Howard Richman
October 19, 1987

EPAM-V DEFINITIONS

NET

This version of EPAM-V utilizes one net named UO. The assumption of one net is not built into the coding. At a later time it would be possible to utilize EPAM-V with several nets, but at present -- the variable <REALM> functions to split the single EPAM-V net into what could just as easily be implemented as several different nets.

OBJECT

There are two kinds of objects in EPAM-V, "simple" objects and complex objects. An EPAM-V "simple" object has the following form:

```
SIMPLE
OBJECT   9-1              0

9-1      0
         <REALM>
         REALM A
         <ATTRIBUTE 2>
         VALUE 2
         ...
         <ATTRIBUTE N>
         VALUE N          0
```

An EPAM-V "complex" object has the following form:

```
COMPLEX
OBJECT   9-1
         SUBOBJECT 1
         SUBOBJECT 2
         ...
         SUBOBJECT N    0

9-1      0
         <REALM>
         REALM A
         <ATTRIBUTE 2>
         VALUE 2
         ...
         <ATTRIBUTE N>
         VALUE N          0
```

Notes:

1. Subobjects are optional. An EPAM-V object that does not have a subobject on it is called a "simple" object. It corresponds to the "primitive" object with no subobjects in EPAM III. An object with more than one subobject on it is called a "complex" object.

2. The various attribute-value pairs on the description list 9-1 are all optional except for <REALM>. All objects must have a REALM. If EPAM-V were to be implemented with several nets, the <REALM> attribute

could be replaced with a <NET> attribute. Some possible REALMS are "semantic", "phonetic", "written", "English", "French", "Russian", "letter", "word", "sentence", and so on... A <REALM> is any system of complex units which are made up of simpler units and which can be translated into another system of complex units. Letter features are probably not a REALM because it is assumed that they cannot be broken down into subunits. However, letter features are the simple units in the LETTER REALM, letters are the simple units in the WORD REALM, and words are the simple units in the SENTENCE REALM. The purpose of an EPAM-V net is to translate from objects of one REALM to objects of another.

3. A subobject is itself an object. Unless otherwise specified it is assumed to have the same <REALM> as the object.

TEST NODES

There are two kinds of test node in EPAM-V. First, here is an illustration of a simple test node:

```
SIMPLE
 TEST
 NODE      9-1             0

 9-1       0
           <TEST>
           SIMPLE TEST A
           <PATTERN>
           NIL
           IF VALUE 1
           BRANCH TO 9-2
           IF VALUE 2
           BRANCH TO 9-3
           ...
           IF <NOT ELSEWHERE CLASSIFIED>
           BRANCH TO 9-4       0
```

Notes:

1. There are two kinds of test nodes in EPAM-V, both of which are, EPAM-III n-ary nodes, rather than EPAM-II binary nodes.

2. The top nodes in the EPAM-V net (i.e. the first nodes sorted) are the "simple" tests. Two kinds of simple tests occur, these are "property" tests and "simple object" tests. The "property" tests occur first at the top of the EPAM-V net. These invariably include tests for <REALM> and <CLASS>. ~~REALM is related to the kind of input be it semantic, written, spoken, French, Spanish, letter features, phonemes, and so on~~. The value of a property test is found through the IPL-V command J10 which finds the value (of the attribute being tested) on the description list of the OBJECT.

3. Just below the "property tests" are the "simple" object tests. There are two such tests: <NUMBER OF SUBOBJECTS> and <NAME OF OBJECT>. If there are no subobjects, an object will be considered a SIMPLE

OBJECT and will be sorted to the <NAME OF OBJECT> test. <NAME OF OBJECT> is the last test in the net for a simple object. At present, a "simple" object in EPAM-V only passes through four test nodes: <REALM>, <CLASS>, <NUMBER OF SUBOBJECTS> and <NAME OF OBJECT>. Hence, in EPAM-V there can only be one translation for a simple object of a certain REALM given a certain class. Note, however, that this translation can be augmented or left out of the translation of a complex object which contains the simple object as one of its subobjects. Also note that each simple object is likely to be represented in a number of classes.

4. The commands BRANCH TO 9-2, BRANCH TO 9-3, and BRANCH TO 9-4, tell the system to branch to the node called 9-2, 9-3, or, 9-4. The node encountered there will either be a TEST NODE or a LEAF NODE. If VALUE 1 is the result of TEST A then the next node that the sorting routine would go to would be node 9-2.

5. The attribute <PATTERN> asks what pattern is sorted at this node. The value "NIL" means that no pattern is sorted at this node since this is a TEST NODE not a LEAF NODE. Both "NIL" and <NOT ELSEWHERE CLASSIFIED> are represented as the IPL-5 regional K10.

```
    COMPLEX
       TEST
       NODE      9-1
                 CLASS I
                 CLASS J
                 ...
                 CLASS K       0

       9-1       0
                 <TEST>
                 COMPLEX TEST A
                 <PATTERN>
                 NIL
                 IF CLASS I
                 BRANCH TO 9-2
                 IF CLASS J
                 BRANCH TO 9-3
                 ...
                 IF CLASS K
                 BRANCH TO 9-4
                 IF <NOT ELSEWHERE CLASSIFIED>
                 BRANCH TO 9-5      0
```

1. Complex tests are based on complex patterns. They are much different from any tests ever before included in an EPAM model. They take their inspiration from the method used by Siklossy (1968) to use syntactic patterns in order to match semantic sentences with natural language sentences within a "translation rule" based system. These tests function much as the productions do in a production system.

