *PREROW*                  167
         BLANK *PRECOL* *PRIMPARTS*                                        168
         WBL = ≠0≠                                                         169
CG1      FC *CH* ≠.≠ *CN* ≠*≠ *X* ≠/≠ = /F(CG2)                            170
         TUP = TUP + ≠1≠                                                   171
         $CH ≠L=≠ *LASTPART* ≠/≠                                           172
         DESCR = ROWORDER(LASTPART,DESCR)                                  173
         IMPLIST = IMPLIST INITHRESH ≠*$≠ CH ≠,≠                           174
         PRIMPARTS = PRIMPARTS CH ≠,≠                                      175
* KEEP LARGEST WOBBLE VALUE FOR THIS COMPOUND CHARACTERIZER
         $(≠TR≠ CN) ≠WOBBLE(≠ *WB* ≠)≠                                     176
         WBL = .GT(WB,WBL) WB                                              177
* KEEP LARGEST TUPLE SIZE OF COMPONENT PARTS AS PIECE SIZE FOR THIS
* COMPOUND
         $(≠TR≠ CN) ≠TUPSIZE(≠ *P* ≠)≠                                     178
         PC = .GT(P,PC) P                                                  179
```

```
           PARTS = .GT(PARTS,#1#) PARTS - #1# /S(CG1)                    180
 CG2       .GE(TUP,#2#) /F(PRIMITIVE)                                    181
           DESCR *PLIST* =                                               182
* CHANGE DESCR TO RELATIVE POSITIONS
 CG3       PLIST *ROW* #,# *COL* #,# *CH* #,# = /F(CG4)                  183
           X1 = ROW - PREROW                                             184
           X2 = COL - PRECOL                                             185
           X3 = #2# * WBL                                                186
           X4 = X3 + SIZE(CH)                                            187
           DESCR = DESCR X1 #,# X2 #,# X4  #*# CH #=1,#                  188
           PREROW = ROW                                                  189
           PRECOL = COL /(CG3)                                           190
 CG4       LASTPART = ROW #,# COL                                        191
* INITIALIZE TRAIT LIST
           $(#TR# N) = #TUPSIZE(# TUP #),PIECE(# PC                      192
.          #),WOBBLE(# WBL #),#
* ASSEMBLE COMPOUND CHARACTERIZER
           $(#CHAR# N ) = #D=# DESCR #/I=# INITHRESH #*# FBK #,#          193
.          INITWT #,# IMPLIST #/P=/T=TR# N #/L=# LASTPART #/#
* MARK EACH PRIMITIVE CHAR COMPONENT PART OF THIS COMPOUND CH
 CG5       PRIMPARTS *CH* #,# = /F(CG6)                                  194
           $CH #P=# = #P=CHAR# N #,# /(CG5)                              195
 CG6       RET = #G5# /(G6)                                              196
* PRIMITIVE GENERATION
* MOVE $ MARKER IN CHTRAITS SO AS TO CYCLE THRU TRAITS
* ATTEMPTING TO GENERATE WITHIN
PRIMITIVE CHTRAITS *X* #/$/# *Y* #/# *Z* = X #/# Y #/$/# Z /S(G1)        197
           CHTRAITS *X* #/# *Y* #/$/# *Z* = X #/$/# Y #/# Z              198
 G1        COPYTRAITS = Z X #/# Y #/#                                    199
* FIRST CHOOSE TRAITS WHICH ARE NECESSARY INFO FOR ANY TUPLE
           TUP = PROBCHOOSE(#TUPSIZE#)                                   200
           PC = PROBCHOOSE(#PIECE#)                                      201
           WBL = PROBCHOOSE(#WOBBLE#)                                    202
* INITIALIZE TRAIT LIST FOR NEW CHARACTERIZER
           $(#TR# N) = #TUPSIZE(# TUP #),PIECE(# PC #),WOBBLE(# WBL #),#  203
* GET ADDIT. TRAIT VALUE TO TRY FOR
           COPYTRAITS *TR* #(#                                           204
           TRYVAL = PROBCHOOSE(TR)                                       205
           TRY = #0#                                                     206
           RET = #G3#                                                    207
* NOW GENERATE TUPLE USING TUPSIZE - PIECE SPECIFICATIONS
 GENTUP    TRY = .LE(TRY,#4#) TRY + #1# /F(G4)                           208
* CREATE RANDOM TUPLE ORDERED BY ROWS
           DESCR = TUPLEDESCR(TUP,PC)      /($TR)                        209
* WILL RETURN FROM TRAIT ROUTINE TO WHATEVER ADDRESS IS IN RET
* SEE IF THIS TUPLE SATISFIED THE DESIRED TRAIT VAL
 G3        EQUALS(VAL,TRYVAL) /F(GENTUP)                                 210
* TUPLE HAD DESIRED VALUE OR ELSE 5 TRIES FAILED AND THIS TUPLE WILL
* BE ACCEPTED
 G4        RET = #G5#                                                    211
* ASSEMBLE CHARACTERIZER
           $(#CHAR# N) = #D=# DESCR #/I=# INITHRESH #*# FBK #,# INITWT   212
.          #,/P=/T=TR# N #/L=# LASTPART #/#
* NOW CYCLE THRU REST OF TRAITS, CALCULATING THIS TUPLES VALUES
 G5        $(#TR# N ) = $(#TR# N) TR #(# VAL #),#                        213
 G6        COPYTRAITS *TR* #(# *X* #)/# = /S($TR)                        214
* AT $TR WILL COMPUTE TUPLES VAL FOR THIS TRAIT, RETURN TO $RET
           PRINT = #NEW CHARACTERIZER#                                   215
           PRINT75((#CHAR# N),$(#CHAR# N))                               216
* IF COMPOUND CHAR., DONT PUT IT IN 1ST LEVEL MEMORY
           DESCR #*C# /S(IN)                                             217
           MEMORY = MEMORY #CHAR# N #,# /(IN)                            218
*
* BEGIN ROUTINES TO COMPUTE TRAIT VALUES FOR TUPLE IN DESCR
*
```

