(18) AFOSR-TR-80-1025

LEVEL

A First Iteration at a Mini-set of Features for a Knowledge Representation Language*

by

Linda Salsburg

(11) Aug 80

(12) 19

August, 1980

Technical Report #80-1

*Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under contract ~~no.~~ F49620-79-C-0131. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

Abstract

In the last three years, several initial versions of a new class of languages, called knowledge representation languages, have emerged ~~(Bobrow and Winograd,~~ 1977; Brachman, 1978; and Roberts and Goldstein, 1977). Rather than presenting new solutions to issues in knowledge representation, this paper selects a mini-set of features that seem necessary to attain the goals of knowledge representation languages. The mini-set selected is definitely <u>not</u> complete, but merely represents the first iteration in trying to define such a mini-set. An example, specifying the meaning of the square root function, is included. Though the example is presented, we are not committed to the syntax presented there; rather, we are committed to finding a more reasonable syntax and user interface for the language. The language has not been implemented, since we plan a second iteration prior to that.

I. Basic Features

In semantic nets (a formalism often used to represent knowledge), the routines for processing the nets often completely determine what the representation means. Two nodes may be tied together by a link, but the interpretation of each of the elements, both nodes and links, can only be found implicitly within the processing routines. The representation itself provides no explicit knowledge of the import of the links. What often results is a knowledge net which is ambiguous and not readily extended plus a set of routines that are not domain independent. One is often forced to begin almost from scratch when trying to work in a new area of knowledge.

By requiring that the representation language carry a fixed interpretation of each interrelationship and of the participants, one is faced with two problems. First, the precise meaning of the above must be understood. Second, one must attempt to capture the kinds of things that would be required to represent any domain.

The following features are necessary in a knowledge representation language. Our features are drawn from three existing languages: KRL (Bobrow and Winograd, 1977), KL-ONE (Brachman, 1978), and FRL (Roberts and Goldstein, 1977).

i) The language must partition knowledge.

Not only must the representation language be able to unambiguously represent the knowledge needed, but it must facilitate subsequent processing of that knowledge. This includes making changes to the knowledge base as well as retrieving information from and deriving conclusions based on the knowledge that is stored.

When trying to reason using the knowledge base, it is important to constrain the number of possible inferences that can be made in an attempt to avoid combinatorial explosion. So, if one is trying to draw a particular conclusion about Martians, only the information relevant to the topic should be used to make inferences. For, in trying to discover whether or not Martians are extraterrestrial beings, it probably would not help to know that Rover the dog lives next door. Therefore, it is necessary to chunk or partition knowledge in order to use it effectively in subsequent processing.

A necessary feature of a knowledge representation language is that it facilitates this chunking or clustering of knowledge. In trying to represent some concept, one must be able to use that concept as a central repository for what must necessarily be associated with it. This should not place any restrictions on the way things are partitioned, but make it easier to bring relevant information into focus.

ii) The language must distinguish between extensional and intensional descriptions.

A representation language must be able to distinguish between describing an individual and what it means to be an individual. For example, one may want to represent Rover as the dog that lives next door, who is brown in color, who bites when provoked, etc. Yet, one may also want to represent what it means to be a Martian as a creature from Mars, who is possibly green, etc. Notice that with the Martian, there is no commitment to the existence of that which is being described. The former is what is called an extensional description, while the latter is an intensional one. It should be clear that sometimes, one would want to represent extensional descriptions, while at other times intensional descriptions would be necessary. Therefore, a knowledge representation language must be able to capture this distinction.

iii) The knowledge language should allow for class membership and inheritance of properties.

An intensional description can be thought of as capturing a generality or describing a class of objects. For instance, one can describe the attributes of any dog, not a specific one, and then know that these properties will be true of any extensional object which is a dog. The intensional description captures features that any object in that class being described will have. The extensional object inherits these attributes.

A knowledge representation language should be able to exhibit the relationship between intensional and extensional descriptions in terms of class membership and inheritance of properties. First, this helps to keep the knowledge base consistent. One can say that dogs are in the class of mammals. Then, each specific occurrence of a dog inherits the properties of mammals by virtue of being in the class of dogs which are in the class of mammals. The attributes are attached higher up in the hierarchy created by class membership, and it is not necessary to remember to attach these properties to each individual dog. As a by-product of attaching properties higher up, the descriptions at the lower levels (which are more numerous) are shortened. As previously stated, it is important to factor knowledge. This hierarchy of class membership helps to do precisely that.

iv) A knowledge language must allow an object to be in more than one class.