2. In order to sort a complex object at a complex test node, first the particular subobject that will be sorted must be identified. COMPLEX TEST A identifies which subobject will be sorted at this node.

In the first implementation there are three complex tests: (1) NEXT SUBOBJECT (which in IPL-V includes a test for the first subobject), (2) LAST SUBOBJECT in the object's list, and (3) PREVIOUS SUBOBJECT.

3. New tests are added to the net through DISCRIMINATION LEARNING. The tests are added based upon a "noticing order" with simple tests added before complex tests. EPAM-V's noticing order for complex tests initially uses the algorithm -- proceed down the list until an unsortable subobject is encountered, sort the unsortable subobject using the <NOT ELSEWHERE CLASSIFIED> branch, then go to the end of the list and proceed up the list until all of the subobject's have been sorted. As in previous versions of EPAM, tests will be added only if they differentiate between the patterns that are classified at the node.

4. After, the subobject to be sorted has been identified, the complicated part of the COMPLEX TEST NODE'S sorting routine begins. The classes listed in the TEST NODE (i.e. CLASS I, CLASS J, CLASS K) are examined one at a time until the subobject is successfully sorted in the net as an OBJECT in that class. If the object can not be successfully sorted in any class, the result of the test is <NOT ELSEWHERE CLASSIFIED>. If an object can be successfully sorted in a class it is considered to be a member of that class.

5. CLASS I, CLASS J, and CLASS K appear on both the test node's main list and also on its description list. If the SUBOBJECT being tested is a member of CLASS I then the system will branch to 9-2. If the system finds that it has made a mistake (it can't sort to an pattern, or the translation doesn't fit), the system will return to the pictured test node and continue down the list of classes beginning where it left off. If it then finds that it can sort in CLASS K, before branching to 9-4, the system will reorder the main list at this test node by putting CLASS K in front of CLASS I.

<u>LEAF NODE</u>

An EPAM net is a sort of "tree". Usually the tree is pictured upsidedown with the roots on top followed by the branches and finally with the leaves on the bottom. An OBJECT being recognized in an EPAM net is somewhat like water entering the root of a real tree. The OBJECT starts at the root node which tests for <REALM> and moves along various branches until it arrives at a particular leaf. A test node is a place where branches separate out from each other, a leaf node stands at the end of a branch at the bottom of the net. A LEAF NODE has the form:

```
LEAF
NODE      9-1        0

9-1       0
          <IMAGE>
          9-2
          <PATTERN>
          PATTERN A        0
```

Notes about LEAF NODES:

1. There is no <TEST> stored at this node.

2. The <IMAGE> stored at this node is used by EPAM's DISCRIMINATION LEARNING and can be elaborated by EPAM'S FAMILIARIZATION processes. While PATTERN A can be located at other LEAF NODES, IMAGE 9-2 only is stored at a single leaf node. An IMAGE has the same form as the PATTERN stored at the LEAF NODE except that a complex image can have "NIL" on its list, instead of the name of a CLASS of SUBOBJECTS.

PATTERNS

Just as there are simple objects and complex objects, and simple test nodes and complex test nodes, there are two kinds of patterns, simple and complex. Simple objects get sorted to simple patterns:

```
SIMPLE
PATTERN     9-1        0

9-1        0
           <REALM>
           REALM A
           <CLASS>
           CLASS X
           <TRANSLATION INTO REALM B>
           9-2
           <ATTRIBUTE 4>
           VALUE 4
           ...
           <ATTRIBUTE N>
           VALUE N                0

9-2        0
           OBJECT OF REALM B      0
```

Notes:

1. The <CLASS> of a simple pattern is always known.

2. Simple patterns have a single OBJECT of REALM B on their translation list (List 9-2). This OBJECT can be sorted in the net under REALM B.

```
COMPLEX
 PATTERN     9-1
           CLASS 1
           CLASS 2
           ...
           CLASS N      0

9-1        0
           <TRANSLATION INTO REALM B>
           9-2
           <REALM>
```

```
                REALM A
                <ATTRIBUTE 3>
                VALUE 3
                ...
                ATTRIBUTE N
                VALUE N           0

        9-2     0
                I
                J
                OBJECT OF REALM B
                ...
                K    0
```

Notes:

1. The class names (CLASS 1, CLASS 2, ...) bear a one to one correspondence with the subobjects (SUBOBJECT 1, SUBOBJECT 2, ...) that are on the list of an OBJECT that is sorted to this pattern. CLASS 1 is the <CLASS> of SUBOBJECT 1, CLASS 2 is the <CLASS> of SUBOBJECT 2, and so on. Note that this correspondence is very different from the list of CLASSES (CLASS I, CLASS J, CLASS K,...) that appeared at the COMPLEX TEST NODE. At the COMPLEX TEST NODE the list of classes corresponded to alternatives for a single subobject.

2. There will be redundant pathways created to each complex pattern. If EPAM is imagined as a branching tree, the pattern should not be thought of as a part of the tree since several leaf nodes will point to the same pattern.

3. Complex patterns are what Siklossy simply calls "patterns". They were first programmed by Siklossy (1968) as part of the ZBIE model that translated sentences from a semantic language to various foreign languages.

