AUTOMATIC PROGRAMMING GROUP

Internal Memo 12
August 1, 1973

MAPL 2

by

William A. Martin

A new version of MAPL(1) is proposed. It is similar to the current MAPL, but it is focused on structural descriptions composed of relations, rather than individual relations.

Data Types

MAPL 2 has only LISP S-expressions as a data type, but the non-atomic S-expressions have properties (as in LISP they are not unique), and there is a restriction given below on what S-expressions are permitted.

Concepts

The basic entities in MAPL 2 are the concept and the fact. Non-numeric, non-NIL S-expressions serve as the names of concepts. The remaining S-expressions define concepts. A fact is a concept which has been presented to the system as rejected, known, believed, doubted, assumed, or supposed.

Concepts form a lattice under KIND. The top level node is CONCEPT. For example, a KIND of a PERSON is a FARMER, another KIND of a PERSON is an EMPLOYEE. It is possible for TOM to be both a FARMER and an EMPLOYEE. Concepts also form a lattice under PART. The top level node is UNIVERSE. For example, a PART of a TRAIN is the ENGINE, another PART of a TRAIN is the CAR. A WHEEL is both PART of an ENGINE and part of a CAR. NIL is the name of the concept at the opposite end of the lattice from CONCEPT, and from UNIVERSE.

Properties of Names of Concepts

Every name has the following properties.

1) A-K-O is the list of all the names which have this name as a KIND in one step.
2) A-P-O is the list of all the names which have this name as a PART in one step.
3) KIND is the list of all the names which this name has as a kind in one step.
4) PART is a list of all the names which this name has as a part in one step.
5) serves-as-argument is a list of all the non-atomic S-expressions where this name appears as an argument at the top level.
6) definition is an S-expression which defines the concept for which this is the name. (This property is optional.)
7) applications is the list of non-atomic S-expressions for which this name appears in functional position at the top level.

Properties of Numbers and NIL

Numbers are all implicitly a KIND of INTEGER or FLOATING-POINT-NUMBER. NIL has no properties.

Properties of Non-Atomic S-Expressions

We define a maximal list to be one which is not the CDR of any list existing in the system. Non-atomic S-expressions have one property on their property list.

1) serves-as-argument is a list of the maximal lists for which this S-expression is an element at the top level.

Non-atomic S-expressions must obey the following restrictions.

1) Only lists are allowed.
2) The element in functional position must be a concept name.
3) If we have the S-expression (A arg-A_1,... arg-A_n) in the system and B is a kind of A, then (B arg-B_1 ... arg-B_n) can be added to the system only if arg-B_i is a kind of arg-A_i for $1 < i < n$.
4) (A arg-A_i ... arg-A_n) is a kind of A unless the function has a KIND-EXPR which computes its kind from the kinds of its arguments. KIND, AND, XOR, and the like require KIND-EXPR's.

World Modeling with MAPL 2

We will explore here the possibility of building world models primarily with S-expressions containing one, two, or three elements. First, for purposes of notation and input, we will represent the KIND and PART relations as

(A-K-O NAME1 NAME2)
(A-P-O NAME1 NAME2)

where the first argument is contained in the second.

In MAPL2 we will want to be able to say both Balls are round, and A blue ball is in the corner. Referring, respectively, to all balls and one ball. Similarly, we want to say Three men sold a big house. and Bob sold a small house. without mixing up the instances of selling. Since list structure is non-unique we achieve this by making all non-functional references which do not apply to all instances of a concept, of the form (KIND CONCEPT). All functional references are automatically in list structure, for example, (SELL HOUSE). We then modify unique list structure copies of these expressions. For this purpose we have the MAPL function, LEARN, which maintains unique copies of list structure during one call. As defined in the first section, the MAPL2 data structure is quite general. However,

we will adopt some conventions in order to speed searching of the data base. The basic idea is that the objects and verb meanings are the most important. These should appear at the top level with properties hanging off of them. If the searcher can find the expressions involving the right objects and verb meanings, it has greatly narrowed the search and can then pattern match on the properties of the remaining candidates. Thus, we could represent Three men sold a big house as ((THREE (KIND MAN))(SELL) (A (BIG (KIND HOUSE))), but we won't. Instead we will form the key expressions of the event and the objects, and then modify and relate these. In doing this it is important to keep in mind the distinction between an expression representing the application of a function and an expression representing its value. (SELL HOUSE) stands for the application of selling to a house. The properties are who did it, when, and also the VALUE. With characteristics of events and objects such as AGENT and SIZE we will have the convention that the third argument, if present, is an expression for the value. For example, (SIZE (KIND HOUSE) BIG). The first tuple to learn must be the top node of the structure being learned.

