KNOWLEDGE BASED LEARNING, AN EXAMPLE

by

Chitoor V. Srinivasan

DCS-TR-90

May 1980

#176

Paper invited for presentation at the
WORKSHOP ON MACHINE LEARNING
July 1980
Carnegie-Mellon University, Pittsburgh, PA.

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, N.J. 08903

KNOWLEDGE BASED LEARNING, AN EXAMPLE

by

Chitoor V. Srinivasan

DCS-TR-90

May 1980

#176

Carnegie-Mellon University, Pittsburgh, PA.

Department of Computer Science

Rutgers, The State University of New Jersey

New Brunswick, N.J. 08903

DCS - TR - 90

KNOWLEDGE BASED LEARNING, AN EXAMPLE

by

Chitoor V. Srinivasan
Department of Computer Science,
Hill Center, Busch Campus,
Rutgers University, New Brunswick, N.J. 08903

ABSTRACT

Domain knowledge is viewed here as consisting of two parts: A domain theory, TH[D], for a domain D, which can be used by a machine to decide whether a given instance, S, belongs to the domain D or not, and domain heuristics, H[D], which can be used by a machine to solve problems in the domain D using the theory TH[D]. A machine that can build theories TH[D] and learn H[D] from the experience of using TH[D] to solve problems in D is called a Knowledge Based Learning machine. The theory formation aspects of a knowledge based learning machine are illustrated here with a simple example. It is argued that such a machine should be endowed with the right kinds of a priori "biases" in order to be able to examine given instances of a domain and formulate domain theories. The example is used here to present a typical set of generally applicable "biases" of a knowledge based learning machine.

May 1980

KNOWLEDGE BASED LEARNING, AN EXAMPLE.

by

Chitoor V. Srinivasan
Department of Computer Science
Hill Center, Busch Campus,
Rutgers University, New Brunswick, N.J. 08903

1. Introduction.

How may a machine "learn" from examples of situations that are presented to it? What may constitute the "knowledge" of a set of such situations? How should the examples be presented to the machine? Are there general principles which a machine can use to acquire the knowledge automatically by examining the examples presented to it, and to use the knowledge so obtained to solve problems in a domain? These are the general concerns of my research.

I view domain knowledge (a domain is a set of situations, or states) as consisting of two parts: A domain theory and domain heuristics. A machine M is called a KBL-machine (Knowledge Based Learning machine), if it can formulate domain theories and learn domain heuristics from the experience of using the theories to solve problems in the domain. The problems will be of the following kind: Given a state (situation) S in the domain change it to a new state S' which is also in the domain, and which in addition satisfies certain other stated conditions. The domain heuristics will be a set of procedures, stated in a suitable procedural language, which a KBL-machine may use to solve problems of the above kind. These procedures will be derived from the domain theory based on the problem

solving experience of a KBL-machine. Methods for using the theory to solve problems in the domain and for learning domain heuristics are discussed in a companion report [1]. The focus in this paper is on the construction of the domain theory. The basic concepts are illustrated through a simple example.

The domain of our example is a set of rectangular arrays of objects (objects are the constants in the domain) having configurations of the kind shown in Figure 1. Let c1, c2, ..., cn be the names of the constants in a given state of the domain, and let W_r denote the domain itself. Each state of W_r will consist of exactly one rectangle of the kind shown in Figure 1. Initially we shall allow

FIGURE 1 ABOUT HERE

rectangles with empty slots like the ones in Figure 1b to be in W_r (the corner slots can never be empty). Later we shall constrain W_r to contain only rectangles with no empty slots. States in W_r will be presented to M one at a time. M is to construct a theory for this domain. The theory should be such that M can use it to recognize all the states that belong to W_r and only the states that belong to it. To express theories of this kind M needs a language. We will use this example to illustrate the nature of this language and to show how M will obtain the language.

