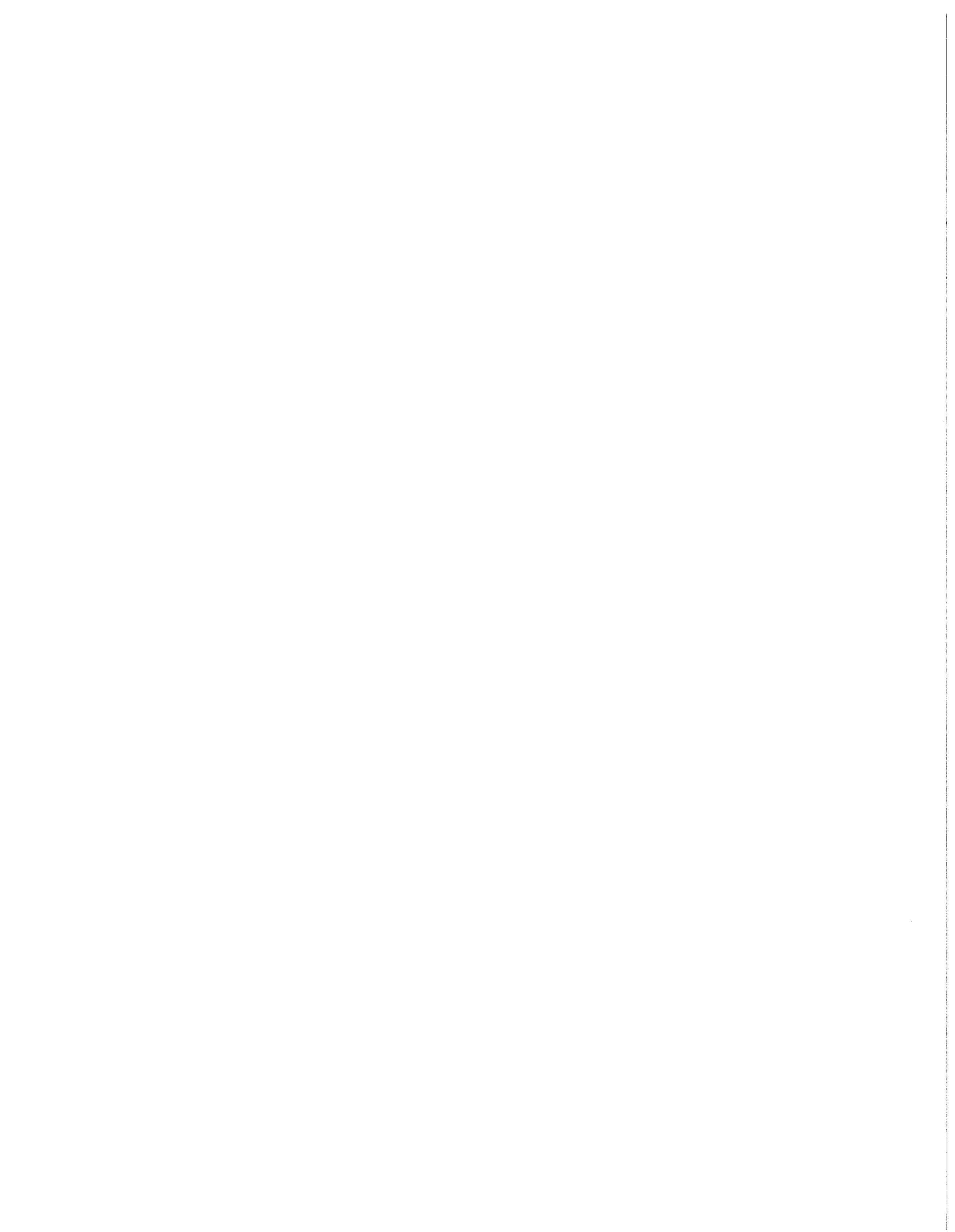
FLEXIBLE PATTERN RECOGNITION

by

Leonard Uhr

Technical Report #56

February 1969



FLEXIBLE PATTERN RECOGNITION

ABSTRACT

This paper presents and describes a sequence of three computer programs that examine what "flexibility" might mean in the context of pattern recognition. Flexibility is a vague, but important, concept, and it is something that artificial intelligence programs have been accused of being without. Various possible meanings of the concept are discussed and programmed. Essentially, flexibility is taken to point to a rich set of methods, which are decided upon and changed, as appropriate. In pattern recognition, this means making a sequence of parallel characterizations, where the program decides, as a function of what it has learned so far about the pattern instance it is trying to recognize, what might be there, and what characterizers should therefore be applied next, and where.

FLEXIBLE PATTERN RECOGNITION

Leonard Uhr

INTRODUCTION

This paper presents a sequence of three computer programs designed to examine what we might mean by "flexibility" in a pattern recognizer. Actual SNOBOL programs and English-language descriptions (called "precis") are given, described, and discussed.

These programs can best be thought of as simulations of highly parallel computers that would be far more appropriate for pattern recognition and, indeed, for most artificial intelligence problems, than are our serial general purpose computers. They also attempt to do pattern recognition the way living animals appear to do it. They gather a little bit of information in the most likely places, from this they infer whether to gather more information, and what kind of information and where, or whether to decide upon a name to output. They thus continue to gather and process information, in a sequential set of parallel programs, whether some characteristic is present somewhere in the pattern. But they flexibly define a characteristic as a threshold element of many pieces, so that many different things will satisfy it, they are flexible as to the position at which the characterizer is to be assessed, and they are flexible as to whether to decide, or to gather more information.