4. List 9-2 is the "translation rule" used by Siklossy. The numbers (I, J, ..., K) correspond to the order of the subobjects on the OBJECT that is sorted to this complex pattern. OBJECT OF REALM B, is an optional OBJECT that can appear in a translation rule. For example, the letters "ed" might get added every time a noun is used in a verb's position. If I=1 then "I" means the first subobject, if I=2 then I means the second subobject. The order of the translation rule is often different from the order of subobjects in the object and some subobjects may even be left out of the translation. For example an object with three subobjects could have the translation rule (1,2,3) or (1,3,2) or even (2,3).

5. A subobject may in turn be a complex object. The translation routine recurses to enable translation of complex objects that are themselves complex objects.

LEARNING

EPAM-V includes several learning processes:

1. DISCRIMINATION LEARNING. When a new pattern is created by EPAM-V that pattern is partially copied as an IMAGE, and EPAM-V builds new tests and/or new branches on the net in order to discriminate that image from all other images that could be sorted in the net.

2. REDUNDANCY LEARNING. EPAM IV discovered that an EPAM net would need to be extremely redundant in order to explain context effects in letter perception. EPAM-V incorporates this redundancy through a REDUNDANCY LEARNING process that can build several paths through the net to a newly learned complex pattern. In a way REDUNDANCY LEARNING is an example of learning something so well that the system knows it "backwards and forwards".

When a complex pattern is created, several images of that pattern may be discriminated. For example, if the pattern Consonant-Long Vowel-Consonant-Silent E (CVCE) had been recently learned, REDUNDANCY LEARNING would learn the following images: CVCE, -VCE, C-CE, CV-E, and CVC-. Each of these images would be learned in the net through Discrimination Learning, ("-" would be sorted along the <NOT ELSEWHERE CLASSIFIED> branch) with all of the new Leaf Nodes grown pointing to the same pattern. REDUNDANCY LEARNING in some ways may replace the FAMILIARIZATION routine of previous EPAMs as a means of explaining familiarity. One of the parameters of the system is the rate at which the system learns REDUNDANT pathways. A conservative system that wanted to make sure that no previous knowledge was pruned from the tree (see TREE PRUNING) would tend to minimize REDUNDANCY LEARNING. An adventurous system that sought to improve upon poor performance might take the bold leap of making patterns fully redundant as soon as they are created.

3. "SIMPLE PATTERN" CREATION. EPAM-V creates SIMPLE PATTERNS as a result of sorting complex objects through <NOT ELSEWHERE CLASSIFIED> pathways. When the system encounters an unknown subobject at a COMPLEX TEST NODE. The system will then sort at that test node using the <NOT ELSEWHERE CLASSIFIED> branch. If the system successfully sorts the COMPLEX OBJECT, the COMPLEX PATTERN will reveal the CLASS of the unknown subobject (i.e. in the sample COMPLEX PATTERN, pictured on a previous page, the CLASS of the second subobject on the OBJECT'S list would be CLASS 2) and the translation rule will reveal the position in REALM B of the subobject's translation. This is sufficient information to build a SIMPLE PATTERN given the translation of the COMPLEX OBJECT. After the SIMPLE PATTERN is built, it will be learned in the net through DISCRIMINATION LEARNING. Note, the learning of a simple pattern is the equivalent of adding the pattern as a member of a particular <CLASS>.

4. TREE PRUNING. When a pattern being discriminated sorts to the same leaf that points to a different pattern, the pattern at the old leaf gets replaced by the new pattern. In this way, old patterns are replaced by newer ones.

5. REORDERING RULES. When EPAM-V learns a new pattern it tends to apply that pattern whenever it is applicable. However, when EPAM-V finds that the new classification at a particular test node leads to a failure to sort and tries to sort again, and then finds a

classification that works that is below the one that failed on the list, it will move the better classification up on the list to the position just above the classification that failed.

6. PATTERN CREATION. Pattern creation uses means-ends analysis with starting with a similar old pattern as the initial state, and attempting to arrive at a goal translation. (The goal may simply be to find a translation that can be sorted through the net given the other REALM.) The initial state may either be a pattern that sorts through the net but arrives at the wrong translation, a pattern that has one less subobject, or a standard simple pattern that can be applied as a default. The "operators" are the already existing patterns (especially the simple patterns). PATTERN creation is a problem solving component of the EPAM-V system.

7. CLASS CREATION. New classes of subobjects are created as a byproduct of PATTERN CREATION. Once PATTERN CREATION has identified a new translation for a subobject, the system will generate existing classes and test whether any previously created class would result in the identified translation. If not it will create a new <CLASS> and it will learn the pattern of the new subobject as the first member of that class.

Notes:

1. FAMILIARIZATION, a form of learning which played a major part in EPAM-III's explanations of effects of familiarity and meaningfulness in people, may be used in EPAM-V to expand the <IMAGE> stored at a leaf node. However, in the present implementation, the IMAGES are fully elaborated when they are first learned, and so FAMILIARIZATION is not being used. Perhaps, the effects previously explained by FAMILIARIZATION can be explained by REDUNDANCY LEARNING. If not, FAMILIARIZATION can be reimplemented.

2. GENERALIZATION, a form of learning which gave EPAM-IV some "top down" aspects is also not implemented in EPAM-V. The complex patterns and complex test nodes of EPAM-V provide an alternative method of top-down learning. Perhaps some unforseen need will cause GENERALIZATION to be reimplemented.