M will have a sensory organ, g. When presented with a state W in W_r it will use g to recognize certain properties of the state W. The properties observed by M will be described in a certain language, L_G, called the Ground Language. Let us assume a very simple ground

language having a vocabulary of exactly four relation names:

$$V = \{\text{left-of, right-of, top-of, below}\}.$$

Sentences in the language L_G will have the form "(x r y)", where x and y are names of constants and r is a relation name in V. For a state W in W_r and a sentence like "(c1 left-of c2)", our machine M can use its sensory organ g to find out whether g(W (c1 left-of c2)) is true. If it is then it will add "(c1 left-of c2)" to the description, g(W) of the state W. g(W) will thus contain all the relational forms "(x r y)" for which g(W (x r y)) is true:

$$g(W) = \{(x\ r\ y) \mid g(W\ (x\ r\ y))\}.$$

FIGURES 2 AND 3 ABOUT HERE.

This is illustrated in Figure 2. For the world state W in Figure 2 its description in L_G is g(W) = S. Thus for example, for the state shown in Figure 1a, its description in L_G will be as shown in Figure 3, where c1, c2, ..., etc., are the names of the constants that appear in Figure 1a. We shall use the symbol D_r to denote the set of all descriptions of the states in W_r and use the symbol S to denote a member of D_r:

$$D\_r = \{g(W) \mid (W \text{ is-in } W\_r)\}.$$

Hereafter we will use the word "domain" to denote this D_r, and use the word "world" to denote W_r. The states presented to the machine will be descriptions in L_G, descriptions of the kind shown in Figure 3. The language L_G is initially given to the machine M.

Upon exposure to descriptions of states S in the domain, M will formulate for itself an abstracted description of the structure of the domain D_r. This abstracted description will be stated in a language L_D, called the Domain Language. This situation is illustrated in Figure 2 for the state S = g(W). The abstraction of S is denoted by AB[S], where AB is the abstraction function. The elements of AB[S] will be sentences in the domain language L_D. M will use the abstractions AB[S] to formulate the theory TH[D_r]. Thus the theory itself will be stated in L_D. The domain language L_D will always be a superset of the language L_G. In other words, there will be new domain concepts which can be expressed in L_D but not in L_G. Thus in our domain D_r, L_D may have concepts like "corner elements", "interior elements", etc., whereas L_G does not have these concepts. M will construct new concepts like these based on the abstractions AB[S] of the states of the domain.

Each sentence in TH[D_r] may be interpreted as expressing a constraint on the states that may exist in D_r. Given an arbitrary sentence, x, in L_D and a state S, if the machine is asked "is x true of S", then it will use its semantic function, Phi, and evaluate Phi(x S). If Phi(x S) is true then "x is true of S", otherwise it is not. M can use the theory TH[D_r] and Phi to find out whether an arbitrary world state that is presented to it is in W_r or not: It can for example assert that if S = g(W) and Phi(x S) is true for all x in TH[D_r] then W is in W_r. Otherwise, it is not. The construction of such a theory for W_r is illustrated in this paper.

The methods of learning and theory formation illustrated here are quite different from those discussed by Winston [2], Hayes-Roth [3], Michalski [4], Mitchell [5] and others. The theories discussed by Brown [6] and Lindsay [7] are special cases of theories discussed here. None of tne above systems can exhibit the kinds of learning abilities shown in this paper.

An essential aspect of this learning is the discovery of concepts that are natural to a domain. In this aspect this work is close to the work of Lenat [8]. Whereas Lenat focussed on set theory to exhibit a spectacular learning ability, we focus on general principles of organization and associated mechanisms. The theory formation processes illustrated here are generally applicable to all domains, even though the discussion in the paper is just on one example. In a certain sense, because of their generality, the processes illustrated in this paper are computationally weak processes. However, as discussed in the companion report [1] they have the potential to specialize themselves to a domain based on the domain theories (partial domain theories) which they may know.

Our KBL-machine is a highly biased one. It will seek only certain kinds of specialized information from its environment, information that is limited by its vocabulary V and its sensory organ g. During its theory formation process it may selectively ask for certain specific information. For example a question of the following type may occur: "Can such and such situation occur in a state of the domain?". Here the machine may obtain the candidate situation by generating a counterexample to one or more of the conjectures it may

have concerning the nature of a domain. It will abstract the information it obtains in specialized ways and use the abstractions to formulate a theory of what it has seen, again in certain specialized ways. The built-in biases controlling the types of abstractions and biases concerning the nature of theories it can construct are essential aspects of the theory formation process.