We are now ready for Three men sold a big house.

```
(LEARN (SELL HOUSE) (COUNT (KIND MAN) 3)
       (SIZE (KIND HOUSE) BIG)
       (COUNT (KIND HOUSE) 1)
       (AGENT (SELL HOUSE) (KIND MAN))
       (INPUT (SELL HOUSE) (KIND HOUSE)))
```

The tuples in the call to LEARN above all describe a single event and the objects associated with it. In general, a call to learn can describe a an

1) deduction
2) event
3) object

We have just described objects and an event. Deductions are of the form

a) (IMPLIES event1 event2)
b) (DID-NOT-BLOCK event1 event2)
c) (SET-GOAL-FOR event1 event2)

The user should think of the structure set up by a single call to LEARN as a pattern to be used in thinking. Consider the sentences found in Moore [2]

John wants to marry the prettiest girl.

The President has been married since 1945.

The President has lived in the White House since 1800.

As Moore says, these are ambiguous because we do not know the appropriate time for evaluation of substructures. If entered into MAPL as they are, and used for thinking, they could match whoever happens to be the President or the prettiest girl at the time they are used, or they could be taken in the generic when they are used. The MAPL 2 philosophy is that this is the decision of the pattern matcher, if unambiguous expressions are wanted they can be created as in English with more elaborate or more specific patterns.

The current President has been married since 1945.

President Nixon has been married since 1945.

Consider the sentence,

Two of the three eggs are rotten.

```
(LEARN  (KIND EGG)
        (DEMONSTRATIVE (KIND EGG) THE)
        (COUNT  (KIND EGG) 3)
        (COUNT  (KIND (KIND EGG)) 2)
        (FRESHNESS (KIND (KIND EGG)) ROTTEN) )
```

Asked if there are any non-rotten eggs the system will look for the uses of EGG and find the (KIND EGG) S-expression above. It will determine that (KIND EGG) is used in (COUNT (KIND EGG) 3) (DEMONSTRATIVE (KIND EGG) THE), and (KIND (KIND EGG)), but no FRESHNESS expressions. However, judgement can be put in the matcher to say that if a specific subset are known rotten then it is likely that the remaining one is not. Again, the matcher must use judgement.

Next, consider

No two A&T stores sell the same kind of items.

```
(LEARN  (SELL ITEM) (AGENT (SELL ITEM) (KIND STORE))
       (NOT (SELL ITEM))
         (COUNT (KIND STORE) 2)
        (A-P-O (KIND STORE) A&T)
        (OTHER (KIND ITEM) SAME)
        (VALUE (SELL ITEM) (KIND ITEM)) )
```

The matcher will recognize the OTHER property as requiring interpretation and do it by looking for a plural AGENT.

Next let us consider the sentences

Bill's phone number is 262-1373.

John knows Bill's phone number.

The first sentence leads most naturally to the fact.

```
(LEARN (PHONE-NUMBER BILL 262-1373))
```

But the second sentence using the same PHONE-NUMBER characteristic would be

```
(LEARN (KNOW VALUE) (AGENT (KNOW VALUE) JOHN)
       (VALUE (KNOW VALUE) (PHONE-NUMBER BILL 262-1373)))
```

which is clearly incorrect since the sentence doesn't say what the phone number is. This is why we need the optional 3rd argument on characteristics. We can say

```
(LEARN (KNOW VALUE) (AGENT (KNOW VALUE) JOHN)
       (VALUE (KNOW VALUE) (VALUE (PHONE-NUMBER BILL))))
```

John knows how to find Bill's phone number.

would be

```
(LEARN (KNOW METHOD) (AGENT (KNOW METHOD) JOHN)
       (VALUE (KNOW METHOD) (METHOD (FIND PHONE-NUMBER)))
       (VALUE (FIND PHONE-NUMBER) (PHONE-NUMBER BILL)))
```

Next, consider

Every boy either loves Sata or hates him.

To handle this we must use the function XOR which takes any number of arguments which are kinds of the same concept. (There is a similar function AND.)

```
(LEARN (XOR (LOVE SANTA)(HATE SANTA))(AGENT (XOR (LOVE SANTA)
    (HATE SANTA)) BOY) (VALUE (XOR (LOVE SANTA)(HATE SANTA)) SANTA))
```

Next, consider the expression of values and negatives.