PROBLEM, BACKGROUND, MOTIVATION

Virtually all pattern recognition programs are organized as follows: A) The input pattern is characterized by a set of tests, and B) a name is chosen as a function of the outcome of these tests. Usually step A is parallel, with all other characterizers being examined before the decision is made. (For example, Bledsoe and Browning, 1959; Kamentsky and Liu, 1963; Marrill et al, 1963; Prather and Uhr, 1964; Uhr and Vossler, 1961; and any program that transforms inputs into points in some n-dimensional feature space that is then examined from the point of view of cluster analysis, correlation, or the construction of

separating hyperplanes is parallel. See Uhr, 1963, 1967; and Nagy, 1968.) Occasionally step A is serial, such that the outcome of one characterizer tells the program which characterizer to use next. In all cases known to the author, this is strictly serial, a single characterizer implying which single characterizer to use next. (For example, Bomba, 1959; Feigenbaum, 1959; and all the programs for "concept information" (Hunt, 1962; Kochen, 1961) and "discrimination nets" turn out to do this.)

There are two major reasons why neither the strictly parallel nor the strictly serial is either realistic or satisfactory, and there are several improvements that can be made, to both combine and extend them.

1) The nervous system of living animals, and the information-processing system of man-made computer systems, are rarely if ever strictly parallel or serial; rather they are parallel and serial. A computer system has a set of input devices (e.g., card readers, teletypes), and each device is usually to some extent parallel (e.g., the card reader inputs 80 alphanumeric symbols in parallel.) The nervous system has nerve endings spread all over the surface skin of the organism, with important parallel concentrations here and there (e.g., roughly 10,000,000 receptor rods in each eye). Layered sequences of computations are performed by nervous systems: for information flows back from the skin, into and through the cortex, crossing a number of "synaptic junctions" that seem to serve as extremely large and complex threshold elements to transform their inputs. Computers are usually strictly serial in the way they transfer information along memory banks and process information under the step-by-step control of a central processing unit. But each transfer or computation often involves a whole set of objects worked upon in parallel.

2) Neither a strictly parallel nor a strictly serial method is optimal. Serial methods, since they force information flow through a single path of a tranformation net, each node of this net (a characterizer test) deciding which single node to go to next, suffer from being only as good as their poorest test,

as Selfridge has pointed out (1959). But parellel methods, since they apply all tests, whether or not they are needed, or even pertinent, for any particular problem, can waste large amounts of time and possibly space. In fact the almost universal use of strictly parallel methods in pattern recognition research, where almost all programs first apply all characterizers, and then decide, points a suspicious finger at the toy aspects of the pattern sets upon which they have been tested. For when they are asked to handle real-life problems they will almost certainly fall down (as many artificial intelligence programs fall down) because computers are not big and fast enough to store and perform tests that they will need. Worse - computers, even ideal computers, could never be big and fast enough in the real world of physical particles, with finite limits, such as the speed of light, as opposed to the timeless, finite but potentially infinite world of mathematical machines. Nor could brains be big enough if they used strictly parallel methods.

PROGRAMS FOR FLEXIBLE PATTERN RECOGNITION

Serial-parallel (or if you choose, parallel-serial) programs can rather easily be developed, and these programs seem to benefit from most of the advantages, and few of the disadvantages, of the two methods taken separately.

But several important things can be done in addition to simply combining serial and parallel processing into a single program. In addition to implying one or more names that might characterize an input and implying one or more characterizers that might be applied next to the input, the outcome of a characterizer might also imply <u>what parts</u> of the input to examine next, and what general <u>types</u> of computations - regularizing transformations and parameters of characterizers - to apply. (For example, a program might begin to suspect that this was a fuzzy input, and therefore try to eliminate noise with some averaging operations and sharpen its contours with some differencing operations; or it might suspect that this was a midget input, and magnify it. Or it might decide

to lower the threshold at which all characterizers were considered to succeed; or, in order to handle inputs with curved corners, to lower the threshold at which all angle-assessing characterizers were considered to succeed.)

A program might also give each characterizer the possibility of implying larger whole characterizers, or names, that can themselves imply further characterizations that should be made in order to reach a high enough level of certainty to decide to accept or reject this implication. For example, a characterizer that picked up a short curving contour might imply two more complex whole loop characterizers (one for a "D" and one for an "O") that would succeed if this plus several other characterizers succeeded, and therefore tell the program to test out these other characterizers. Or a vertical-line characterizer might imply "E," "F," "D," etc., and the program would therefore be asked to test out all the other characterizers that imply each of these particular letters or all of these particular letters.

Finally, once a program has been given such abilities, it should, and can, be asked to treat what we typically think of as the names that a pattern recognition program is to output (for example, "A," "B," "plane," "table," "gene,") as inputs for further pattern recognition. Now a program might decide upon a "D" and a nearby "G," transform the input into these symbols, and then decide to consider "DOG" or "DIG", which leads it to choose and apply further characterizers. Thus the previous attitude and suggested extensions lead to the idea of a heirarchical pattern recognizer that processes continuous, contextually interrelated fields of more than one, heirarchically-organized, patterns. (Some attempts have been made to handle such problems for one-dimensional strings of natural language using methods rather different from those presented in the present paper; see Uhr (1964), Sauvain and Uhr (1968), Siklossy (1968), and Klein (1968).