I exhibit here one set of built-in biases. These biases and mechanisms associated with them can be used to define a meta-theory of theory building machines of the kind discussed here. The meta-theory itself can be used to define the properties of completeness and consistency of the theories, to identify the assumptions on domains tnat make theory construction possible, and to define the semantics of tne constraint description language and the procedural language that is used by M to express its domain knowledge. Logical foundations of this meta-theory and these various aspects of its use are discussed in the companion report [1]. A major objective of this research is to identify generally useful biases of the kind illustrated here, which a KBL-machine can successfully use in a variety of domains.

2. Equivalence Classes of Constants and the Theory of Dimensions.

I shall present here an informal development of the theory construction process. The formal development appears in the companion report [1]. Let me begin with the first bias:

> BIAS 1: Attempt to classify the constants in a state into equivalence classes that are "natural" to the domain.

This is the first task of our machine. It will use the following

rule: If two constants appear to have "similar" descriptions in the language L_G then assert that they should be equivalent to each other. Let us define this notion as follows: For a constant, c, define its predicate signature in a state S to be R(c S),

$$R(c\ S) = \{r \mid ((\text{EXISTS } y)(c\ r\ y))\},$$

and define two constants c and d to be equivalent to each other in the state S if R(c S) = R(d S) (For a more complete definition of this concept see [1]). Applying this rule to the constants that appear in Figure 1a we will get the predicate signatures and equivalence classes shown in Figure 4; each equivalence class in this case is a singleton class. However, if we apply the same definition to Figure 1b we will get classes with more than one element in them. Thus, the class "Left Boundary" (L) in Figure 1b is {c6, c16}, and the class "Interior" (I) is {c8, c9, c12, c14, c17, c18, c19}, etc. But the classes that contain the corner elements are always singleton classes.

This is a "natural" classification of the elements that occur in tne state. There are several reasons why this classification scheme has the tendency to produce classification of constants in a domain into types that are natural and intrinsic to the domain. But we shall not enter into a discussion of this at this place (see [1] for some comments). The KBL-machine will use these classes as its initial basis for observing patterns in the states that are presented to it. It will first use the classes to define an abstraction function as follows:

ABSTRACTION AND DIMENSION:

1. The abstraction of a constant, c, called the dimension of c, written as dimension(c), is the equivalence class to which it belongs. Thus, the dimension of c1 in Figure 1a is the equivalence class LT (see Figure 4). Every constant that appears in a state will have a dimension associated with it.

   AB(c) = dimension(c) = The Equivalence Class of c.

2. The abstraction of a relational form "(x r y)" is the form "(AB(x) r AB(y))". The dimension of a relational form is its abstraction.

Our machine M may now use this concept of abstraction to define the following augmented description language for the domain. Let us call this the Language of Dimensions, Ld_D. It should be noted that this Ld_D is not the same as the domain language L_D introduced earlier. In general, Ld_D will be a subset of L_D. As the theory construction process continues to develop the machine may add to Ld_D additional terms and sentential forms.

LANGUAGE OF DIMENSIONS, Ld_D:

1. Ld_D will contain the following new constants: LT, RT, LB, RB, L, R, T, B and I. These are the names of the equivalence classes shown in Figure 4. Notice that these names of constants do not exist in L_G, and each one of these constants represents a set, a concept that L_G does not have. At this point, this is a concept that is entirely local to the KBL-machine. It is based on the particular classification scheme that the machine uses.

2. For every statement that is in L_G, Ld_D will contain that statement and its abstraction as sentences of Ld_D. For our example here let us define AB(AB(x)) = AB(x), for all x in Ld_D.