SUMMARY

EPAM-V is a combination of EPAM III, EPAM IV and ZBIE. It preserves the generality of EPAM III, the redundancy of EPAM IV and makes use of ZBIE's conception of man as a a pattern seeker.

```
9
2 A              32
2 C              32
2 D              32
2 E              32
2 F              32
2 K              32
2 L              32
2 M              32
2 N              32
2 Q              32
2 S              32
2 U              32
2 X              32
2 Y              32
2 Z              32
5          0
  A0     10A0    J10    <CLASS>
  A1     10A1    J10    <TEST>
  A2     10A2    J10    <REALM>
  A3     10A3    J10
  A4     10A4    J10    <IMAGE>
  A5     10A5    J10
  A6     10A6    J10
  A7     10A7    J10
  A8     10A8    J10
  A9     10A9    J10
  A10    10A10   J10
  A11    10A11   J10    1ST BIT
  A12    10A12   J10    2ND BIT
  A13    10A13   J10    3RD BIT
  A14    10A14   J10    4TH BIT
  A15    10A15   J10    5TH BIT
  A16    10A16   J10    6TH BIT
  A17    10A17   J10    7TH BIT
  A18    10A18   J10    8TH BIT
  A19    10A19   J10    9TH BIT
  A20    10A20   J10    10TH BIT
  A21    10A21   J10    11TH BIT
  A22    10A22   J10    12TH BIT
  A23    10A23   J10    13TH BIT
  A24    10A24   J10    14TH BIT
  A25    10A25   J10
  A26    10A26   J10    <PATTERN> takes NIL value (i.e. K10) at a Test node
  A27    10A27   J10
  A28    10A28   J10
  A29    10A29   J10
  A30    10A30   J10
  A31    10A31   J10
;
;                       C ROUTINES ARE USED FOR INPUT AND OUTPUT OPERATIONS
;
  C0       J203         READING ASCII LIST FROM FILE
         10K0           "seeing:"
           J164
```

```
        10N1          INPUTS NEXT ITEM FROM A SEQUENTIAL
          J212        FILE THAT IS OPEN AS FILE 1
        70J8    J4    OUTPUT (0) IS A LIST IN L-FORM OR H5- IF EOF(1)
;
;                     D ROUTINES ARE USED TO SORT AN OBJECT IN THE NET
;
 D0       J44         --SORTING ROUTINE
;                       Sort OBJECT of IMAGE(0) in NET (1)
;                       OUTPUT (0) is TEST NODE or LEAF NODE
;                              (1) is the Location on the object's list
;                                 of the subobject tested at the last
;                                 test node-- (LLST) or OBJECT.
;                       Also adds the PATTERN found at the node to the
;                       OBJECT'S description list.
;                       In the case of complex object's this routine is
;                       used recursively to sort the subobjects.
;
        60W0          W0 = OBJECT
        20W1          W1 = OBJECT (Location of Last Subobject Tested -- LLST)
        60W2          W2 = NET (TEST NODE)
        20W3          W3 = NET
 9-7    11W2
          A26         What pattern? (K10 at test node)
        70J7          Emergency stop if there is no attribute <PATTERN>.
        10K10
          J2
        709-5   9-6   9-5 if LEAF NODE; 9-6 if TEST NODE
 9-5    11W2          SUCCESSFUL CLEANUP
          A26
        20W5          W5=PATTERN
        11W0
        11W5
        10A26
          J11         PATTERN AT LEAF = PATTERN(OBJECT)
 9-8    11W1          UNSUCCESSFUL CLEANUP
        11W2    J34
 9-6    11W3
        11W1
        11W2
        11W0          (0) is OBJECT; (1) is TEST NODE; (2) is LLST; (3) NET
          D3          TEST VALUE EXTRACTOR Output (0) Value of test; (1) LLST
          J6
        20W1          Renew W1= Location Last Subobject Tested--LLST
        11W2
          J6          (0) is Value A; (1) is TEST NODE
          J10
        709-8         CLEANUP if value not classified and no NEC value at NODE.
        20W2    9-7   Put new NODE in W2.
 D2       J46         SUBOBJECT SORTER FOR COMPLEX TEST NODE
;                          INPUTS (0) SUBOBJECT; (1) TEST NODE; (2) NET
;                          OUTPUT (0) VALUE OR K10
;                     Also does many other useful things:
;                     1. If it is the first time trying the classes at a
;                     test node it registers the location of the class that
;                     the subobject first matched as the value of name of test
```

```
;                        node on the subobject's description list.  Then if the
;                        subject tries to sort the object a second time after
;                        an initial failure it can pick up at a particular node
;                        where it left off.
;                        2. This routine learns in the same fashion as a
;                        production list learns when it reorders or reweights its
;                        productions.
;                            If an object was just falsely
;                        sorted at this TEST NODE but later sorting at other
;                        nodes proved that a mistake had been made, and as
;                        just described it resorted and got back to this node,
;                        if this routine then finds a different apparently
;                        successful match further down the list, it will move
;                        the better match in front of the match that had resulted
;                        in failure.
;
          J22            W0=SUBOBJECT; W1=TEST NODE; W2=NET
        11W0
          A5             Find if a location already registered on subobject list.
          J5
        709-3            GOTO 9-3 to pick up where left off last time.