```
* CALCULATE MAX COLS. BETWEEN LEFTMOST AND RIGHTMOST PARTS OF TUPLE
HOROSPRED COPY = DESCR                                                    219
          COPY *X* #,# *LEFT* #,# *X* #,# =                               220
          RT = LEFT                                                       221
          NEXT = LEFT                                                     222
T5        COPY *X* #,# *NOW*  #,# *X* #,# = /F(T6)                        223
          NEXT = NOW + NEXT                                               224
          RT = .GT(NEXT,RT) NEXT /S(T5)                                   225
          LEFT = .LT(NEXT,LEFT) NEXT /(T5)                                226
T6        VAL = RT - LEFT /($RET)                                         227
*
* CALCULATE MAX ROWS BETWEEN TOPMOST AND BOTMOST PARTS OF TUPLE
VERTSPRED DESCR *X* #,# *COPY*                                            228
          VAL = #0#                                                       229
T7        COPY *NEXT* #,# *X* #,# = /F($RET)                              230
          VAL = VAL + NEXT /(T7)                                          231
*
*CALCULATE NUMBER OF POINTS IN TUPLE WHICH LIE ON EDGE OF PATTERN
* VAL NORMALIZED OVER 10
GRIDEDGE COPY = DESCR                                                     232
          BLANK *ROW* *COL*                                               233
          VAL = #0#                                                       234
T9        COPY *RO*  #,# *CO*  #,# *X* #,# = /F(T9R)                      235
          ROW = ROW + RO                                                  236
          VAL = .EQ(ROW,#0#) VAL + #1# /S(T9)                             237
          VAL = .EQ(ROW,COLSIZE - #1#) VAL + #1#  /S(T9)                  238
          COL = COL + CO                                                  239
          VAL = .LE(COL,#0#) VAL + #1# /S(T9)                             240
          VAL = .GE(COL + PC * #2# * WAL,ROWSIZE) VAL + #1# /(T9)         241
T9R       VAL = (VAL * #10#) / TUP /($RET)                                242
*
*CALCULATE SUM OF ABSOLUTE DIFFERENCES BETWEEN CORRESPONDING  DIGITS
* IN ALL POSSIBLE PAIRS
PROXIM    COPY = DESCR                                                    243
          VAL = #0#                                                       244
          BLANK *ROWS* *COLS*  *ROW* *COL*                                245
T10       COPY *RO*  #,# *CO*  #,# *X* #,# = /F($RET)                     246
          ROW = ROW + RO                                                  247
          COL = COL + CO                                                  248
          CROW = ROWS                                                     249
T11       CROW *NEXT* #,# = /F(T12)                                       250
          VAL = VAL + ABSVAL(ROW - NEXT) /(T11)                           251
T12       CCOL = COLS                                                     252
T13       CCOL *NEXT* #,# = /F(T14)                                       253
          VAL = VAL + ABSVAL(COL - NEXT) /(T13)                           254
T14       ROWS = ROWS ROW #,#                                             255
          COLS = COLS COL #,# /(T10)                                      256
*
* CHECKS TO SEE IF CHAR IS COMPOUND
COMPOUND DESCR #*C# /S(T15)                                               257
          VAL = #NO# /($RET)                                              258
T15       VAL = #YES# /($RET)                                             259
***
**FUNCTIONS
*
* FUNCTION TRAITWT(TN,DX) ADDS DX (= INC OR DEC) TO THE WEIGHT IN
* CHTRAITS FOR EACH TRAIT IN THE TN-TH TRAIT LIST
T1        COPY = $(#TR# TN)                                               260
T1A       COPY        *TRAIT* #(# *VAL* #),# = /F(RETURN)                 261
          CHTRAITS TRAIT #(# *VLIST* #)/# = /F(T2)                        262
          VLIST #,# VAL #-# *WT* #,# = #,# VAL #-# WT + DX #,#             263
.         /S(T4)F(T3)
T2        VLIST = #,#                                                     264
T3        VLIST = VLIST VAL #-# DX #,#                                    265
```

```
* IF UPWEIGHTING, ALSO PUT VAL + 1 ON VLIST IF NOT ALREADY THERE
T4        .GT(DX,#0#) /F(T4A)                                          266
          .NUM(VAL) /F(T4A)                                            267
          VLIST #,# (VAL + #1#) #=# /S(T4A)                            268
          VLIST = VLIST (VAL + #1#) #=# INC #,#                        269
* REASSEMBLE CHTRAITS
T4A       CHTRAITS = TRAIT #(# VLIST #)/# CHTRAITS /(T1A)              270
*
* RANTUP(LIST) RETURNS PLIST WITH A NEW RANDOMLY CHOSEN GRID POSITION
* (ROW,COL) INSERTED IN ORDER OF ASCENDING ROW NUMBER, THEN COL. NUMBER
RT        POS = .RANF(GRIDSIZE) - #1#                                  271
          ROW = #0#                                                    272
RT1       .LT(POS,ROWSIZE) /S(RT2)                                     273
          ROW = ROW + #1#                                              274
          POS = POS - ROWSIZE /(RT1)                                   275
* PUT NEW PART IN ORDERED POSITION IN PLIST
RT2       BLANK *LEFT*                                                 276
RT3       LIST *R* #,# *C* #,# = /F(RT5)                               277
          LEFT = .LT(R,ROW) LEFT R #,# C #,# /S(RT3)                   278
          LEFT = .GT(R,ROW) LEFT ROW #,# POS #,# R #,# C #,# /S(RT4)   279
* ROWS ARE SAME
          LEFT = .GT(C,POS) LEFT ROW #,# POS #,# R #,# C #,# /S(RT4)   280
          LEFT = .LT(C,POS) LEFT R #,# C #,# /S(RT3)                   281
* IF FAIL, A POSITION ALREADY IN TUPLE HAS BEEN DUPLICATED;
* TRY ANOTHER POSITION
          LIST = LEFT R #,# C #,# LIST /(RT)                           282
RT4       RANTUP = LEFT LIST /(RETURN)                                 283
RT5       RANTUP = LEFT ROW #,# POS #,# /(RETURN)                      284
*
* PROBCHOOSE(T)
* CHOOSES ACCORDING TO VALUE PROBABILITIES FROM T SUBLIST IN COPY OF
* CHTRAITS
PROBCH    BLANK *PL* *K*                                               285
* GET T LIST TO CHOOSE VALUE FROM
          COPYTRAITS T #(,# *LIST* #)/# =                              286
CH1       LIST *V* #=# *W* #,# = /F(CH3)                               287
          .LE(W,#0#) /S(CH1)                                           288
CH2       PL = PL V #,#                                                289
          W = W - #1#                                                  290
          K = K + #1#                                                  291
          .EQ(W,#0#) /F(CH2)S(CH1)                                     292
CH3       PROBCHOOSE = .EQ(K,#0#) V /S(RETURN)                         293
          W = .RANF(K)                                                 294
CH4       PL *PROBCHOOSE* #,# = /F(RETURN)                             295
          W = W - #1#                                                  296
          .EQ(W,#0#) /S(RETURN)F(CH4)                                  297
*
* RETURNS ABSOLUTE VALUE OF K
ABSVAL    ABSVAL = .GE(K,#0#) K /S(RETURN)                             298
          ABSVAL = #0# - K /(RETURN)                                   299
*
* TUPLEDESCR(T,P)
* GENERATES TUPLE DESCRIPTION WITH RANDOM PARTS ORDERED BY ROWS
* RELATIVE ADDRESSES
* PUTS ABSOLUTE ADDRESS OF LAST PART IN LASTPART
D1        BLANK *PLIST* *DES* *PREROW* *PRECOL*                        300
* GET T RANDOM PARTS INTO PLIST(RANTUP RETURNS ORDERED PLIST)
D2        PLIST = RANTUP(PLIST)                                        301
          T = .GT(T,#1#) T - #1# /S(D2)                                302
* CHANGE THE ABSOLUTE ADDRESS TO RELATIVE AND BUILD DESCR
D3        PLIST *ROW* #,# *COL* #,# = /F(D4)                           303
          .LE(COL + P,ROWSIZE) /S(D5)                                  304
          $(#ROW# ROW) *LEFT/(COL + SIZE(EDGE))* *THIS/(ROWSIZE - COL)*  305
          DES = DES ROW - PREROW #,# COL - PRECOL #,# (ROWSIZE - COL +   306
.         (#2# * VAL)) #*# THIS #-),# /(D6)
D5        $(#ROW# ROW) *LEFT/(COL + SIZE(EDGE))* *THIS/P*              307
```