M may now use this concept of dimension (abstraction) to notice the following: There are dimensional relational forms that can never occur in the domain. Thus for example. "(L right-of R)" is a dimensional relational form that can never occur in any of the states

of D_r. To take advantage of this new property our KBL-machine will define tne following notion of dimensional consistency associated with tne relational forms that may occur in the states of D_r:

DIMENSIONAL CONSISTENCY:

A relational form, x, in Ld_D is dimensionally consistent, D-consistent(x), if and only if there exists a state S in D_r and a relational form y in S, such that AB(y) = x.

Already, Ld_D is richer than L_G. It now has the function term dimension in its vocabulary and the new predicate symbol D-consistent. It is not hard to now define truth values for statements of the form "D-consistent(x)" in a given state S. This will define the semantic function, Phi, for the language Ld_D, namely the truth value of expressions of the form Phi(x S) for x in Ld_D (See [!] for details). The set of all dimensionally consistent relational forms that use the various equivalence classes in the domain D_r is shown in Figure 5.

FIGURE 5 ABOUT HERE

The language enrichment scheme we just outlined above is a general scheme which is a part of the KBL-machine. It will apply this scheme to all the domains that it encounters. By giving different vocabularies to the machine and different sensory organs we may now ask the machine to examine different domains in this manner. This may now be stated as the second bias of M:

BIAS 2: Define the language Ld_D for the domain.

The machine will now use Ld_D and the classes in D_r to notice new patterns that occur in the states presented to it. The most

elementary of tne patterns that it may seek is,

BIAS 3: See if every equivalence class occurs in every state.

With this bias, using circumscription [9], the machine may soon produce the following initial theory:

INITIAL THEORY: D_r has the equivalence classes shown in Figure 4, namely the classes LT, RT, LB, RB, L, R, T, B, and I. A state S belongs to D_r if and only if none of the above equivalence classes in S is empty.

The machine now has the task of seeking counterexamples to this initial theory. But as of this point it does not have the ability to generate examples which may violate this conjecture and then enquire whether such examples may belong to the domain. All it could do is to present its identification of the various equivalence classes and enquire whether a state can occur in D_r with one or more of the equivalence classes being empty. Certainly a question of this kind is highly pertinent and natural to the domain. To make this discussion brief and proceed on to further ramifications of this process let us just assume that the above initial theory is the correct theory for D_r. Clearly, M can now use this theory to correctly identify all the world states in W_r. Given an arbitrary world state, W, it will first construct g(W) and then test whether g(W) satisfies the above theory. If it does then M will accept W as a member of W_r, otherwise it will not. Thus, the states shown in Figure 6 will all be rejected by our machine.

FIGURE 6 ABOUT HERE

Let me hasten to point out a subtle difference here between recognizing the states that belong to D_r and recognizing the world states that belong to W_r. Notice that in W_r, from the nature of the natural physical laws, it will be always true that relational forms like "(x left-of y)" and "(x right-of y)" can never be simultaneously true for any of the constants in a state. Our theory as presented above will allow such forms to be simultaneously true for the Interior constants in a state. This is because our machine does not yet know that

(x left-of y) <=> (y right-of x).

Thus the initial theory given above is adequate only to characterize the set W_r, but not the set D_r. Further analysis is necessary to do this. M will now undertake this further analysis fully exploiting all it now knows about the theory of dimensions associated with the domain D_r, namely the theory consisting of the language Ld_D and the concepts of D-consistency and dimension.

Throughout its theory construction process M will at every stage attempt to make full use of what it may already know about a domain. It is in this sense that M is a KBL-machine. M comes with the right kinds of initial biases and processors that enables it to bootstrap itself into this kind of theory construction process.

One may note that the theory of dimensions cannot capture logical constraints of the form shown above. To generate examples of descriptions of states that may occur in W_r it is necessary that we have a theory that characterizes D_r. It is also necessary to define

appropriate instantiation processes, which can correctly make use of M's theory of a domain. We shall not enter into a discussion of the instantiation processes in this paper (see [1] for a discussion of tnis process). In general it is not enough if the theory produced by M only characterized W_r. Thus M will always undertake the task of characterizing D_r. It will do this by representing the descriptions of states that it encounters using certain graphical forms. By choosing to analyze the states using graphical forms, which are introduced in the next section, we are in effect endowing our KBL-machine with the following built-in new concepts:

1. the concept of relation composition, i.e. that if M knows that (x r y) and (y r1 z) are both true in S then the new form (x r.r1 y) is also true in S.
2. the concept of cardinality of sets and lengths of relation paths.