Generally, it is agreed that capturing class membership and inheritance from more general classes is important in any knowledge representation language. It seems that a knowledge representation language should have multiple class membership and inheritance paths for the following reasons. First, since what is being represented may serve multiple functions whose attributes are not even vaguely similar, it makes sense to be able to view things from different viewpoints. For example, a rock has certain properties of interest when viewed by a geologist (type, weight, etc.), and still others when being purchased in a store as a "pet rock" (price, packaging, etc.). Second, multiple class membership paths help in subsequent processing by further factoring the knowledge base. Lastly, a linear scheme can result in extensive changes to the knowledge base should a shift in emphasis occur.

v) The knowledge language should allow inherited properties to be modified.

It is not always desirable to inherit properties directly. For instance, Rover the dog may only have three legs, but the idea that dogs have four legs is probably stored higher in the hierarchy than with the individual. There is nothing preventing the property of number of legs from being stored with the individual, it just makes more sense that it be stored higher up. One would still like to say that Rover is in the class of dogs and modify the attribute of having four legs. This is just one type of modification to inheritance without which it would be difficult to group certain items together and still capture the generality at the highest level possible in the hierarchy. The most important factor here too is that all interrelationships be represented explicitly and unambiguously in the knowledge representation language.

vi) One must be able to represent knowledge about the parts making up an entity or about the entities normally associated with it.

So far, some necessary and desirable features of any knowledge representation language have been discussed. But, what is needed to represent a concept, such as a physical object, a function, a class of objects, or virtually anything that one desires to cluster information around? First, the knowledge representation language must be able to describe the attributes of the concept being represented. For example, that a person has a heart and lungs, or that a function operates on two variables in order to derive its result. The use of attributes is common to most representation schemes including KL-ONE, KRL and FRL. Second, the language must be able to express how the attributes interrelate. Saying that a function takes two arguments does not fully capture what the function means. The interrelationship of the two arguments in deriving the final result must also be stated. Among KL-ONE, KRL, and FRL, KL-ONE is the only one to have explicitly captured this need in the representation language, what Brachman calls a structural condition. Lastly, the language must also be able to specify how the concepts themselves interact. Class membership is only one example of the ways in which two concepts may relate to one another.

vii) Default values are necessary in the knowledge language.

Both KRL and FRL allow default values to be associated with attributes. When the value of an attribute is not known, the default value for that attribute may be used if one exists. This makes sense in terms of everyday reality since people rarely have complete knowledge. Default information allows processing to continue even in the absence of certain information.

viii) <u>It should be possible to attach procedures to the objects in the representation</u>.

KL-ONE, KRL, and FRL all allow for attached procedures in the form of a programming language. This does violate the desire to capture knowledge explicitly in the knowledge representation language itself, as meaning will be couched in these routines. However, the benefits to processing because of these procedures is a factor which argues for including them. In FRL, an IF-NEEDED procedure attached to an item of information will be executed if the information is needed. A procedure such as this can be viewed as performing goal directed processing. IF-ADDED and IF-REMOVED procedures are executed when an item is added or removed, respectively. IF-ADDED procedures represent data directed processing and IF-REMOVED procedures are just the negative of these.

II. An Example

To further define and elaborate the features of the knowledge representation language, a few examples are presented: the concept of a function, the specification of the function square root, and a specific call of the function.

Briefly, we may specify square root as a function which takes two required arguments, say x and y, with x greater than or equal to zero and y (the tolerance) at least .005. The result of the function, which is assumed to be positive, has the following relationship to the two input parameters:

$$x-y <= result^2 <= x+y.$$

First, we define the concept of a function in general; square root is a member of that class.

```
(1)  [CONCEPT function CLASS
(2)   <ATTRIBUTE
(3)     (ROLENAME arguments)
(4)     (MODALITY NECESSARY)
(5)     (VALUE/RESTRICTION (THECONCEPT type))
(6)     (NUMBER (>= arguments 1))>
(7)   <ATTRIBUTE
(8)     (ROLENAME result)
(9)     (MODALITY DERIVED)
(10)    (VALUE/RESTRICTION (THECONCEPT type))
(11)    (NUMBER 1)>]
```

Figure 1

The definition itself is enclosed in square brackets, [ ], with keywords capitalized. Line 1 contains the concept's name, i.e. "function". The word CLASS, as opposed to INDIVIDUAL, indicates that the definition of "function" represents an intensional description of a class of objects. Each attribute for "function" is contained within angle brackets, < >. As Brachman (1978) suggests, there is additional information regarding each attribute other than constraints on its values. Each piece of information is surrounded by parentheses, and preceded by the keyword indicating its significance.