        11W1
   9-3  20W3             W3=TEST NODE (LAST LOCATION SORTED)
   9-1  11W3
          J60            Find next CLASS on SUBOBJECT TEST NODE list
        709-7            9-7 is unsuccessful cleanup
        12H0             CLASS NAME in H0
        20W4             W4=NEXT CLASS NAME from TEST NODE
        30H0
        11W0             (0) is subobject
        11W4
        10A0
          J11            Assign class being tried as CLASS(Subobject)
        11W2
        11W0             (0) subobject; (1) net
          D0             EXECUTE SORTING ROUTINE output is (0) NODE (1) LLST
          J6
        30H0             POP irrelevant LLST
          A26            Get PATTERN (K10 at TEST NODE)
        40H0             (0) PATTERN; (1) PATTERN
        10K10
          J2
        709-2            Goto 9-2 IF PATTERN <> K10
        30H0     9-1
   9-2  20W5             W0=S.O.;W1=T.N.;W2=NET;W3=CLASS LOC.;W4=CLASS;W5=PATTERN
        11W0
        11W1             Was there a false match before?
          J10            i.e.: Is there a value of TESTNODE(Subobject)
        709-4            To 9-4 if this is the first match at this node.
        60W6             W3=Loc. Good Match;W4=Good Match;W6=Loc. False Match
        11W4             (0)Good Match; (1)Location false match
          J63            Insert good match before false match in TEST NODE list.
        11W3
          J68            Delete good match at its old location on TEST NODE list.
        709-5    9-6     Goto 9-5 if old location was last cell on list
```

```
  9-5    11W1
           J70
         70J7     9-6   Emergency stop, list should have more than one cell!
  9-4    11W0
         11W1
         11W3           Assign location in TEST NODE as TESTNODE(Subobject)
           J11    9-6
  9-6    11W4     J36   Successful cleanup
  9-7    10K10    J36   Unsuccessful cleanup
  D3      11W9           TEST VALUE EXTRACTOR
;                       --This routine determines the tested value of
;                         OBJECT (0); at TEST NODE (1); with LLST (2) in NET (3)
;                         Output (0) is VALUE of test; (1) is LLST
           J45
           J21          W0 is OBJECT; W1 is TEST NODE; W2 is NET
         20W4           W4 is LLST (Location of Last Subobject Tested)
         20W2           W2 is NET
         11W1
           A1           What test is stored at this node?
         70J7           Emergency stop if no test is stored at this TEST NODE.
         20W3           W3 is Test tested at node.
         11W0
           J60          Is this a complex TEST NODE?
         709-3          GOTO 9-3 if it is a simple TEST NODE
         30H0           POP LOCATION
         11W4           Find SUBOBJECT being TESTED
         11W0
         11W3
           J1           Executes test with (0)OBJECT (1)LLST
;                       Output of a complex test is (0)LLST -- The location
;                       of the Subobject on the OBJECT'S LIST.
         60W4           W2=NEW LLST
         12H0           (0) SUBOBJECT (1) LOC STIM OBJECT
         20W5           W5=SUBOBJECT
         30H0
         11W0
         11W5
           J77          TEST if SUBOBJECT is on TEST NODE'S main list.
         709-4          GOTO 9-4 if SUBOBJECT IS NOT A CLASS NAME
         11W1           This part of routine sorts an IMAGE at a COMPLEX NODE
         11W5
           J10    9-2   Find SUBOBJECT(TEST NODE)
  9-4    11W2
         11W1
         12W4           (0) Subobject; (1) TEST NODE; (2) NET
           D2     9-2   Output of D2: (0) VALUE OR K10
;                       Also finds PATTERN of subobject (0) and adds it as value
;                       of attribute <PATTERN> A26 of the subobject.
  9-3    11W0           This is a simple node.
         11W3
           J1           Execute TEST (0) on OBJECT (1)
;                       Output of TEST should be (0) VALUE or H5-
;                       NOTE: A test for a value of an attribute on the
;                       description list of an object is simply the usual
;                        10Ai   J10
```

```
         709-1
   9-2   11W0      J35     Exit with (0) VALUE or K10; (1)LLST
   9-1   10K10     9-2
   E1      J51             CHECK NEXT SUBOBJECT
;                          INPUTS (0) OBJECT; (1) LLST
;                          OUTPUTS (0) LLST -- Location of the subobject on object.
;                          W0=OBJECT; W1=LLST
          11W1              (0) LLST
           J60
         70J7      J31     EMERGENCY STOP IF NO NEXT OBJECT
   E2      J51             CHECK LAST SUBOBJECT ON LIST
;                          INPUTS (0) OBJECT; (1) LLST
;                          OUTPUTS (0) NEW LLST
;                          W0=OBJECT; W1=LLST
         11W0              (0) OBJECT
           J61
         70J7      J31     EMERGENCY STOP IF NO OBJECT ON LIST
   E3      J51             CHECK PREVIOUS SUBOBJECT
;                          INPUTS (0) OBJECT; (1) LLST
;                          OUTPUTS (0) NEW LLST
         11W0
         11W1              (0) LLST; (1) OBJECT
           E4              FIND LOCATION OF SYMBOL BEFORE LOC (0) ON LIST (1)
         70J7      J31
   E4      J52             LOCATE PREVIOUS SYMBOL BEFORE CELL (0) ON LIST (1)
;                          OUTPUT IS (0) LOCATION or H5- if no previous symbol.
         10K10
         20W2
   9-1   11W1              W0=CELL; W1=PRESENT LOC.; W2=LAST LOCATION