```
         DES = DES ROW - PREROW #.# COL - PRECOL #.# PC + (#2# * WBL)     308
.        #*# THIS #-1,#
D6       PREROW = ROW                                                    309
         PRECOL = COL /(D3)                                              310
D4       LASTPART = ROW #.# COL                                          311
         TUPLEDESCR = DES /(RETURN)                                      312
*
* LARGEST(TR) RETURNS THE LARGEST TRAIT VALUE LISTED FOR TR IN CHTRAITS
LARGEST  CHTRAITS TR #(.# *LIST* #)#                                     313
         LIST *L* #-# *V* #,# =                                          314
LAR1     LIST *NEXT* #-# *V* #,# = /F(LAR2)                              315
         L = .GT(NEXT,L) NEXT /(LAR1)                                    316
LAR2     LARGEST = L /(RETURN)                                           317
*
*ROWORDER(PT,LIST) RETURNS LIST WITH PT INSERTED ACCORDING TO
* ASCENDING ROW NUMBER, THEN COL NUMBER
ROWORDER PT *ROW* #.# *COL* /F(RETURN)                                   318
         BLANK *LEFT*                                                    319
RW3      LIST *R* #.# *C* #.# *X* #,# = /F(RW5)                          320
         LEFT = .LT(R,ROW) LEFT R #.# C #.# X #,# /S(RW3)                321
         LEFT = .GT(R,ROW) LEFT ROW #.# COL #.# CH #,# R #.# C  #.# X    322
.        #,# /S(RW4)
* ROWS ARE SAME
         LEFT = .GT(C,COL) LEFT ROW #.# COL #.# CH #,# R #.# C  #.# X    323
.        #,# /S(RW4)
         LEFT = LEFT R #.# C #.# X #,# /(RW3)                            324
RW4      ROWORDER = LEFT LIST /(RETURN)                                  325
RW5      ROWORDER = LEFT ROW #.# COL #.# CH #,# /(RETURN)                326
*
* FUNCTION SECS(T) RETURNS THE NUMBER OF SECONDS IN TIME STRING T.
* OF FORM HHMMSS
SEC      T *H/#2#* *M/#2#* *S/#2#*                                       327
         SECS = S + M * #60# + H * #3600#  /(RETURN)                     328
* FUNCTION PRINT75(C,S) PRINTS OUT THE CHARACTERIZER C AND ITS CONTENTS
* S IN LINES OF NOT MORE THAN 75 COLUMNS
PRINT75  S *S1/#69#* = /S(PR1)                                           329
         PRINT = #   # C # = # S /(RETURN)                               330
PR1      PRINT = #   # C # = # S1                                        331
PR2      S *S1/#75#* = /S(PR3)                                           332
         PRINT = #      # S /(RETURN)                                    333
PR3      PRINT = #      # S1 /(PR2)                                      334
*
* STORE LEARNING FOR THIS RUN
KEEPFILE SYSPPT = #MEMORY==# MEMORY #$$#                                 335
         SYSPPT = #N==# N #$$#                                           336
         K = #1#                                                         337
UPK      SYSPPT = #CHAR# K #==# $(#CHAR# K) #$$#                         338
         SYSPPT = #TR# K #==# $(#TR# K) #$$#                             339
         K = .LT(K,N) K + #1# /S(UPK)                                    340
         SYSPPT = #CHTRAITS==# CHTRAITS #$$#                             341
         SYSPPT = #TOGEN==# TOGEN #$$#                                   342
         SYSPPT = #$$$#                                                  343
END                                                                      344

LATION COMPLETED AT 0223 -49
L PROGRAM OCCUPIES  2575 WORDS
```

Performance at Start of Run
(No Characterizers in Memory)

```
INITIAL MEMORY
INPUT PATTERN
      0011111100
      0011111100
      0000110000
      0000110000
      0000110000
      0000110000
      0000110000
      0000110000
      0011111100
      0011111100
FEEDBACK = I
IT IS UNKNOWN
REWEIGHTED CHARACTERIZERS
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR1 = D=3.6.1*0-1,6.-5.1*0-1,/I=2*I.5,/P=/T=TR1/L=9.1/
INPUT PATTERN
      1100000011
      1100000011
      1100000011
      1100000011
      1111111111
      1111111111
      1100000011
      1100000011
      1100000011
      1100000011
FEEDBACK = H
IT IS UNKNOWN
REWEIGHTED CHARACTERIZERS
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR2 = D=2.0.1*1-1,5.5.1*0-1,/1=2*H.5,/P=/T=TR2/L=7.5/
INPUT PATTERN
      0011111100
      0111111110
      1110000111
      1100000011
      1100000011
```

```
        1100000011
        1100000011
        1110000111
        0111111110
        0011111100
  FEEDBACK = O
  IT IS H
  REWEIGHTED CHARACTERIZERS
    CHAR1 = D=3.6.1*0-1.6.-5.1*0-1./I=2*0.5.2*1.5./P=/T=TR1/L=9.1/
    CHAR2 = D=2.0.1*1-1.5.5.1*0-1./I=2*0.5.2*H.4./P=/T=TR2/L=7.5/
  WOBBLE-ADJUSTED CHARACTERIZERS
  NEW CHARACTERIZER
    CHAR3 = D=1.1.1*1-1.5.7.1*1-1./I=2*0.5./P=/T=TR3/L=6.8/
  INPUT PATTERN
        1111111111
        1111111111
        1100000000
        1100000000
        1111110000
        1111110000
        1100000000
        1100000000
        1111111111
        1111111111
  FEEDBACK = F
  IT IS O
  REWEIGHTED CHARACTERIZERS
    CHAR2 = D=2.0.1*1-1.5.5.1*0-1./I=2*E.5.2*O.4.2*H.4./P=/T=TR2/L=7.5/
  WOBBLE-ADJUSTED CHARACTERIZERS
  NEW CHARACTERIZER
    CHAR4 = D=0.7.1*1-1.3.-1.1*0-1./I=2*E.5./P=/T=TR4/L=3.6/
  INPUT PATTERN
        0000110000
        0001111000
        0011001100
        0011001100
        0110000110
        0111111110
        0111111110
        1100000011
        1100000011
        1100000011
  FEEDBACK = A
  IT IS UNKNOWN
  REWEIGHTED CHARACTERIZERS
  WOBBLE-ADJUSTED CHARACTERIZERS
  NEW CHARACTERIZER
    CHAR5 = D=0.9.1*0-1.6.-1.1*1-1./I=2*A.5./P=/T=TR5/L=6.8/
  INPUT PATTERN
        0001111110
        0001111100
        0000011000
        0000110000
        0000110000
        0001100000
        0001100000
        0001100000
        0111110000
        0111110000
  FEEDBACK = I
  IT IS E
  REWEIGHTED CHARACTERIZERS
    CHAR4 = D=0.7.1*1-1.3.-1.1*0-1./I=2*I.5.2*E.4./P=/T=TR4/L=3.6/
```