FPR-1 - SIMPLEST FLEXIBLE PATTERN RECOGNITION

The following Program ("FPR-1," for Flexible Pattern Recognition-1) embeds one of the central aspects of "flexibility" in the simplest interesting set of procedures for pattern recognition that I have been able to find. Essentially, this program characterizes a pattern with a set of "piece-templates." The piece-template is a string of symbols that, when it matches some substring of the pattern, implies one or more possible pattern names, each with an associated weight. This program applies each piece-templates in turn, gets the implied names of those that successfully match the input, combines the weight of all these implications, and chooses the first name whose sun of weights exceeds a "DECIDE" level.

[There are a large number of simple variations that can be made on programs of this sort (See Uhr, 1969a, 1969b). For simplicity, this program reads patterns in as 1-dimensional strings; but they could easily be handled in two dimensions, as we will see in the next program. Piece-templates are considered to be matched if found anywhere; but they might be restricted to a certain point, or area, of the input, as in subsequent programs. Weights are combined by simple summing, but more sophisticated functions could be used. The program decides to output the first name whose sum of weights exceeds the level for deciding; but it might insist that the chosen name be implied sufficiently more highly than any of the other possible names]

Flexibility is introduced into this rather traditional pattern recognition program by having a characterizer imply not only a set of output names, but also a set of "ACTS" that the program should effect. These acts will simply be some more characterizers to apply. Now, instead of simply going through a set of characterizers that will be applied to all inputs, the program must add the implied acts to the set of characterizers it is looking for. Thus statement 7 of Program FPR-1 is the major variation on the traditional pattern recognition prototype.

*PRECIS FPR-1. SIMPLEST PROGRAM IN WHICH CHARACTERIZERS THAT ARE FOUND
*IMPLY OTHER CHARACTERIZERS TO BE LOOKED FOR. FPR-1

GO	Let ATTEND contain the names of Primitive characterizers	M1
	Let each PI (name of a Primitive characterizer) contain its Description, Implied names, and implied Acts.	M1.1
	Let each CI (name of a Compound characterizer) contain its Description, Implied names, and implied Acts.	M1.2
	Let DECIDE (the level at which the program will decide to choose a name to output) equal 20.	M2
INIT	Erase FOUND.	1
IN	READ in the next INPUT (all on one line). (If no more, go to END.)	2
INITCHAR	Let LOOKFOR contain the names of the characterizers on ATTEND.	3
CHARACTERIZE	Get the next CHARacterizer from LOOKFOR. (If no more, go to O1.)	4
	Get the DESCRiption, IMPLIEDS, and ACTS for this CHARacterizer.	5
	Look for the DESCRiption in the INPUT (anywhere). (If Fail to find it, go to CHARACTERIZE.)	6
TODO	Add any ACTS implied by this characterizer to the end of LOOKFOR.	7
IMPLY	Get the next NAME and its WeighT from IMPLIEDS. (If no more, go to CHARACTERIZE.)	8
	If this NAME is on FOUND, remove it, and its SUM of weights. (If not, go to I1.)	9
	Add this SUM to the WeighT (WT).	10
	If this sum of WeighTs (WT is GreaterThan DECIDE, go to OUT.)	11
I1	Put the NAME and its SUM on FOUND. Go to IMPLY.	12
OUT	PRINT out the NAME decided on. Go to INIT.	13
O1	PRINT out 'MIGHT BE- ' followed by whatever possible names have been put on FOUND. Go to INIT.	14
END	GO	--

This program differs from traditional pattern recognition programs only in that statement 7 puts any ACTS implied by the successful characterizer onto LOOKFOR, so that the new characterizers stored in ACTS will be looked for in their turn. This entails the use of the LOOKFOR list, which initially contains only what is in the primitive MEMORY, and the addition of the acts (signalled by 'A/=' on the list of information for each characterizer) and the augmented match of statement 5, which pulls them off the characterizer's list.

If the program put the ACTS at the beginning, rather than at the end, of LOOKFOR, its behavior would be radically different. It would be following up new leads and hence going in new directions immediately, rather than making them wait their turn. A better program might have a number associated with each characterizer on MEMORY and in each characterizer's list of ACTS that would reflect its value. Then characterizers could be ordered on LOOKFOR from highest to lowest value, so that the program would apply them in that order. The program should also check to see if a characterizer is already on LOOKFOR, so that it does not add it twice. (If it used values, it could then merge, for example by adding or averaging, the several values for the same characterizer.)

```
*PROGRAM FPR-1.  SIMPLEST PROGRAM IN WHICH CHARACTERIZERS IMPLY OTHER         1FPR-1
*CHARACTERIZERS TO LOOK FOR.
GO             ATTEND = 'P1,P2,P3,...PN,'                                     1M1
               P1 = 'D=01110/I=B-6,F-3,/A=C7,C12,/'                           M1.1
               C7 = 'D=1000010001/I=B-7,F-2,/A=C15,/'                         M1.2
               DECIDE = '20'                                                  M2
INIT           FOUND =                                                        1
IN             .READ *INPUT* '    ' /F(END)                                   2
INITCHAR       LOOKFOR = ATTEND                                               3
CHARACTERIZE   LOOKFOR *CHAR* ',' = /F(O1)                                    4
               $CHAR 'D=' *DESCR* '/I=' *IMPLIEDS* '/A=' *ACTS* '/'           5
               INPUT  DESCR  /F(CHARACTERIZE)                                 6
TODO           LOOKFOR = LOOKFOR ACTS                                         7
IMPLY          IMPLIEDS *NAME* '-' *WT* ',' = /F(CHARACTERIZE)                8
               FOUND '/' NAME '-' *SUM* ',' = /F(I1)                          9
               WT = WT + SUM                                                  10
               .GT(WT, DECIDE)  /S(OUT)                                       11
I1             FOUND = FOUND '/' NAME '-' WT ',' /(IMPLY)                     12
OUT            .PRINT = 'THE PATTERN IS- ' NAME  /(INIT)                      13
O1             .PRINT = 'MIGHT BE- ' FOUND /(INIT)                            14
END            GO                                                             -
```