           J60             (0) is LOCATION of first SYMBOL on LIST
         70J31             Exit with H5-
         60W1              W1 is location moving forward from beginning of list.
         11W0
           J2              TEST if PRESENT LOC = CELL
           J5
         709-2             Goto 9-2 if present location = CELL
         11W1
         20W2      9-1
   9-2   11W2
         10K10             If CELL holds the first SYMBOL on LIST will exit H5-
           J2              otherwise exit with (0)= Last Location and H5+
           J5
         70J32
         11W2      J32
   E6      J41             TEST: NUMBER OF SUBOBJECTS
;                          INPUTS (0) OBJECT
;                          OUTPUTS (0) I-TERM (# OF SUBOBJECTS)
;                          also exit with H5+
         20W0
           J90
           J124
         20W1              PUT COUNTER=0 INTO W1
   9-1   11W0              W0=OBJECT
           J60
         709-2             GOTO 9-2 if have reached the end of the list.
```

```
        20W0           W0= new location on object list
        11W1
          J125         COUNTER=COUNTER + 1
        20W1    9-1
  9-2   11W1
        10I255
          J116         Test if more than 255 subobjects in OBJECT.
          J5
        70J7           EMERGENCY STOP if more than 255 subobjects in OBJECT!
        11W1
          J211         New IPL-V routine converts DATA TERM 0<DATA TERM<255
;                      into an I-term, a regional predefined data term.
;                      I0=0, I1=1, I2=2,..., I255=255.
        11W1
          J71          Erase data term's list.
        20W1           I-term to W4
        11W1
          J4    J34
  E7      J4    0      TEST WHAT IS THE NAME OF THE OBJECT
                                      expand for case when must recurse
                                      from word to letter REALM
                                      to sort features
;
;                      F routines are for DISCRIMINATION LEARNING
;
  F0      J203         TEST FINDER
;                      This routine determines test to add at LEAF NODE
;                      Inputs: (0) LEAF NODE
;                              (1) NEW IMAGE to be learned
;                              (2) NET
;                      Outputs: (0) TEST
;                               (1) Value of test for NEW IMAGE
;                               (2) Value of the test for OLD IMAGE
;                                        or
;                               EXITS with H5- if can't discriminate the new
;                               image from the old image that was stored at (0).
;
;                      Tests are added according to these rules:
;                      1. CLASS
;                      2. NUMBER OF SUBOBJECTS
;                      3. IF NUMBER OF SUBOBJECTS = 0 then OBJECT NAME
;                      4.    ELSE NEXT SUBOBJECT
;                      5. IF RESULT OF NEXT SUBOBJECT = NIL THEN 6 ELSE 4
;                      6. LAST SUBOBJECT
;                      7. PREVIOUS SUBOBJECT
;                      8. GOTO 7
        10K1
          J164
          J49
          J22          W0=LEAF NODE; W1=NEW IMAGE; W2=NET
        10W0
          A4
        70J7           Emergency STOP; Leaf Node does not point to IMAGE.
        20W3           W3=OLD IMAGE
        11W3
        10A0           BEGIN TEST for CLASS
        60W6           W6= TEST
          A0           <CLASS>
```

```
          9-1
         20W4            W4= TEST(OLD IMAGE)
         11W1
          A0             <CLASS>
          9-1
         20W5            W5= TEST(NEW IMAGE)
          9-3            Check if TEST(OLD IMAGE)=TEST(NEW IMAGE)
         709-2           Images don't equal; Goto 9-2 for successful cleanup.
         10E6
         20W6
         11W3            BEGIN TEST for NUMBER OF SUBOBJECTS
          E6
         20W4            W4=NUMBER(OLD IMAGE)
         11W1
          E6
         20W5            W5=NUMBER(NEW IMAGE)
          9-3            TEST FOR EQUALITY
         709-2           9-2 IF don't equal, successful cleanup.
         11W4
          J117           Are they simple images
         709-4           To 9-4 if complex images
         11W1
         20W5
         11W3
         20W4
         10E7
         20W6     9-2    Simple images, sort by their names.
9-4      11W1            COMPLEX IMAGE
         20W8            W8 will equal NEW LLST
         11W3
         20W7            W7 will equal OLD LLST
         11W4
         20W2            W2 now equals number of subobjects on image list
          J90
          J124
         20W9            W9=COUNTER=0
9-6      10E1            9-6 is GET NEXT
          9-12
         709-11          Successful cleanup
          9-7            Counter routine
         709-8    0      If end of list then exit with H5- (unsuccessful cleanup)
          9-5            TEST IF THE VALUES WERE K10
         709-9    9-6    9-9 if last node NEC
9-9      10E2            9-9 is GET LAST
          9-12
         709-11          Successful cleanup
          9-7            Counter routine
         709-8    9-10   If end of list then exit with H5- (unsuccessful cleanup)
9-10     10E3            9-10 is GET PREVIOUS
          9-12
         709-11          Successful cleanup
          9-7            Counter routine
         709-8    9-10   If end of list then exit with H5- (unsuccessful cleanup)