```
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR6 = D=3.0.1*0-1,6.7.1*0-1,/I=2*I.5,/P=/T=TR6/L=9.7/
INPUT PATTERN
     0110000110
     0110000110
     0110000110
     0110000110
     0110000110
     0111111110
     0111111110
     0110001110
     0110001110
     0110001110
FEEDBACK = H
IT IS A
REWEIGHTED CHARACTERIZERS
  CHAR3 = D=1.1.1*1-1,5.7.1*1-1,/I=2*H.5,2*O.5,/P=/T=TR3/L=6.8/
  CHAR4 = D=0.7.1*1-1,3.-1.1*0-1,/I=2*H.5,2*I.5,2*E.4,/P=/T=TR4/L=3.6/
  CHAR5 = D=0.9.1*0-1,6.-1.1*1-1,/I=2*H.5,2*A.4,/P=/T=TR5/L=6.8/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR7 = D=1.4.1*0-1,5.3.1*1-1,/I=2*H.5,/P=/T=TR7/L=6.7/
INPUT PATTERN
     0011111000
     0110001100
     0110000110
     0110000110
     0110000110
     0110000110
     0110000110
     0110000110
     0110001100
     0011111100
FEEDBACK = O
IT IS H
REWEIGHTED CHARACTERIZERS
  CHAR1 = D=3.6.1*0-1,6.-5.1*0-1,/I=2*O.6,2*I.5,/P=/T=TR1/L=9.1/
  CHAR3 = D=1.1.1*1-1,5.7.1*1-1,/I=2*H.4,2*O.6,/P=/T=TR3/L=6.8/
  CHAR5 = D=0.9.1*0-1,6.-1.1*1-1,/I=2*O.5,2*H.4,2*A.4,/P=/T=TR5/L=6.8/
  CHAR7 = D=1.4.1*0-1,5.3.1*1-1,/I=2*O.5,2*H.4,/P=/T=TR7/L=6.7/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR8 = D=6.8.5*CHAR3-1,0.0.5*CHAR5-1,3.-7.5*CHAR1-1,/I=2*O.5,2*$CHAR1,2*$CHA
    R3,2*$CHAR5,/P=/T=TR8/L=9.1/
INPUT PATTERN
     .0111111110.
     .0111111110.
     .0110000000.
     .0110000000.
     .0111100000.
     .0111100000.
     .0110000000.
     .0110000000.
     .0111111110.
     .0111111110.
FEEDBACK = E
IT IS I
REWEIGHTED CHARACTERIZERS
  CHAR4 = D=0.7.1*1-1,3.-1.1*0-1,/I=2*H.5,2*I.4,2*E.5,/P=/T=TR4/L=3.6/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR9 = D=6.4.4*00-1,3.-3.4*11-1,0.2.4*11-1,/I=2*E.5,/P=/T=TR9/L=9.3/
```

```
INPUT PATTERN
      .0000110000.
      .0000110000.
      .0001111000.
      .0011001000.
      .0011001100.
      .0011111100.
      .0111111100.
      .0110000110.
      .1100000110.
      .1100000110.
FEEDBACK = A
IT IS UNKNOWN
REWEIGHTED CHARACTERIZERS
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR10 = D=3.8.4*.0-1.2.-4.4*11-1.4.1.4*00-1./I=2*A.5./P=/T=TR10/L=9.5/
INPUT PATTERN
      .0011111100.
      .0011111100.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0011111100.
      .0011111100.
FEEDBACK =
IT IS E
INPUT PATTERN
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
      .1111111111.
      .1111111111.
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
FEEDBACK =
IT IS O
INPUT PATTERN
      .0011111100.
      .0111111110.
      .1110000111.
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
      .1110000111.
      .0111111110.
      .0011111100.
FEEDBACK =
IT IS E
INPUT PATTERN
      .1111111111.
      .1111111111.
      .1100000000.
      .1100000000.
      .1111110000.
      .1111110000.
```

```
      .1100000000.
      .1100000000.
      .1111111111.
      .1111111111.
FEEDBACK =
IT IS E
INPUT PATTERN
      .0000110000.
      .0001111000.
      .0011001100.
      .0011001100.
      .0110000110.
      .0111111110.
      .0111111110.
      .1100000011.
      .1100000011.
      .1100000011.
FEEDBACK =
IT IS O
INPUT PATTERN
      .0011111100.
      .0011111100.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0000110000.
      .0011111100.
      .0011111100.
FEEDBACK = I
IT IS E
REWEIGHTED CHARACTERIZERS
  CHAR4 = D=0.7.1*1=1.3.-1.1*0-1./I=2*H.5.2*I.5.2*E.4./P=/T=TR4/L=3.6/
  CHAR9 = D=6.4.4*00-1.3.-3.4*11-1.0.2.4*11-1./I=2*I.5.2*E.4./P=/T=TR9/L=9.3/
  CHAR8 = D=6.8.5*CHAR3-1.0.0.5*CHAR5-1.3.-7.5*CHAR1-1./I=2*I.5.2*O.5.2*$CHAR1.
    2*$CHAR3.2*$CHAR5./P=/T=TR8/L=9.1/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR11 = C=5.2.4*00-1.2.-2.4*00-1.2.9.3*0-1./I=2*I.5./P=/T=TR11/L=9.9/
INPUT PATTERN
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
      .1111111111.
      .1111111111.
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
FEEDBACK = E
IT IS O
REWEIGHTED CHARACTERIZERS
  CHAR2 = D=2.0.1*1-1.5.5.1*0-1./I=2*E.5.2*O.3.2*H.5./P=/T=TR2/L=7.5/
  CHAR10 = C=3.8.4*00-1.2.-4.4*11-1.4.1.4*00-1./I=2*H.5.2*A.5./P=/T=TR10/L=9.5/
  CHAR8 = D=6.8.5*CHAR3-1.0.0.5*CHAR5-1.3.-7.5*CHAR1-1./I=2*H.5.2*I.5.2*O.4.2*$
    CHAR1.2*$CHAR3.2*$CHAR5./P=/T=TR8/L=9.1/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR12 = C=7.5.7*CHAR2-1.2.-4.7*CHAR8-1.0.4.8*CHAR10-1./I=2*H.5.2*$CHAR2.2*$CH
    AR10.2*$CHAR8./P=/T=TR12/L=9.5/
INPUT PATTERN
```

```
      .0011111100.
      .0111111110.
      .1110000111.
      .1100000011.
      .1100000011.
      .1100000011.
      .1100000011.
      .1110000111.
      .0111111110.
      .0011111100.
FEEDBACK = O
IT IS I
REWEIGHTED CHARACTERIZERS
  CHAR4 = D=0.7.1*1-1,3.-1.1*0-1,/I=2*O.5,2*H.5,2*I.4,2*E.4,/P=/T=TR4/L=3.6/
  CHAR9 = D=6.4.4*00-1,3.-3.4*11-1,0.2.4*11-1,/I=2*O.5,2*I.4,2*E.4,/P=/T=TR9/L=
    9.3/
  CHAR11 = C=5.2.4*00-1,2.-2.4*00-1,2.9.3*0-1,/I=2*O.5,2*I.4,/P=/T=TR11/L=9.9/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*O.5,2*H.5,2*$CHAR2
    ,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR13 = C=3.6.7*CHAR4-1,6.-3.7*CHAR9-1,0.6.8*CHAR11-1,/I=2*O.5,2*$CHAR4,2*$CH
    AR9,2*$CHAR11,/P=/T=TR13/L=9.9/
INPUT PATTERN
      .1111111111.
      .1111111111.
      .1100000000.
      .1100000000.
      .1111110000.
      .1111110000.
      .1100000000.
      .1100000000.
      .1111111111.
      .1111111111.
FEEDBACK = E
IT IS O
REWEIGHTED CHARACTERIZERS
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*E.5,2*O.4,2*H.5,2*
    $CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
  CHAR13 = C=3.6.7*CHAR4-1,6.-3.7*CHAR9-1,0.6.8*CHAR11-1,/I=2*E.5,2*O.4,2*$CHAR4
    ,2*$CHAR9,2*$CHAR11,/P=/T=TR13/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR14 = C=0.0.5*111-1,1.9.3*1-1,6.-8.5*100-1,/I=2*E.5,/P=/T=TR14/L=7.1/
INPUT PATTERN
      .0000110000.
      .0001111000.
      .0011001100.
      .0011001100.
      .0110000110.
      .0111111110.
      .0111111110.
      .1100000011.
      .1100000011.
      .1100000011.
FEEDBACK = A
IT IS H
REWEIGHTED CHARACTERIZERS
  CHAR6 = D=3.0.1*0-1,6.7.1*0-1,/I=2*A.5,2*I.5,/P=/T=TR6/L=9.7/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*A.5,2*E.5,2*O.4,2*
    H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
```