The breakdown chosen for the attribute subparts is modeled after those suggested in Brachman's work. The ROLENAME of an attribute names the part or role that the attribute plays in the definition of the concept. It is an arbitrary word that one chooses to aid in the understanding of the description. The possibility for name conflicts does exist, but this will be discussed later. When defining a concept which is a member of a more general class, the ROLENAME is used to allign the attributes. In this example, as lines 3 and 8 indicate, "function" has two attributes whose names or roles are "arguments" and "result".

MODALITY may be employed to indicate how critical an attribute is in the definition of a concept. An instance of "function" must have an attribute called "arguments", as indicated by the keyword NECESSARY on line 4. The keyword OPTIONAL would carry with it the meaning that the particular attribute is not essential in order to have an instance of the concept. For example, a picnic would still be one without ants. The word DERIVED which accompanies the MODALITY on line 9 signifies that the attribute named "result" is not part of "function" per se, but rather is a by-product of the other attributes.

In the definition of a concept, the VALUE/RESTRICTION of an attribute limits the set of legal fillers for a particular role. In this example, line 5 specifies that the only legal values for "arguments" are in the class of "type".

It is possible to have more than one item filling an attribute. NUMBER represents a predicate which must be true of the number of entities filling the particular role in an occurrence of the concept. In an instance of the "function" concept, there must be one and only one "result". Line 6 indicates that there is an unspecified number of "arguments" in an instance of "function". This example helps demonstrate why a NUMBER indicator is needed. First, at the time of defining "function" it is not known just how many entities will fill the role of "arguments" in an occurrence of the concept. Second, the role may potentially be filled by a large (and with some roles even infinite) number of items. One would not want to have an individual attribute for each of these, especially since, as in this case, they all act the same. Conceptually what results is a single name for a group of entities.

The concept "function", being an intensional definition, describes a class or set of concepts. Every time another concept has "function" as its superconcept, it is the same as saying that the new concept represents some subset of the set defined. In the case of an individual (extensional) concept in a class, it defines a set which contains one member.

```
(1)   [CONCEPT two-argument-function CLASS
(2)     {SUPERCONCEPT function
(3)       <ATTRIBUTE DIFFERENTIATE
(4)          (ROLE arguments)
(5)          (NUMBER 1)
(6)          (ROLENAME x)>
(7)       <ATTRIBUTE DIFFERENTIATE
(8)          (ROLE arguments)
(9)          (NUMBER 1)
(10)         (ROLENAME y)>}]
```

Figure 2

The description above is for "two-argument-function", an intensional object. "Two-argument-function" is in the broader class defined by "function". The keyword SUPERCONCEPT on line 2 indicates membership in a class, with the following word specifying the class concept's name. All facts about "two-argument-function", as a subclass of "function", are contained within the set brackets, { }.

Recall that the definition for "function" had a required attribute that played the role of "arguments" in the concept. In "function", the number of items in an instance of the concept that would ultimately play the role of "arguments" was not known. However, with "two-argument-function" it is known that there are precisely two "arguments". One would like to further refine the attribute specification for "arguments" to single out the two required ones for an instance of "two-argument-function". The keyword DIFFERENTIATE on line 3 indicates that the particular attribute is a refinement of some attribute in the hierarchy which potentially stood for more than one item. On line 4, ROLE names the attribute which is being differentiated.

When differentiating an attribute, all pieces of information for the attribute from the more general concept are inherited. Only the NUMBER aspect of the attribute can be replaced. Line 5 declares the attribute as having one and only one item filling that role in an instance of "two-argument-function".

In "two-argument-function", both of the attributes are a result of differentiating the attribute "arguments" in "function". There must be some way to refer individually to the two attributes in "two-argument-function" from concepts still lower in the hierarchy. Using "arguments" would be ambiguous. When an attribute modifies another attribute through differentiation, it is given a new name. Here, as lines 6 and 10 indicate, the attributes are now named "x" and "y".