[1]NOTES: Numbers refer to Program named at the top of the column.
M1 signifies Memory statement 1. .1, .2 signify inserted statements.
.A signifies an altered statement.

FPR-2. FLEXIBLE RECOGNITION OF 2-DIMENSIONAL INPUTS USING CONFIGURATION OF PIECE-TEMPLATES.

We will now look at a program that inputs instances of patterns to be named that are actually presented, stored, and processed in 2-dimensional form. Thus this program allows for realistic handling of the real 2-dimensional problem. The program also characterizes patterns by looking for matches of whole configurations of piece-templates, that is, of sets of strings of symbols, where these strings are to be matched in specified positions. Each string has an associated FIRE weight, and a name is implied if the sum of weights of found strings exceeds its threshold. This is actually a relatively powerful kind of characterizer.

Though simple to code and to understand, it appears to work about as well as any other kind, in existing pattern recognition programs.

The program also reads in feedback as to the correct name it should have chosen to output, and adjusts its DECIDE level as a function of this feedback: If the program wrongly chose a wrong name, DECIDE is raised, so that the program will be forced to collect more information before it makes decisions in the future, to be less brash. If the program couldn't decide at all, DECIDE is lowered, so that less information will be needed. This is a very simple form of "learning" as a function of feedback, but it is so intimately related to flexibility, and so easy to demonstrate, that it seems appropriate to present it here.

The basic flexibility of characterizers implying not only names, but also other characterizers, is handled exactly as in the first program, except that characterizers are TRIED only once and, for variety, the implied ACTS that the program is to LOOK FOR, along with the original set of characterizers it was told to ATTEND are added to the beginning, not the end, of the ATTEND list.

```
RECIS FRP-2.  SUCCESSFUL CHARACTERIZERS (2-DIMENSIONAL SETS OF POSITIONED        F-PR-2
IECE-TEMPLATES) IMPLY CHARACTERIZERS TO LOOK FOR.
             Let ATTEND contain the names of characterizers to attend to first.   M1
             Let P1,..,PN contain the information about chars on ATTEND.          M1.1
             Let C1,..,CN contain the information about chars pointed to.         M1.2
             Let the level at which the program DECIDES equal 20.                 M2
T            Initialize FOUND to be empty, ROW to equal 0.                        1-2
             READ in the next LINE of the input. (If no more, go to END.)         3
             If LINE contains '****' get the FeedBacK, and go to INITCHAR         4
                 (this input has been completed).
             Or Add 1 to ROW.                                                     5
             Store this LINE under this ROW number as its name.  Go to IN.        6
E PATTERN HAS BEEN INPUT.  START TRYING TO RECOGNIZE IT.
TCHAR        Let LOOKFOR contain the names of characterizers on ATTEND,           7
                 to start.
RACTERIZE    Get the next CHARacterizer from LOOKFOR. (If no more, Go to O1.)     8
             Get this CHARacterizer's DESCRiption, IMPLIEDS (pattern names),      9
                 and ACTS (other characterizers it implies should be tried).
             Blank our FIRE.                                                      10
             Get the next PIECE from this DESCRiption, along with its ROW         11
                 and COLumn positions, and WeighT. (If no more, go to TODO.)
```