```
   CHAR15 = D=2.0.5*001-1+1.2.5*110-1+3.-1.5*111-1+/I=2*A.5+/P=/T=TR15/L=6.1/
 INPUT PATTERN
      .0001111110.
      .0001111100.
      .0000011000.
      .0000110000.
      .0000110000.
      .0001100000.
      .0001100000.
      .0001100000.
      .0111110000.
      .0111110000.
 FEEDBACK = I
 IT IS E
 REWEIGHTED CHARACTERIZERS
   CHAR6 = D=3.0.1*0-1+6.7.1*0-1+/I=2*A.5+2*I.6+/P=/T=TR6/L=9.7/
   CHAR13 = D=3.6.7*CHAR4-1+6.-3.7*CHAR9-1+0.6.8*CHAR11-1+/I=2*I.5+2*E.4+2*O.4+2*
     $CHAR4+2*$CHAR9+2*$CHAR11+/P=/T=TR13/L=9.9/
   CHAR12 = D=7.5.7*CHAR2-1+2.-4.7*CHAR8-1+0.4.8*CHAR10-1+/I=2*I.5+2*A.5+2*E.4+2*
     O.4+2*H.4+2*$CHAR2+2*$CHAR10+2*$CHAR8+/P=/T=TR12/L=9.5/
 WOBBLE-ADJUSTED CHARACTERIZERS
 NEW CHARACTERIZER
   CHAR16 = D=0.2.5*011-1+5.5.5*000-1+2.-5.5*011-1+/I=2*I.5+/P=/T=TR16/L=7.2/
 INPUT PATTERN
      .0110000110.
      .0110000110.
      .0110000110.
      .0110000110.
      .0110000110.
      .0111111110.
      .0111111110.
      .0110001110.
      .0110001110.
      .0110001110.
 FEEDBACK = H
 IT IS O
 REWEIGHTED CHARACTERIZERS
   CHAR7 = D=1.4.1*0-1+5.3.1*1-1+/I=2*O.4+2*H.5+/P=/T=TR7/L=6.7/
   CHAR13 = D=3.6.7*CHAR4-1+6.-3.7*CHAR9-1+0.6.8*CHAR11-1+/I=2*H.5+2*I.5+2*E.4+2*
     O.3+2*$CHAR4+2*$CHAR9+2*$CHAR11+/P=/T=TR13/L=9.9/
   CHAR12 = D=7.5.7*CHAR2-1+2.-4.7*CHAR8-1+0.4.8*CHAR10-1+/I=2*I.5+2*A.5+2*E.4+2*
     O.3+2*H.5+2*$CHAR2+2*$CHAR10+2*$CHAR8+/P=/T=TR12/L=9.5/
 WOBBLE-ADJUSTED CHARACTERIZERS
 NEW CHARACTERIZER
   CHAR17 = D=5.0.8*0111-1+0.9.5*0-1+1.-8.8*1111-1+1.1.8*1000-1+/I=2*H.5+/P=/T=TR
     17/L=7.2/
 INPUT PATTERN
      ..0011111000..
      ..0110001100..
      ..0110000110..
      ..0110000110..
      ..0110000110..
      ..0110000110..
      ..0110000110..
      ..0110000110..
      ..0110001100..
      ..0011111100..
 FEEDBACK = O
 IT IS H
 REWEIGHTED CHARACTERIZERS
   CHAR7 = D=1.4.1*0-1+5.3.1*1-1+/I=2*O.5+2*H.4+/P=/T=TR7/L=6.7/
   CHAR14 = D=0.0.5*111-1+1.9.3*1-1+6.-8.5*100-1+/I=2*O.5+2*E.5+/P=/T=TR14/L=7.1/
```

```
  CHAR13 = D=3.6.7*CHAR4-1,6.-3.7*CHAR9-1,0.6.8*CHAR11-1,/I=2*H.4,2*I.5,2*E.4,2*
    O.4,2*$CHAR4,2*$CHAR9,2*$CHAR11,/P=/T=TR13/L=9.9/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.6*CHAR10-1,/I=2*I.5,2*A.5,2*E.4,2*
    O.4,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR18 = C=1.9.5*0-1,1.-3.8*0110-1,3.0.8*0110-1,2.-6.8*0110-1,/I=2*0.5,/P=/T=T
    R18/L=7.0/
INPUT PATTERN
      ..0111111110..
      ..0111111110..
      ..0110000000..
      ..0110000000..
      ..0111100000..
      ..0111100000..
      ..0110000000..
      ..0110000000..
      ..0111111110..
      ..0111111110..
FEEDBACK = E
IT IS H
REWEIGHTED CHARACTERIZERS
  CHAR14 = D=0.0.5*111-1,1.9.3*1-1,6.-8.5*100-1,/I=2*O.5,2*E.6,/P=/T=TR14/L=7.1/

  CHAR17 = D=5.0.8*0111-1,0.9.5*0-1,1.-8.8*1111-1,1.1.8*1000-1,/I=2*E.5,2*H.4,/P
    =/T=TR17/L=7.2/
  CHAR13 = D=3.6.7*CHAR4-1,6.-3.7*CHAR9-1,0.6.8*CHAR11-1,/I=2*H.3,2*I.5,2*E.5,2*
    O.4,2*$CHAR4,2*$CHAR9,2*$CHAR11,/P=/T=TR13/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR19 = D=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*E.5,2*$CHAR14,
    2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
INPUT PATTERN
      ..0000110000..
      ..0000110000..
      ..0001111000..
      ..0011001000..
      ..0011001100..
      ..0011111100..
      ..0111111100..
      ..0110000110..
      ..1100000110..
      ..1100000110..
FEEDBACK = A
IT IS A
INPUT PATTERN
      ..0001111110..
      ..0001111100..
      ..0000011000..
      ..0000110000..
      ..0000110000..
      ..0001100000..
      ..0001100000..
      ..0001100000..
      ..0111110000..
      ..0111110000..
FEEDBACK =
IT IS I
INPUT PATTERN
      ..0110000110..
      ..0110000110..
      ..0110000110..
      ..0110000110..
```