"Two-argument-function" inherits all attributes from "function" by virtue of being a subset of that class. Only those attributes being modified appear in the definition. "Two-argument-function" could also be in other classes and thereby inherit attributes from them. The problem of name conflicts arises. It is assumed here, as well as in the remainder of the example, that a mechanism for specifying unique path names and for accessing any piece of information exists.

So far the example has clearly concentrated on the aspect of representing the attributes of a concept within the knowledge representation language. How one expresses the interrelationships among the attributes needs further elaboration. Below, the concept for the specification of "square-root" (given earlier) is represented.

```
(1)   [CONCEPT square-root CLASS
(2)     {SUPERCONCEPT two-argument-function
(3)        <ATTRIBUTE RESTRICT
(4)           (ROLE x)
(5)           (VALUE/RESTRICTION (THECONCEPT positive-number))>
(6)        <ATTRIBUTE RESTRICT
(7)           (ROLE y)
(8)           (VALUE/RESTRICTION {(THECONCEPT positive-number)
(9)                                (PREDICATE (>= y .005)
(10)                               (DEFAULT .01)})
(11)       <STRUCTURE
(12)         (AND (<= (- x y) (square result))
(13)              (<= (square result) (+ x y))
(14)              (>= result 0))>}]
```

Figure 3

"Square-root" is a concept which is in the more general class represented by "two-argument-function" and therefore "function". Two of the attributes from "two-argument-function" are being modified. This time they are being restricted as indicated by the keyword RESTRICT on lines 3 and 6. Within a restricted attribute, the NUMBER and VALUE/RESTRICTION may be specified which are in turn added conjunctively to the information in the more general concept. As indicated by lines 8 and 9, the attribute "y" is in the class of "positive number", is greater than or equal to .005, and is a "type" (inherited from above). Line 10 indicates that the default value for "y" is .01. This information would help processing routines reason about "square-root" even though there may not be an instance of the concept.

The purpose of STRUCTURE beginning on line 11 is to indicate how the attributes interrelate. The STRUCTURE, enclosed in angle brackets, contains an expression which is instantiated any time an instance of "square-root" occurs. Attributes from "square-root" participate in the concept within the STRUCTURE by filling ROLEs, thereby indicating how they interrelate.

Square, and, <=, >=, -, and + are assumed to be concepts defined elsewhere; the expression indicates how their roles are filled by position (in this case). Recall that three facts must be true of "square-root". First, the difference between "x" and "y" must be less than or equal to the "result" squared. Second, the "result" squared must be less than or equal to the sum of "x" and "y". Lastly, the "result" must be greater than or equal to zero. We have assumed that a consistent way of viewing concepts as predicates and attributes as arguments can be worked out; this assumption allowed us to state the STRUCTURE as an expression in a LISP-like syntax and semantics. We have left the language details that would make such an assumption valid for the second iteration in defining a knowledge representation language.

The final part of this example is an actual instance of "square-root".

```
(1)  [CONCEPT square-root-of-4 INDIVIDUAL
(2)    {SUPERCONCEPT square-root
(3)       <ATTRIBUTE SATISFY
(4)          (ROLE x)
(5)          (VALUE 4)>
(6)       <ATTRIBUTE SATISFY
(7)          (ROLE y)
(8)          (VALUE .02)>}]
```

Figure 4

"Square-root-of-4" is an extensional concept. The attributes of "square-root" are modified by actually fixing their values. The keyword SATISFY is used to indicate that the filler of the attribute is fixed. Line 5 states that "x" has the value of 4 and line 8 that "y" is equal to 0.02.

III. Conclusions

The basic features of section I seem necessary to any knowledge representation language. However, there are several issues that must be considered for a second iteration in the language.

i) The potential of having multiple views as in KRL seems high, but it is unclear how to add them without sacrificing the semantic clarity of inheritance based on subset relationships.

ii) Establishing a correspondence between concepts and predicates and between attributes and arguments would be very useful, since this would enable one to write STRUCTUREs as expressions. Not only is this compact but it would make the STRUCTUREs more readable as in the square root example.

iii) The user interface and syntax needs careful review and design.

iv) An appropriate implementation has to be undertaken when the second iteration of the language is defined.

References

Bobrow, Daniel G. and Terry Winograd, "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, Vol. 1, #1, 1977, 3-46.

Brachman, R. J., "A Structural Paradigm for Representing Knowledge," Report No. 3605, Bolt Beranek and Newman Inc., Cambridge, MA, 1978.

Roberts, R. B. and I. P. Goldstein, "The FRL Manual", AIM-409, Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, 1977.