Let us now proceed to consider the graphs constructed by M and the way M will analyze the graphs to identify new patterns and concepts that are relevant to given domains.

3. Graph Abstractions and Constraint Induction.

Let us modify our world W_r and make it contain a more restricted class of rectangles, say only the rectangles that have no empty grid points. Let W_r' denote this modified world and D_r' denote the set of descriptions of the states in W_r'. Our language Ld_D does not have tne necessary power to express this new restriction. Let us now consider an embellishment to the language that will make it possible to express this concept.

Let W be the state in Figure 1b, and let S be the description of W in L_G using the constant names c1 through c25. Then the constant c1 will have the following description (again, we are using the convention we used in Figure 5, to write down the relations). Also, we have further enriched Ld_D by adding sentential forms with logical connectives to it):

$$[(c1 \text{ left-of } (c2, c3, \ldots, c23, c25) \;\&\; (c1 \text{ top-of } (c6, c8, \ldots, c23, c25)) \;\&\; (c1 \text{ member-of } LT)]. \qquad (3.1)$$

Among the properties given above only the first two are observable properties. The last property is internal to the machine: It has significance only with reference to the way the machine had characterized the class LT (namely, the predicate signature of LT). Notice that this description uniquely denotes the object 1 in the scene, since object 1 is the only object that satisfies all the properties in the description. We shall say that object 1 has an individual description in the language Ld_D. Objects with individual descriptions in a language will generaly play an important role in the formulation of theories expressed in that language. Our machine will thus have a bias to view given states of a domain in terms of the objects that have individual descriptions. We now have the fourth bias,

> BIAS 4: Examine given states to see whether there are constants in the state that have individual descriptions. Use such constants to uniquely identify as many of the other constants that exist in the state as is possible, using notions of relation composition.

It should be noted that the concept of individual description is an essential concept that M should have in order to be able to relate its

own internal descriptions of constants in a state to the objects that actually exist in a world state. For example the machine should be able to identify the object "1" in Figure 1b with the name c1 in the description of Figure 1b in its memory. To give such a denotational interpretation to names in a description it is necessary to have the ability to generate individual descriptions. Thus, M will now attempt to characterize the objects in W, using c1 as its reference point. This will be done as follows.

First it will construct a tree of the kind shown in Figure 7, called the

FIGURE 7 ABOUT HERE.

connection tree, C-tree(c1) (see section 7 in Part III of [1] for definition of connection tree) with c1 as its root node. This tree contains only the relation names that belong to the predicate signature of c1, namely R(c1 S). This is coincidental. Normally trees like this will be constructed using a set of relation names that include R(c1 S), but do not include any of the converses of the relations in R(c1 S). Our machine does not yet have the concept of relation converses. This is not a hard concept to identify. In fact, the meta-theory discussed in [1] assumes that this is one of those a priori concepts that is given to KBL-machine at the time of its birth.

Every node in Figure 7 represents an object in Figure 1b. When a constant, c, occurs more than once in the tree, all the nodes with label c other than the first are terminated. The underlined nodes in Figure 7 denote such termination points. The immediate descendents of

a node x, namely its daughter nodes, are nodes y and z, if (x left-of y) and (x top-of z) are respectively true, and there is no relation path r1.r2...rk, k > 1, consisting only of the relation names "left-of" and "top-of", such that (x r1.r2...rk y) or (x r1.r2...rk z) is true. In other words, the daughter nodes of a node x are its immediate neighbors under the "left-of" and "top-of" relations.