```
        ..0110000110..
        ..0111111110..
        ..0111111110..
        ..0110001110..
        ..0110001110..
        ..0110001110..
  FEEDBACK =
  IT IS E
  INPUT PATTERN
        ..0011111000..
        ..0110001100..
        ..0110000110..
        ..0110000110..
        ..0110000110..
        ..0110000110..
        ..0110000110..
        ..0110000110..
        ..0110001100..
        ..0011111100..
  FEEDBACK =
  IT IS E
  INPUT PATTERN
        ..0111111110..
        ..0111111110..
        ..0110000000..
        ..0110000000..
        ..0111100000..
        ..0111100000..
        ..0110000000..
        ..0110000000..
        ..0111111110..
        ..0111111110..
  FEEDBACK =
  IT IS E
  INPUT PATTERN
        ..0000110000..
        ..0000110000..
        ..0001111000..
        ..0011001000..
        ..0011001100..
        ..0011111100..
        ..0111111100..
        ..0110000110..
        ..1100000110..
        ..1100000110..
  FEEDBACK =
  IT IS A
  INPUT PATTERN
        ..0011111100..
        ..0011111100..
        ..0000110000..
        ..0000110000..
        ..0000110000..
        ..0000110000..
        ..0000110000..
        ..0000110000..
        ..0011111100..
        ..0011111100..
  FEEDBACK = 1
  IT IS I
  INPUT PATTERN
        ..1100000011..
        ..1100000011..
```

```
      ..1100000011..
      ..1100000011..
      ..1111111111..
      ..1111111111..
      ..1100000011..
      ..1100000011..
      ..1100000011..
      ..1100000011..
FEEDBACK = H
IT IS O
REWEIGHTED CHARACTERIZERS
  CHAR3 = D=1.1.1*1-1,5.7.1*1-1,/I=2*H.5,2*O.6,/P=CHAR8,/T=TR3/L=6.8/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.5,2*A.5,2*E.4,2*
    O.3,2*H.5.2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
  CHAR19 = C=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*H.5,2*E.5,2*$C
    HAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
NEW CHARACTERIZER
  CHAR20 = C=1.8.6*11-1,2.-5.8*0000-1,3.-1.8*0000-1,1.7.5*1-1,/I=2*H.5,/P=/T=TR2
    0/L=7.9/
INPUT PATTERN
      ..0011111100..
      ..0111111110..
      ..1110000111..
      ..1100000011..
      ..1100000011..
      ..1100000011..
      ..1100000011..
      ..1110000111..
      ..0111111110..
      ..0011111100..
FEEDBACK = O
IT IS H
REWEIGHTED CHARACTERIZERS
  CHAR20 = C=1.8.6*11-1,2.-5.8*0000-1,3.-1.8*0000-1,1.7.5*1-1,/I=2*O.5,2*H.4,/P=
    /T=TR20/L=7.9/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.5,2*A.5,2*E.4,2*
    O.4,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
  CHAR19 = C=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*O.5,2*H.4,2*E.
    5,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
      ..1111111111..
      ..1111111111..
      ..1100000000..
      ..1100000000..
      ..1111110000..
      ..1111110000..
      ..1100000000..
      ..1100000000..
      ..1111111111..
      ..1111111111..
FEEDBACK = E
IT IS O
REWEIGHTED CHARACTERIZERS
  CHAR20 = C=1.8.6*11-1,2.-5.8*0000-1,3.-1.8*0000-1,1.7.5*1-1,/I=2*E.5,2*O.4,2*H
    .4,/P=/T=TR20/L=7.9/
  CHAR12 = C=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.5,2*A.5,2*E.5,2*
    O.3,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
  CHAR19 = C=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*O.4,2*H.4,2*E.
    6,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
```

```
      ...0000110000...
      ...0001111000...
      ...0011001100...
      ...0011001100...
      ...0110000110...
      ...0111111110...
      ...0111111110...
      ...1100000011...
      ...1100000011...
      ...1100000011...
FEEDBACK = A
IT IS A
INPUT PATTERN
      ...0001111110...
      ...0001111100...
      ...0000011000...
      ...0000110000...
      ...0000110000...
      ...0001100000...
      ...0001100000...
      ...0001100000...
      ...0111110000...
      ...0111110000...
FEEDBACK = I
IT IS E
REWEIGHTED CHARACTERIZERS
  CHAR6 = D=3.0.1*0-1,6.7.1*0-1,/I=2*A.5,2*I.7,/P=/T=TR6/L=9,7/
  CHAR16 = D=0.2.5*011-1,5.5.5*000-1,2.-5.5*011-1,/I=2*I.6,/P=/T=TR16/L=7.2/
  CHAR20 = D=1.8.6*11-1,2.-5.8*0000-1,3,-1.8*0000-1,1.7.5*1-1,/I=2*I.5,2*E.4,2*0
    .4,2*H.4,/P=/T=TR20/L=7.9/
  CHAR12 = D=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.6,2*A.5,2*E.4,2*
    0.3,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
  CHAR19 = D=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2,7.10*CHAR13-1,/I=2*I.5,2*0.4,2*H.
    4,2*E.5,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0111111110...
      ...0111111110...
      ...0110001110...
      ...0110001110...
      ...0110001110...
FEEDBACK = H
IT IS H
INPUT PATTERN
      ...0011111000...
      ...0110001100...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110001100...
      ...0011111100...
FEEDBACK = O
IT IS I
REWEIGHTED CHARACTERIZERS
  CHAR7 = D=1.4.1*0-1,5.3.1*1-1,/I=2*O.6,2*H.4,/P=/T=TR7/L=6.7/
```

```
   CHAR20 = D=1.8.6*11-1,2.-5.8*0000-1,3.-1.8*0000-1,1.7.5*1-1,/I=2*I.4,2*E.4,2*0
      .5,2*H.4,/P=/T=TR20/L=7.9/
   CHAR19 = D=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*I.4,2*0.5,2*H.
      4,2*E.5,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
   CHAR12 = D=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.5,2*A.5,2*E.4,2*
      0.4,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
       ...0111111110...
       ...0111111110...
       ...0110000000...
       ...0110000000...
       ...0111100000...
       ...0111100000...
       ...0110000000...
       ...0110000000...
       ...0111111110...
       ...0111111110...
FEEDBACK = E
IT IS O
REWEIGHTED CHARACTERIZERS
   CHAR20 = D=1.8.6*11-1,2.-5.8*0000-1,3.-1.8*0000-1,1.7.5*1-1,/I=2*I.4,2*E.5,2*0
      .4,2*H.4,/P=/T=TR20/L=7.9/
   CHAR19 = D=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*I.4,2*0.4,2*H.
      4,2*E.6,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
       ...0000110000...
       ...0000110000...
       ...0001111000...
       ...0011001000...
       ...0011001100...
       ...0011111100...
       ...0111111100...
       ...0110000110...
       ...1100000110...
       ...1100000110...
FEEDBACK = A
IT IS E
REWEIGHTED CHARACTERIZERS
   CHAR15 = D=2.0.5*001-1,1.2.5*110-1,3.-1.5*111-1,/I=2*A.6,/P=/T=TR15/L=6.1/
   CHAR12 = D=7.5.7*CHAR2-1,2.-4.7*CHAR8-1,0.4.8*CHAR10-1,/I=2*I.5,2*A.6,2*E.3,2*
      0.4,2*H.4,2*$CHAR2,2*$CHAR10,2*$CHAR8,/P=/T=TR12/L=9.5/
   CHAR19 = D=7.1.10*CHAR14-1,0.1.10*CHAR17-1,2.7.10*CHAR13-1,/I=2*A.5,2*I.4,2*0.
      4,2*H.4,2*E.5,2*$CHAR14,2*$CHAR17,2*$CHAR13,/P=/T=TR19/L=9.9/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
       ...0001111110...
       ...0001111100...
       ...0000011000...
       ...0000110000...
       ...0000110000...
       ...0001100000...
       ...0001100000...
       ...0001100000...
       ...0111110000...
       ...0111110000...
FEEDBACK =
IT IS I
INPUT PATTERN
       ...0110000110...
       ...0110000110...
       ...0110000110...
```