Purple is a color.

Apples are not purple.

John does not sell apples.

```
(LEARN (VALUE (COLOR PHYSICAL-OBJECT) COLOR-VALUE))
(LEARN NIL (A-K-O PURPLE COLOR-VALUE))
(LEARN APPLE (NOT (COLOR APPLE PURPLE)))
(LEARN (SELL APPLE) (AGENT (SELL APPLE) JOHN) (NOT (SELL APPLE))
   (VALUE (SELL APPLE) APPLE))
```

Another problem on input is the occurance of multiple distinct instances of the same concept in a single fact, as in

For every number there is some larger number.

To handle this we distinguish the instances with (TOKEN S-expr n), LEARN removes the TOKEN, keeping only those S-expr's with the same value of n unique.

```
(LEARN (IMPLIES (TOKEN (KIND CARDINAL-NUMBER) 1)
                (TOKEN (KIND CARDINAL-NUMBER) 2))
       (GREATER-THAN (TOKEN (KIND CARDINAL-NUMBER) 2)
                     (TOKEN (KIND CARDINAL-NUMBER) 1)))
```

We must handle measures.

An adult rabbit weighs more than 2 pounds.

```
(LEARN (GREATER-THAN
   (VALUE (POUND-MEASURE (WEIGH (KIND RABBIT)))) 2)
      (MATURITY (KIND RABBIT) ADULT))
```

Naming

Let us turn now to an example from Winston [3]. First we introduce the concepts with which he describes an arch

```
(LEARN NIL (A-K-O STAND ORIENT)
           (A-K-O LIE ORIENT)
           (A-K-O LEFT LOCATION)
           (A-K-O ABUT LOCATION)
           (A-K-O BRICK PHYSICAL-OBJECT)
           (A-K-O WEDGE PHYSICAL-OBJECT) )
```

An arch can be defined as

```
(LEARN (KIND STRUCTURE)(A-N-O ARCH (KIND STRUCTURE) )
     (A-P-O (TOKEN (KIND BRICK) 1) (KIND STRUCTURE))
     (A-P-O (TOKEN (KIND BRICK) 2) (KIND STRUCTURE))
     (A-P-O (KIND PHYSICAL-OBJECT)(KIND STRUCTURE))
   (MUST (TOKEN (SUPPORT PHYSICAL-OBJECT) 1))
     (AGENT (TOKEN (SUPPORT PHYSICAL-OBJECT) 1) (TOKEN (KIND BLOCK) 1))
   (MUST (TOKEN (SUPPORT PHYSICAL-OBJECT) 2))
     (AGENT (TOKEN (SUPPORT PHYSICAL-OBJECT) 2)(TOKEN (KIND BLOCK) 2))
   (MUST (NOT (ABUT)))
     (AGENT (ABUT)
          (AND (TOKEN (KIND BRICK) 1)
               (TOKEN (KIND BRICK) 2)))
          (LIE (KIND PHYSICAL-OBJECT))
          (STAND (TOKEN (KIND BRICK) 1))
          (STAND (TOKEN (KIND BRICK) 2))
          (LEFT (TOKEN (KIND BRICK) 1) (TOKEN (KIND BRICK) 2))
          (A-N-O ARCH-TOP (KIND PHYSICAL-OBJECT)) )
```

The first A-N-O makes this fact the definition of ARCH and makes ARCH A-K-O STRUCTURE. The second A-N-O makes ARCH-TOP A-K-O PHYSICAL-OBJECT and defines it to be an object which participates in an ARCH in the way given by the definition.

Asking Questions

A question is asked by giving an S-expression with a ? for one of the elements. The other elements can be (HAS-KIND element), (KIND-OF element), (HAS-PART element), (PART-OF element), or just an S-expression. In evaluating these expressions it is critical to note that because of the restrictions we made on S-expressions, one need only move up and down from the function name to find what the S-expression has as kinds or is a kind of. When one checks the KIND relation he can either go down the KIND lattice or refer to the definition of a concept and try to match an object against the definition.

REFERENCES

1. W. A. Martin, R. B. Krumland, and A. Sunguroff, "More MAPL: Specifications and Basic Structures", MIT Project MAC APG Memo 8.

2. R. C. Moore, "D-SCRIPT: A Computational Theory of Descriptions", MIT A/I Lab Memo 278 (February 1973).

3. P. H. Winston, "Learning Structural Descriptions from Examples", MIT A/I TR-231 (September 1970).