		F-PP-2
	Look for this PIECE in the designated ROW and starting at the designated COLumn. (If Fail, go to C1 to continue to process this characterizer.)	12
TODO	Add the WeighT of this PIECE to total FIRED. Go to C1.	13
	Put all ACTS implied by this successful characterizer at the start of the list of things to LOOKFOR.	14
IMPLY	Get the next THreshold, NAME and its WeighT from the list of names IMPLIEDS by the characterizer. (If no more, go to CHARACTERIZE.)	15
	Is FIRED GreaterThan THreshold? No - Fail to IMPLY.	16
	Yes - Look for this NAME on the list of FOUND names and, if it is found, remove it, along with its associated SUM of weights. (If Fail, go to I1.)	17
	Add the SUM of weights to the WeighT for this implication.	18
	Is this new (sum of) WeighTs (WT) GreaterThan the number in DECIDE? Yes - Go to Out.	19
I1	Put this NAME and its WeighT (which may be a sum of weights) on FOUND. Go to IMPLY.	20
OUT	PRINT out the NAME that has been decided on.	21
	See if the NAME EQUALS the FeedBacK (correct) name. Yes - go to INIT.	22
	No - Add 1 to DECIDE (so the program will consider more information the next time). Go to INIT.	23
O1	If no name has been implied with a sum of weights above DECIDE, print out that there were no strong implications, and print whatever is on FOUND.	24
	Subtract 1 from DECIDE so that the program will be able to decide a bit more quickly the next time. Go to INIT.	25
END	GO	--

```
*PROGRAM FPR-2.  SUCCESSFUL CHARACTERIZERS (2-DIMENSIONAL SETS OF POSITIONED
*PIECE-TEMPLATES) TO LOOK FOR.  LEARNS WHEN TO 'DECIDE.'  MATCHES OVER
*THRESHOLD.                                                        FPR-1    FPR-2
GO             ATTEND = 'P1,P2,P3,...,PN,'                          M1       M1
               P1 = 'D=01111*2-3*4,01100*4-3*2,/I=3*B-7,*E-5,/      M1.1A    M1.1
.                A=P7,C7,C12,/'
               C7 = 'D=1001*3-4*3,010*7-2*4,/I=3*B-6,2*P-7,/        M1.2A    M1.2
.                A=C9,C29,C81,/'
               DECIDE = '20'                                        M2       M2
INIT           FOUND =                                              1        1
               ROW = '0'                                            1.1      2
*READ IN THE INPUT PATTERN, LINE BY LINE.
IN             .READ *LINE* '    ' /F(END)                          2.A      3
               LINE '***' *FBK* /S(INITCHAR)                        2.1      4
               ROW = ROW + '1'                                      2.2      5
               $('R.' ROW) = LINE /(IN)                             2.3      6
*APPLY THE CHARACTERIZERS TO THE INPUT PATTERN.
INITCHAR       LOOKFOR = ATTEND                                     3        7
CHARACTERIZE   LOOKFOR *CHAR* ',' = /F(O1)                          4        8
               $CHAR 'D=' *DESCR* '/I=' *IMPLIEDS* '/A=' *ACTS* '/' 5        9
               BLANK *FIRE*                                         5.1      10
C1             DESCR *PIECE* '*' *ROW* '-' *COL* '*' *WT* ',' =     5.2      11
.              /F(TODO)
               $('R.' ROW) *LEFT/COL* PIECE /F(C1)                  6.A      12
               FIRE = FIRE + WT /(C1)                               6.1      13
TODO           LOOKFOR = .GE(FIRE,'1') ACTS LOOKFOR                 7.A      14
```

```
                                                                              FPR-1   FPR-
IMPLY   IMPLIEDS *TH* '*' *NAME* '-' *WT* ',' = /F(CHARACTERIZE)              8.A     15
        .GT(FIFE,TH) /F(IMPLY)                                                8.1     16
        FOUND '/' NAME '-' *SUM* ',' = /F(I1)                                 9       17
        WT = WT + SUM                                                         10      18
        .GT(WT,DECIDE) /S(OUT)                                                11      19
I1      FOUND = FOUND '/' NAME '-' WT ',' /(IMPLY)                            12      20
OUT     .PRINT = 'THE PATTERN IS- ' NAME                                      13.A    21
        EQUALS(FBK,NAME) /S(INIT)                                             13.1    22
        DECIDE = DECIDE + '1' /(INIT)                                         13.2    23
01      .PRINT = 'NO STRONG IMPLIC, WEAK ONES = ' FOUND                       14.A    24
        DECIDE = DECIDE - '1' /(INIT)                                         14.1    25
END     GO                                                                            -
```

FPR-3. FLEXIBLE RECOGNITION WITH HEIRARCHICAL, RELATIVELY-POSITIONED CHARACTERIZERS

Rather than merely imply which further characterizers should be applied to the input pattern, a characterizer that succeeds might also imply where these characterizers should be applied. A found characterizer might also be a part of a heirarchically higher-level characterizer. In order to handle this efficiently, it is now necessary for the program to put the names of the found characterizers in the input, so that higher-level characterizers can refer and look for these lower-level characterizers by name only. A piece of a configurational characterizer can now be another characterizer's name. Implied names now imply other characterizers than imply them, and those that have not as yet been TRIED are added to the LOOK FOR list, so that the program will look further into the conjecture that the input is of the kind named. A more sophisticated program would use weights associated with these names to merge them into the LOOK FOR list, and further choose to apply characterizers that will be most instrumental in deciding among the several most highly implied names.

Program FPR-3 also modifies the basic ATTEND list of the characterizers that it should start looking for in each input, by moving characterizers that matched to the beginning of ATTEND. [A more sophisticated program might move the characterizer only a little bit toward the beginning, or in some other way re-order characterizers as a function of matching, and/or of feedback as to the helpfulness of this match. Quite a bit more difficult would be to have the program decide to add characterizers to the ATTEND list, and to take them off - for this would entail checking whether a characterizer was a part of some

higher-level characterizer, so that the proper order of applying characterizers would always be observed.]