```
      ...0110000110...
      ...0110000110...
      ...0111111110...
      ...0111111110...
      ...0110001110...
      ...0110001110...
      ...0110001110...
 FEEDBACK =
 IT IS 0
 INPUT PATTERN
      ...0011111000...
      ...0110001100...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110000110...
      ...0110001100...
      ...0011111100...
 FEEDBACK =
 IT IS 0
 INPUT PATTERN
      ...0111111110...
      ...0111111110...
      ...0110000000...
      ...0110000000...
      ...0111100000...
      ...0111100000...
      ...0110000000...
      ...0110000000...
      ...0111111110...
      ...0111111110...
 FEEDBACK =
 IT IS E
 INPUT PATTERN
      ...0000110000...
      ...0000110000...
      ...0001111000...
      ...0011001000...
      ...0011001100...
      ...0011111100...
      ...0111111100...
      ...0110000110...
      ...1100000110...
      ...1100000110...
 FEEDBACK =
 IT IS A
 INPUT PATTERN
      ...$END...
```

Memory After Given Feedback for 134 Instances

```
INITIAL MEMORY
  CHAR1 = D=2.4.1*1-1,0.1.1*1-2,3.-1.1*1-1,/I=2*A.4,2*I.3,3*T.7,/P=CHAR3,/
    T=TR1/L=5.4/
  CHAR2 = D=0.0.1*0-2,0.9.1*0-3,/I=2*S.4,2*A.4,2*O.5,3*I.3,3*T.-1,/P=CHAR1
    0,CHAR3,/T=TR2/L=0.9/
  CHAR3 = D=*CHAR2-5,*CHAR1-4,/I=2*S.4,2*A.4,2*O.5,7*I.6.5*T.4,6*$CHAR1,7*
    $CHAR2,/P=CHAR10,/T=TR3/L=5.4/
  CHAR4 = D=3.6.1*0-1,6.-5.1*1-1,/I=2*S.6,2*E.3,2*H.5,/P=CHAR19,CHAR6,/T=T
    R4/L=9.1/
  CHAR5 = D=2.0.1*1-1,0.5.1*0-1,/I=2*S.6,2*E.3,2*O.5,/P=CHAR19,CHAR6,/T=TR
    5/L=2.5/
  CHAR6 = D=2.5.5*CHAR5-1,7.-4.5*CHAR4-1,/I=2*O.3,2*H.7,2*S.2,2*E.6,2*$CHA
    R4,2*$CHAR5,/P=CHAR16,CHAR10,/T=TR6/L=9.1/
  CHAR7 = D=4.5.3*001-1,4.2.3*011-1,/I=2*A.6,/P=/T=TR7/L=8.7/
  CHAR8 = D=3.6.3*000-1,3.-2.3*000-1,/I=2*E.5,2*S.5,/P=CHAR11,/T=TR8/L=6.4
    /
  CHAR9 = D=0.0.2*11-1,5.1.2*11-1,/I=2*E.5,2*H.5,/P=CHAR11,/T=TR9/L=5.1/
  CHAR10 = D=0.9.5*CHAR2-1,5.-5.5*CHAR3-1,4.-3.5*CHAR6-1,/I=2*E.5,2*O.4,2*$
    CHAR2,2*$CHAR3,2*$CHAR6,/P=CHAR11,/T=TR10/L=9.1/
  CHAR11 = D=5.1.5*CHAR9-1,1.3.5*CHAR8-1,3.-3.6*CHAR10-1,/I=2*O.6,2*H.5,2*I
    .2,2*T.5,2*S.5,2*A.2,2*E.4,2*$CHAR8,2*$CHAR9,2*$CHAR10,/P=CHAR16,/T=TR1
    1/L=9.1/
  CHAR12 = D=6.4.6*1111-1,2.5.3*1-1,1.-8.6*1000-1,/I=2*H.5,2*A.5,/P=/T=TR12
    /L=4.1/
  CHAR13 = D=0.1.6*1111-1,5.5.6*1111-1,2.2.4*11-1,/I=2*O.5,2*S.6,/P=/T=TR13
    /L=7.8/
  CHAR14 = D=5.0.6*0000-1,0.3.6*0110-1,2.2.6*1000-1,2.2.5*000-1,/I=2*S.5,2*
    A.5,2*E.2,2*O.4,2*I.5,2*T.5,/P=/T=TR14/L=9.7/
  CHAR15 = D=3.1.6*0001-1,2.2.6*0110-1,1.-1.6*0011-1,0.7.3*0-1,/I=2*A.4,2*S
    .5,2*T.6,2*I.5,/P=/T=TR15/L=6.9/
  CHAR16 = D=9.1.5*CHAR6-1,0.0.6*CHAR11-1,/I=2*S.5,2*E.4,2*O.4,2*H.4,2*$CHA
    R6,2*$CHAR11,/P=CHAR19,/T=TR16/L=9.1/
  CHAR17 = D=2.1.6*1100-1,5.-1.6*1110-1,1.6.6*1110-1,0.2.4*10-1,/I=2*E.6,2*
    H.7,2*O.5,/P=/T=TR17/L=8.8/
  CHAR18 = D=1.6.6*1111-1,5.-3.6*0000-1,1.5.4*00-1,2.-7.6*1111-1,/I=2*S.5,2
    *E.4,/P=CHAR19,/T=TR18/L=9.1/
  CHAR19 = D=2.5.7*CHAR5-1,7.-4.7*CHAR4-1,0.0.8*CHAR18-1,0.0.8*CHAR16-1,/I=
    2*A.6,2*I.3,2*T.4,2*E.3,2*O.5,2*H.5,2*S.4,2*$CHAR4,2*$CHAR5,2*$CHAR18,2
    *$CHAR16,/P=/T=TR19/L=9.1/
  CHAR20 = D=1.5.6*0001-1,3.3.4*11-1,1.-2.6*1111-1,1.0.6*0011-1,/I=2*O.5,2*
    S.5,2*H.5,/P=/T=TR20/L=6.6/
  CHAR21 = D=0.1.9*01111-1,0.8.5*0-1,3.-8.9*10000-1,0.7.6*11-1,5.-5.9*11111
    -1,/I=2*H.4,2*A.5,2*T.3,2*I.4,2*S.4,2*E.5,2*O.5,/P=/T=TR21/L=8.3/
  CHAR22 = D=2.3.9*00000-1,3.3.8*0000-1,4.-6.9*11111-1,0.6.8*1111-1,0.1.7*1
```