9-1       J5             Replaces H5- with K10
         70J4
```

```
        10K10   0
 9-3    11W4
        11W5
          J2    0      H5+ if TEST(OLD IMAGE)=TEST(NEW IMAGE)
 9-7    11W9
          J125         Counter=Counter + 1
        11W3           Input # of subobjects.
          J114  J5     Exit with H5- at end of list, H5- still more sobobjects
 9-5    11W4           Testing if the values were K10
        10K10            to see if the image had been sorted via the NEC node.
          J2    J5     Exit with H5- if values were K10 (i.e. NIL)
 9-12   20W6           INPUT (0) TEST / OUTPUT H5- if found test that discrims.
        11W7           W0=LN;W1=NI;W2=#;W3=OI;W4=TO;W5=TN;W6=T;W7=LLO;W8=LLN;9=C
        11W3
        11W6
          J1
        70J7           EMERGENCY STOP -- No LOCATION on old image!
        12H0
        20W4
        20W7
        11W8
        11W1
          J1
        70J7           EMERGENCY STOP -- No LOCATION on new image!
        12H0
        20W5
        20W8
          9-3   0
 9-8      J39   J3     UNSUCCESSFUL CLEANUP
 9-11   11W9           SUCCESSFUL CLEANUP FOR COMPLEX IMAGES
          J71          ERASE COUNTER
          J4
        10I1
        11W4
        11W5
        11W6    J39
 9-2      J4           SUCCESSFUL CLEANUP FOR SIMPLE TESTS
        10I0
        11W4
        11W5
        11W6    J39
 F1       J203         TEST NODE CREATOR
;                      This routine creates a TEST and Test Node that
;                      discriminates a NEW IMAGE from the OLD IMAGE
;                      that is present at the leaf.
;                      INPUTS:
;                             (0) LEAF NODE
;                             (1) NEW IMAGE
;                             (2) NET
;                      OUTPUT:
;                             (0) IS CREATED EMPTY LEAF NODE (9-4) or H5-
;                      EXAMPLE:
;
;                                          INPUT
;
```

```
;                              9-0
;                           LEAF NODE  9-1           0
;
;                              9-1     0
;                                      A26 <PATTERN>
;                                      PATTERN A
;                                      A4 <IMAGE>
;                                      OLD IMAGE              0
;
;                                              OUTPUT
;                              9-0
;                            CREATED
;                           TEST NODE  9-1
;                                      TEST(NEW IMAGE)
;                                      TEST(OLD IMAGE)        0
;
;                             9-1     0
;                                      A1 <TEST>
;                                      TEST
;                                      A26 <IMAGE>
;                                      K10 (NIL)
;                                      TEST(OLD IMAGE)
;                                      9-2
;                                      TEST(NEW IMAGE)
;                                      9-4           0
;
;                              9-2
;                           LEAF NODE  9-3           0
;
;                               9-3    0
;                                      A26 <PATTERN>
;                                      PATTERN A
;                                      A4 <IMAGE>
;                                      OLD IMAGE       0
;
;                             9-4
;                           CREATED
;                            EMPTY
;                          LEAF NODE   0               0
;
;
       10K1
         J164
         J49
         J22                 W0=9-0 of illustration; W1=NEW IMAGE; W2=NET
       11W2
       11W1
       11W0
         F0                  TEST MAKER -- OUTPUT (0)TEST; (1) TEST(NI); (2) TEST(OI)
;                                                (3) I0=SIMPLE TEST; I1=COMPLEX TEST
;                              OR H5- if can't make a test
         709-2               9-2 is unsuccessful cleanup
         20W7
         20W8
         20W9
```

```
        20W5          W5=I0 is simple test node; I1 is complex test node
          J90
        20W3          W3=Created leaf node (9-3 in illustration above)
        11W0
          A26
        70J7          EMERGENCY STOP no pattern at LEAF NODE!
        20W6          W6 is temporary holder of PATTERN
        11W3
        11W6
        10A26
          J11         Transferring from leaf node turned test node to leaf node
        11W0
          A4
        70J7          EMERGENCY STOP no image at LEAF NODE!
        20W6          W6 is temporary holder of IMAGE
        11W0
        10A4
          J14         Erase attribute IMAGE on 9-0's description list.
        11W3
        11W6
        10A4
          J11
        11W0
        11W7
        10A1
          J11         1W7=TEST(TEST NODE)
        11W0
        10K10
        10A26
          J11         NIL=IMAGE(TEST NODE)
          J90
        20W4          W4=EMPTY LEAF NODE (FOR OUTPUT (0))
        11W0
        11W4
        11W8
          J11         CREATED LEAF NODE=1W8(TEST NODE)
        11W3
        11W2
        11W9
          J11
        11W5
        10I1
          J2          Is this a complex test node?
        709-1         GOTO 9-1 if it is a simple test node.
        11W0
        11W8
          J65         PUT 1W8 ON TEST NODE'S MAIN LIST
        11W0
        11W9