This program handles either positioned or unpositioned configurations, and their pieces. It makes partial threshold matches. [A better program would (as shown in Jordan and Uhr, 1969) allow for a certain amount of wobble in the matching of positioned pieces. It would also be extremely easy to have parts of characterizers be names of subroutines or functions that could compute any arbitrary characteristic for which the code was written (e.g., counts of line crossings, angle detectors, edge, curve and loop detectors).]

```
'PROGRAM FPR-3.  HEIRARCHICAL CHARACTERIZERS IMPLY RELATIVELY-POSITIONED WHOLES
'TO LOOK FOR NEXT.                                                          FPR-3
GO            Let ATTEND contain the Primitive characterizers, each, optionally, M1
                 with a position (relative to the upper-left of the matrix)
                 attached to it.
              Let P1 (and P2,P3,...) contain information as to its Description,  M2.1
                 what, if anything, it Implies, and the Acts it implies.
              Let C1(C2,C3,...), which is a Compound characterizer the program   M3.1
                 applies only when it has been implied as an Act by some other
                 characterizer, contain its Description, Implieds and implied Acts.
              Let A(B,C,...,Z) contain the Primitive characterizers that imply it.M4.1
              Let the level to DECIDE equal 20.                                  M5
INIT          Erase FOUND (this could be combined with the instruction below).   1
              BLANK out anything in ROW and TRIED.                               2
IN            READ in the next LINE of the input pattern.  (If no more, go to    3
                 END.)
              If this LINE contains '***' get the FeedBacK, which follows, and   4
                 go to INITCHAR.
              Or add 1 to ROW.                                                   5
              Store this LINE as the contents of this ROW.  Go to IN.            6
INITCHAR      Let LOOKFOR contain whatever characterizers (the primitives) are   7
                 stored on ATTEND.
CHARACTERIZE  Get the next CHARacterizer and its POSition (which may be null)    8
                 from LOOKFOR.  (If no more, go to O1.)
              Add this CHARacterizer to the list of those TRIED.                 9
              If POSition contains a '-', let AROW contain what precedes the     10
                 '-' and ACOL contain what follows it, and go to C2.
              Or let AROW contain 'ANYWHERE' (unless a position is given, the    11
                 program will try to match the characterizer anywhere).
C2            Get the DESCRiption, IMPLIEDS, and ACTS for this CHARacterizer.    12
              BLANK out FIRE, WeighT.                                            13
              Add WeighT to FIRE                                                 14
C1            Get the next PIECE, and its DROW and DCOLumn from the DESCRiption. 15
```

```
             (DROW and DCOL, which give the difference in row and in
               column, may be null).  If fail, go to REORDER.
           Let DROW equal AROW plus DROW and, if succeed, go to C4.          16
           Or, if fail (because program couldn't add 'ANYWHERE', which      17
               is not a number), Let R equal zero.
C3         Is R LessThan ROW?  No - Go to CHARACTERIZE (the entire input has 18
               been searched, and this characterizer was not found).
           Add 1 to R.                                                        19
           Look for this PIECE in this ROW.  (If fail, go to C3.)           20
           If found, let AROW equal R.                                        21
           See if '/(' is found immediately to the RIGHT of the piece that   22
               was found (which means it is a name that was inserted by the
               program, with its position as a subscript, during this run),
               get ACOL, which designates the position of this piece, and
               go to C1.
           Or let ACOLumn equal the SIZE of the part of this ROW to the      23
               LEFT of the piece that matched.  Go to C1.
*LOOK FOR A POSITIONED PIECE, RATHER THAN LOOKING ANYWHERE.
C4         Let DCOLumn equal ACOLumn plus DCOLumn.                           24
           Look in the DROW specified for the PIECE immediately followed by  25
               the specified DCOL as its subscript.  (If Succeed, go to C1).
           Or look in the DROW specified at the DCOLumn specified (from the  26
               left) for the PIECE.  If Succeed, go to C1, to get the next
               PIECE of this characterizer.  If Fail, go to CHARACTERIZE, to
               get the next characterizer.
REORDER    See if this CHARacterizer contains 'P' (which means it is a       27
               Primitive).  If not, go to TODO.
           Yes - Put it at the beginning of ATTEND.  (Thus primitive         28
               characterizers that are found will be moved to the front of
               ATTEND and therefore attended to more.)
TODO       Get the next CHARacterizer from ACTS.  (If Fail, go to ADDFOUND.) 29
           Get the RelativeRow and RelativeCOLumn from this CHARacterizer    30
               (if given), and add them to DROW and DCOLumn, respectively,
               to position where this CHARacterizer as a whole should be
               looked for.
           If this CHARacterizer has already been TRIED, go directly to      31
               TODO.
           Add this CHARacterizer to the end of LOOKFOR.  Go to TODO.        32
ADDFOUND   Put this found CHARacterizer into the input, in the DROW          33
               specified, at the end, subscripted by '(DCOLumn)'.