```
    11-1,/I=2*I.4,2*0.1,2*S.7,2*E.5,/P=/T=TR22/L=9.7/
  CHAR23 = D=5.1.9*00011-1,2.7.6*11-1,1.-6.9*11111-1,0.2.9*11111-1,1.0.9*11
    111-1,/I=2*I.4,2*E.4,2*0.3,2*S.6,/P=/T=TR23/L=9.4/
INPUT PATTERN
 TIMEX = 12
     ...0011111100...
     ...0011111100...
     ...0000110000...
     ...0000110000...
     ...0000110000...
     ...0000110000...
     ...0000110000...
     ...0000110000...
     ...0011111100...
     ...0011111100...
FEEDBACK = I
IT IS S
REWEIGHTED CHARACTERIZERS
  CHAR14 = D=5.0.6*0000-1,0.3.6*0110-1,2.2.6*1000-1,2.2.5*000-1,/I=2*S.4,2*
    A.5,2*E.2,2*0.4,2*I.6,2*T.5,/P=/T=TR14/L=9.7/
  CHAR15 = D=3.1.6*0001-1,2.2.6*0110-1,1.-1.6*0011-1,0.7.3*0-1,/I=2*A.4,2*S
    .4,2*T.6,2*I.6,/P=/T=TR15/L=6.9/
  CHAR21 = D=0.1.9*01111-1,0.8.5*0-1,3.-8.9*10000-1,0.7.6*11-1,5.-5.9*11111
    -1,/I=2*H.4,2*A.5,2*T.3,2*I.5,2*S.3,2*E.5,2*0.5,/P=/T=TR21/L=8.3/
  CHAR22 = D=2.3.9*00000-1,3.3.8*0000-1,4.-6.9*11111-1,0.6.8*1111-1,0.1.7*1
    11-1,/I=2*I.5,2*0.1,2*S.6,2*E.5,/P=/T=TR22/L=9.7/
  CHAR23 = D=5.1.9*00011-1,2.7.6*11-1,1.-6.9*11111-1,0.2.9*11111-1,1.0.9*11
    111-1,/I=2*I.5,2*E.4,2*0.3,2*S.5,/P=/T=TR23/L=9.4/
  CHAR19 = D=2.5.7*CHAR5-1,7.-4.7*CHAR4-1,0.0.8*CHAR18-1,0.0.8*CHAR16-1,/I=
    2*A.6,2*I.4,2*T.4,2*E.3,2*0.5,2*H.5,2*S.3,2*$CHAR4,2*$CHAR5,2*$CHAR18,2
    *$CHAR16,/P=/T=TR19/L=9.1/
  CHAR11 = D=5.1.5*CHAR9-1,1.3.5*CHAR8-1,3.-3.6*CHAR10-1,/I=2*0.6,2*H.5,2*I
    .3,2*T.5,2*S.4,2*A.2,2*E.4,2*$CHAR8,2*$CHAR9,2*$CHAR10,/P=CHAR16,/T=TR1
    1/L=9.1/
WOBBLE-ADJUSTED CHARACTERIZERS
INPUT PATTERN
 TIMEX = 115
     ...1100000011...
     ...1100000011...
     ...1100000011...
     ...1100000011...
     ...1111111111...
     ...1111111111...
     ...1100000011...
     ...1100000011...
     ...1100000011...
     ...1100000011...
FEEDBACK = H
IT IS H
INPUT PATTERN
 TIMEX = 145
     ...0011111100...
     ...0111111110...
     ...1110000111...
     ...1100000011...
     ...1100000011...
     ...1100000011...
     ...1100000011...
     ...1110000111...
     ...0111111110...
     ...0011111100...
FEEDBACK = O
IT IS O
INPUT PATTERN
 TIMEX = 184
```

```
       ...1111111111...
       ...1111111111...
       ...1100000000...
       ...1100000000...
       ...1111110000...
       ...1111110000...
       ...1100000000...
       ...1100000000...
       ...1111111111...
       ...1111111111...
FEEDBACK = E
IT IS E
INPUT PATTERN
 TIMEX = 196
       ...0000110000...
       ...0001111000...
       ...0011001100...
       ...0011001100...
       ...0110000110...
       ...0111111110...
       ...0111111110...
       ...1100000011...
       ...1100000011...
       ...1100000011...
FEEDBACK = A
IT IS A
INPUT PATTERN
 TIMEX = 206
       ...0001111110...
       ...0001111100...
       ...0000011000...
       ...0000110000...
       ...0000110000...
       ...0001100000...
       ...0001100000...
       ...0001100000...
       ...0111110000...
       ...0111110000...
FEEDBACK = I
IT IS I
INPUT PATTERN
 TIMEX = 218
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0111111110...
       ...0111111110...
       ...0110001110...
       ...0110001110...
       ...0110001110...
FEEDBACK = H
IT IS H
INPUT PATTERN
 TIMEX = 229
       ...0011111000...
       ...0110001100...
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0110000110...
       ...0110000110...
```

```
      ...0110001100...
      ...0011111100...
FEEDBACK = O
IT IS O
INPUT PATTERN
 TIMEX = 239
      ...0111111110...
      ...0111111110...
      ...0110000000...
      ...0110000000...
      ...0111100000...
      ...0111100000...
      ...0110000000...
      ...0110000000...
      ...0111111110...
      ...0111111110...
FEEDBACK = E
IT IS E
INPUT PATTERN
 TIMEX = 250
      ...0000110000...
      ...0000110000...
      ...0001111000...
      ...0011001000...
      ...0011001100...
      ...0011111100...
      ...0111111100...
      ...0110000110...
      ...1100000110...
      ...1100000110...
FEEDBACK = A
IT IS A
INPUT PATTERN
 TIMEX = 260
      ...0011111100...
      ...0011111100...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0011111100...
      ...0011111100...
FEEDBACK =
IT IS I
INPUT PATTERN
 TIMEX = 272
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1111111111...
      ...1111111111...
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1100000011...
FEEDBACK =
IT IS H
INPUT PATTERN
 TIMEX = 284
      ...0011111100...
      ...0111111110...
      ...1110000111...
```

```
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1110000111...
      ...0111111110...
      ...0011111100...
FEEDBACK =
IT IS O
INPUT PATTERN
 TIMEX = 296
      ...1111111111...
      ...1111111111...
      ...1100000000...
      ...1100000000...
      ...1111110000...
      ...1111110000...
      ...1100000000...
      ...1100000000...
      ...1111111111...
      ...1111111111...
FEEDBACK =
IT IS E
INPUT PATTERN
 TIMEX = 307
      ...0000110000...
      ...0001111000...
      ...0011001100...
      ...0011001100...
      ...0110000110...
      ...0111111110...
      ...0111111110...
      ...1100000011...
      ...1100000011...
      ...1100000011...
FEEDBACK =
IT IS A
INPUT PATTERN
 TIMEX = 317
      ...0011111100...
      ...0011111100...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0000110000...
      ...0011111100...
      ...0011111100...
FEEDBACK = I
IT IS I
INPUT PATTERN
 TIMEX = 328
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1111111111...
      ...1111111111...
      ...1100000011...
      ...1100000011...
      ...1100000011...
      ...1100000011...
FEEDBACK = H
```