          J65         PUT 1W9 AFTER 1W8 ON TEST NODE'S MAIN LIST
9-1     11W4
          J39    J4
9-2       J203
        10K5          "can't derive test"
          J164
```

```
          J39   J3    Unsuccessful cleanup.
F3        J203        MAKE LEAF NODE
;                     INPUTS (0) Newly Created Empty Leaf node
;                            (1) NEW IMAGE
;                            (2) NEW PATTERN
        10K3
          J164
          J52
        11W0
        11W1
        10A4
          J11         NEW IMAGE= A4(NODE)
        11W0
        11W2
        10A26
          J11   J32   NEW PATTERN= A26(IMAGE)
F5        J44         DISCRIMINATE
          J22         INPUTS (0)IMAGE, (1)NET, (2)PATTERN
        11W1          OUTPUT: Adds PATTERN to net either by Creating Leaf
        11W0                  or by TREE TRIMMING
          D0          SORT Image (0) in Net (1) Output is (0)NODE; (1)LLST
          J6
        20W4          W4= LLST of Image at last TEST NODE
        60W3          W3=NODE
          A26         Is this a TEST NODE or a LEAF NODE
        70J7          EMERGENCY STOP -- No value for pattern at node!
        10K10
          J2
          J5
        709-2         TO 9-2 IF TEST NODE (PATTERN is K10)
        11W3          LAST NODE IS LEAF NODE
          A26
        11W2
          J2          Test if Pattern at LEAF NODE already same as PATTERN
          J5
        709-7         Goto Cleanup -- Nothing to Learn!
        11W1
        11W0
        11W3
          F1          MAKE A NEW TEST NODE-- Output (0) is empty LN or H5-.
        709-8         GOTO 9-8 for Tree Trimming (can't derive test)
9-9     20W3          W3=NEWLY CREATED LEAF NODE
        11W2
        11W0
        11W3
          F3    9-7   MAKE A NEW LEAF NODE and EXIT
9-2     11W4
        11W0
        11W3
          F6    9-9
9-8     11W3          TREE TRIMMING
        11W2
        10A26
          J11   9-7   Replaces Old Pattern with New Pattern at Leaf Node
9-7       J34   0
```

```
 F6       J46            MAKE BRANCH
;                        INPUT (0) TESTING NODE; (1) NEW IMAGE; (2)LLST at T.N.
;                        OUTPUT (0) EMPTY LEAF NODE
          J22
          J90            Create what will be new LEAF
        20W3
        11W0
          J60            Is this a SIMPLE or a COMPLEX node?
        709-3            GOTO 9-3 if it is a Simple node.
        20W6             LOCATION of first Class on Complex Test Node list.
        11W2             LLST is location of object on list tested at this node.
        12H0
        20W5             W5=SUBOBJECT on IMAGE
        30H0
        11W6
        11W5
          J63            INSERT SUBOBJECT at top of list at complex test node.
        11W0
        11W3
        11W5
          J11            EMPTY LEAF NODE=SUBOBJECT(TEST NODE)
 9-4    11W3     J36     CLEANUP
 9-3    11W0
          A1
        70J7             EMERGENCY STOP -- No Test stored at TEST NODE
        20W4             W4=TEST
        11W1
        11W4
          J10
        20W5             W5=TEST(NEW IMAGE)
        11W0
        11W3
        11W5
          J11    9-4
;                        L Routines Manipulate Patterns
 L0       J47            TRANSLATOR
;                        INPUTS: (0) OBJECT; (1) REALM TO TRANSLATE TO
;                                1M2 IS CELL TO PUT TRANSLATION AT END OF.
                         EXIT WITH H5- IF THE OBJECT IS K10
        20W0             W0=OBJECT with PATTERN attached
        20W2             W2=TRANSLATION REALM
        11W0
          A26
        70J3             EXIT H5- IF OBJECT IS K10
        60W1
        10W2
          J10            GET TRANSLATION LIST
        70J7             EMERGENCY STOP-- No translation list
        20W4             W4=LAST LOCATION ON TRANSLATION LIST
        11M2             1M2 IS LIST TO TAG TRANSLATION ON.
        11W2
        10A2
          J11            REALM= TYPE(CELL)
 9-2    11W4
          J60
```

```
        709-3           EXIT AT END OF TRANSLATION RULE (or blank rule)
        12H0
        20W5            W5=I-VALUE OR OBJECT OF REALM W2 ON THE TRANS LIST
        20W4            REFRESH W4=LAST LOC. ON TRANSLATION LIST
        11W5
        11W2
          J10           TEST IF ITEM HAS REALM W2 ON ITS DESCRIPTION LIST
          J5
        709-4           GO TO 9-4 IF OBJECT IS REALM W2
          J90
          J124
        20W6            W6= COUNTER SET TO 0
        11W0
        20W7            W7= LOCATION IN OBJECT'S LIST
9-5     11W7
          J60
        70J7            EMERGENCY STOP. Can't get to W5th subobject.
        20W7
        11W6
          J125          COUNTER=COUNTER+1
        11W5
          J114          TEST IF COUNTER=1W4
        709-5           NEXT CELL IF COUNTER = 1W5
        11W6
          J71           ERASE COUNTER
        12W7
        20W7            W7= W5TH SUBOBJECT
        11W2
        11W7
          L0            RECURSE
          9-7   9-2     GOTO 9-7 IF OBJECT IS K10
9-4     11W5
        11M2
          J65   9-2     PUTS OBJECT AT END OF 1M2
9-3       J37   J4      CLEANUP
K0        0             "***starting***"
          Q10
          Q10
          Q10
          S19
          S20
          S1
          S18
          S20
          S9
          S14
          S7
          Q10
          Q10
          Q10   0
K1        0             "making test"
          S13
          S1
          S11
          S9
```