*THE REST OF THE PROGRAM IS IDENTICAL TO STATEMENTS 15-25 OF FPL-2,
*EXCEPT THAT ALL CHARACTERIZERS OF AN IMPLIED NAME ARE PUT AT THE
*START OF LOOKFOR, (IF THEY HAVEN'T BEEN TRIED ALREADY).  (STATEMENTS 37-40.)
```

```
*PROGRAM FPR-3. HEIRARCHICAL CHARACTERIZERS IMPLY RELATIVELY-POSITIONED
*WHOLES TO LOOK FOR NEXT.                                                  FPR-2    FPR-3

GO              ATTEND = 'P1*2-3,P2*P3*,..,PN*'                                     M1
*PRIMITIVE CHARACTERIZERS CALLED PI, OTHERS CALLED CI.
                P1 = 'D=0110*2-3*2,01*4-4*2,1111*5-2*3,/I=3*E-1,/A=C7*,             M2.1..
.                  C12*3-4,/'
                :
                :                                                                   M2.N
                C7 = 'D=C3*-*3,C2*2-3*4/I=4*E-3,5*F-5,/A=C12*,/'                    M3.1..
                :
                :                                                                   M3.N
*THE PRIMITIVES IMPLYING IT ARE STORED UNDER EACH NAME.                             M4.1..
                A = 'P1*2-3,P3*,P8*,'                                               M4.N
                DECIDE = '20'                                              M4       M5
INIT            FOUND =                                                    1        1
                BLANK *ROW* *TRIED*                                        2        2
IN              .READ *LINE* '    '  /F(END)                               3        3
                LINE '****' *FBK* /S(INITCHAR)                             4        4
                ROW = ROW + '1'                                            5        5
                $('R.' ROW) = LINE /(IN)                                   6        6
INITCHAR        LOOKFOR = ATTEND                                           7        7
CHARACTERIZE    LOOKFOR *CHAR* '*' *POS* ',' = /F(01)                      8        8
                TRIED = TRIED  CHAR ','                                             9
                POS *AROW* '-' *ACOL* = /S(C2)                                      10
                AROW = 'ANYWHERE'                                                   11
C2              $CHAR 'D=' *DESCR* '/I='  *IMPLIEDS* '/A=' *ACTS* '/'      9        12
                BLANK *FIRE* *WT*                                          10.A     13
C1              FIRE = FIRE + WT                                           13.A     14
                DESCR *PIECE* '*' *DROW* '-' *DCOL* '*' *WT* ',' = /F(TODO)11.A     15
                DROW = AROW + DROW /S(C4)                                           16
*ADDITION FAILS WHEN AROW = 'ANYWHERE' (WHICH IS NON-NUMERIC).  LOOKS ANYWHERE.
                R = 0                                                               17
C3              .LT(R,ROW) /F(CHARACTERIZE)                                         18
                R = R + '1'                                                         19
                $('R.' R) *LEFT*  PIECE *RIGHT* /F(C3)                     12.A     20
                AROW = R                                                            21
                RIGHT ANCHOR() '/(' *ACOL* ')' /S(C1)                               22
                ACOL = SIZE(LEFT) /(C1)                                             23
*LOOK FOR A POSITIONED PIECE.
C4              DCOL = ACOL + DCOL                                                  24
                $('R.' DROW) PIECE '/(' DCOL ')'   /S(C1)                           25
                $('R.' DROW) ANCHOR () *LEFT/DCOL* PIECE /S(C1)F(CHARACTERIZE)      26
*CHARACTERIZER SUCCEEDED.  MOVE IT UP ON ATTEND.  ADD THINGS TODO TO LOOKFOR.
REORDER         CHAR 'P' /F(TODO)                                                   27
                ATTEND *LEFT* CHAR '*' *REST* ',' = CHAR '*' REST ',' LEFT          28
TODO            ACTS *CH* ',' = /F(ADDFOUND)                                        29
*COMPUTES AND SPECIFIES RELATIVE POSITION (IF GIVEN) OF CHARACTERIZER POINTED TO.
                CH '*' *RROW* '-' *RCOL* = '*' RROW + DROW '-' RCOL + DCOL          30
                TRIED CH ',' /S(TODO)                                               31
                LOOKFOR = LOOKFOR CH ',' /(TODO)                           14.A     32
*ADD 'CHAR' (THE FOUND CHARACTERIZER'S NAME), AND ITS POSITION, TO THE INPUT.
ADDFOUND        $('R.' DROW) = $('R.' DROW) CHAR '/(' DCOL ')'                      33
IMPLY           IMPLIEDS *TH* '*' *NAME* '-' *WT* ',' = /F(CHARACTERIZE)   15       34
                .GT(FIRE,TH)  /F(IMPLY)                                    16       35
```

```
                                                                FPR-2   FPR-3
         FOUND '/' NAME '-' *SUM* ',' = /S(I2)                   17.A     36
*PUT THE PRIMITIVES THAT IMPLY THIS NAME ONTO LOOKFOR.
         $NAME *CHAR*                                                     37
I3       CHARS *CH* ',' = /F(I2)                                          38
         TRIED CH ',' /S(I3)                                              39
         LOOKFOR = CH ',' LOOKFOR /(I3)                                   40
I2       WT = WT + SUM                                           18.A     41
         .GT(WT, DECIDE) /S(OUT)                                 19       42
I1       FOUND = FOUND '/' NAME '-' WT ',' /(IMPLY)              20       43
OUT      .PRINT = 'THE PATTERN IS- ' NAME                        21       44
         EQUALS(FBK,NAME)   /S(INIT)                             22       45
         DECIDE = DECIDE + '1' /(INIT)                           23       46
O1       .PRINT = 'NO STRONG IMPLIC. WEAK ONES = ' FOUND         24       47
         DECIDE = DECIDE - '1' /(INIT)                           25       48
END      GO                                                               -
```

DISCUSSION

What else should a "flexible" pattern recognition program do, and what else might we mean by "flexibility" in this context? It seems proper to ask this question now, for our situation is quite clear-cut, and the issues seem to be relatively simple and well-delineated. The programs we have just examined introduce flexibility and possibilities for variation at just about every point in their memory structures and their flow of processing. Should they have more structure and more decision points? Flexibility seems to boil down to peppering the program with decision points as to each aspect of what it is doing, so that it can and will at any moment change its direction of processing as a reflection of information it has gathered up to that point in its processing. Flexibility is thus closely related to learning. As the program learns about this input that it is trying to recognize it continually assesses and changes its tactics for recognition. It is self-reflective (self-conscious?) in that it decides not only about what pattern name to output, as is the case with most pattern recognition programs, but also about the various aspects of what to do next.

But this is not true learning, which modifies a program or an organism so that the next time it processes the same or for that matter other inputs it will behave differently. For the flexible programs erase all of the information that they have temporarily built up about the input, re-initialize all their

starting parameters and lists, and process subsequent inputs in exactly the same way no matter what inputs were processed before. A variety of learning techniques - for induction, hypothesis generation and discovery, and parameter adjusting and discovery - might be added to increase further the flexibility of these programs. (See Jordan and Uhr, 1969, for a program that incorporates a large number of learning mechanisms.)

With the exception of learning, I am at a loss as to what else to add, or where else to go with these programs, in order to make them more "flexible." This is not at all to say that they exhibit the ultimate in flexibility, or even that this is what we commonly mean, or ought to mean, by flexibility. Rather the purpose of this paper is to pose the question of flexibility in a clear and concrete manner, present some answers that have occurred to me, and most important, urge, challenge, and seduce the reader into taking up the problem and finding more and better answers.

SUMMARY

This paper presents and discusses a series of successively more complex computer programs that: 1) use a parallel-serial organization of characterizers, 2) decide <u>where</u> to look for <u>what type</u> of characterizer as a function of the outcome of previous tests, 3) change their level of assurance as to when to decide, 4) characterize patterns using loosely positioned, partially matching piece-templates, 5) develop conjectures as to what output names, or higher-level characterizers might be appropriate, and therefore make further tests designed to establish or deny these conjectures, and 6) heirarchically recognize things that are themselves composed of previously-recognized things. Actual running computer programs are presented to exhibit some, but not all of the features that programs to handle the above problems might have. They are designed to be as simple as possible, and they are presented in a high-

level pattern-matching and list-processing language (SNOBOL) that makes communication, and coding and debugging, as simple as possible. They run extremely slowly on the computer, and for extensive tests to determine their power they should be recoded into a faster-running language. But they appear to give a relatively good coverage of, and make quite clear, the above problems and the methods that can be used to attack them.

What else might we mean by "flexible" in the context of pattern recognition? What other functions should a computer program be given? Pattern recognition seems to be a relatively simple and straightforward problem area, but one in which a concept like "flexibility" has pertinence and meaning. It would be very nice to exhaust all the possibilities for "flexibility," to code them, and to thus demonstrate what this concept might mean. It would be at least as interesting to discover meaningful aspects of flexibility that could not be coded.

BIBLIOGRAPHY

Bledsoe, W. W. and Browning, I., Pattern recognition and reading by machine. Proc. Eastern Joint Comp. Conf., 1959, 225-232.

Bomba, J. S., Alpha-numeric character recognition using local operations, Proc. Eastern Joint Comp. Conf., 1959, 218-224.

Feigenbaum, E., The simulation of verbal learning., Proc. Western Joint Computer Conf., 1961.

Hunt, E. B., Concept Formation: An Information Processing Problem. New York: Wiley, 1962.

Kamentsky, L. A. and Liu, C. N., Computer-automated design of multi-font print recognition logic. IBM J. of Research and Devel., 1963, 7, 2-13.

Klein, S., Fabens, W., Herriot, W., Katke, R., Kuppin, M., and Towster, The AUTOLING System, U. W. Comp. Sci. Dept. Tech. Report No. 43, 1968.

Kochen, M., An experimental program for the selection of "disjunctive hypotheses," Proc. Western Joint Computer Conf., 1961.

Marrill, T. et al., Cyclops-I: a second generation recognition system., Proc. Fall Joint Computer Conf., 1963.

Nagy, G., State of the art in pattern recognition, Proceedings of the IEEE, 1968, 56.

Prather, Rebecca and Uhr, L., Discovery and learning techniques for pattern recognition., Proc. 19th Annual Meeting of the ACM, 1964.

Sauvain, R. and Uhr, L., A teachable pattern describing and recognizing program, Pattern Recognition, 1969, in press.

Selfridge, O. G., Pandemonium: a paradigm for learning., In: Mechanization of Thought Processes (D. V. Blake and A. M. Uttley, Eds.), London: H. M. Stationary Office, 1959.

Siklossy, L., Natural Language Learning by Computer, Unpublished Doctoral Dissertation, Pittsburgh: Carnegie-Mellon Univ., 1968.

Uhr, L., Pattern-string learning programs, Behav. Sci., 1964, 9, 258-270.

Uhr, L., (Editor) Pattern Recognition, New York: Wiley, 1967.

Uhr, L., Pattern recognition computers as models for form perception. Psychol. Bull., 1963, 60, 40-73.

Uhr, L., A tutorial description of pattern recognition programs, Paper submitted for publishing - 1969.

Uhr, L., _Pattern Recognition, Problem-Solving, and Learning_. 1969b. (In preparation.)

Uhr, L. and Jordan, Sara., The learning of parameters for generating compound characterizers for pattern recognition, submitted for publication, 1969.

Uhr, L. and Vossler, C., A pattern recognition program that generates, evaluates, and adjusts its own operators, _Proc. Western Joint Computer Conf._, 1961, 555-569.