We used two important pieces of information to construct this tree: one is the choice of the root node and the other is the choice of the relation names that appear in the tree. The constant c1 is chosen as the root node because there is an individual description for it in the language Ld_D (see (3.1)). The choice of "left-of" and "top-of" as relation names for the tree follow naturally from the choice of c1 as the root. These are the relations in R(c1, S). The other relations in the vocabulary of L_G are converses of the relations in R(c1, S). They are therefore not included in the tree. Clearly, C-tree(c1) does not represent all the information that is available in S. It presents one possible view of S, the view from c1 with respect to "left-of" and "top-of" relations. We shall say that C-tree(c1) is an abstraction of S with respect to the set {c1, left-of, top-of}, and denote it uniquely by

C-tree(c1, S ; {left-of, top-of}).

In this case C-tree(c1) covers all the objects in the scene shown in Figure 1b. In addition, it also provides an individual description for every object in the scene of Figure 1b in terms of the constant c1. Thus, for example, "(c1 left-of.left-of c3)" uniquely identifies the object 3 in the scene relative to the object 1. A set of relation

names which may thus be used to uniquely resolve all the objects in a scene (through individual descriptions for the objects) is called a prime set of the scene. We believe that prime sets are likely to play an important role in theory formation. Methods for the discovery of prime sets should be a part of KBL-machines. Thus, the 5th bias of a KBL-machine is,

> BIAS 5: Identify the Prime sets of relation names associated with given states.

The criteria that was used for selecting the information, that was included in this tree, was based on certain preconceived notions on the importance of individual descriptions and prime sets that our KBL-machine had. Notions like these may be stated in general terms, in a manner that is applicable to large classes of situations in a domain. The graph abstractions that are used for constraint induction will all be based on "preconceived notions" like the ones above. I believe that there are classes of interesting "preconceived notions" which are useful for constraint induction in varied domains. Every KBL-machine should have notions like these built into them. This research is partly concerned with the discovery of such notions, notions that are likely to be useful for large classes of empirical situations.

Let us now consider how our machine might further analyze this tree. One may define a distance between ordered pairs of nodes in C-tree(c1) as follows: the daughter nodes of a node x are by definition at distance 1 from x. A node y is at a distance n from x, if it is at distance (n-1) from one of the daughter nodes of x. This

notion of distance may in turn be used to define a size, (n m), for a rectangle in a natural way. It may be noticed that for the above connection tree the distance of a node y from c1, say Distance(c1 y), may not be unique, and for some pairs of nodes the distance is undefined. Thus, for example Distance(c1 c18) has two possible values in Figure 7, namely 3 and 4, depending on which route is taken, and for nodes c10 and c8, Distance(c10 c8) is undefined. Also, this tree has some empty nodes. In fact, it is the presence of empty nodes that causes the "distance" function to be non-deterministic. If the rectangular array had no empty slots then Distance(c1 y) will be unique for all y. It is easier to notice and express these properties in terms of the connectivities in the directed graphs shown in Figures 8 and 9 below. These graphs are obtained from the connection tree.

The graph in Figure 8 is the connection graph, C-graph(c1) (see section 7 of Part III of [1] for a general definition of these graphs). C-graph(c1) is obtained by merging together the nodes in C-tree(c1) that have identical labels. The root node of C-tree(c1) is the anchor of C-graph(c1). The graph in Figure 9 is called the dependency graph of c1, D-graph(c1 LT). It is obtained from

FIGURES 8 AND 9 ABOUT HERE.

C-graph(c1) by replacing each node label by the class to which the label belongs. It may be noted that each node in the dependency graph D-graph(c1 LT) represents a distinct occurrence of a constant. Thus, two nodes in Figure 9 with the same label, say label I, will represent two distinct occurrences of constants belonging to the class I.

Now, if relations "right-of" and "below" are superimposed on the D-graph in figure 9, in the right manner, then one may use this augmented D-graph to identify the pairs (r r') that are converses of each other, and relations that may be symmetric (antisymmetric), transitive and/or reflexive (irreflexive) in a given state. Graphs like these may also be used to discover implicit definitions of the kind "father.father = grandfather" that are considered by Brown and Lindsay [6, 7]. We shall not here discuss the details of the search processes involved in these tasks. It may be noted that the search will not involve expensive intersections as was the case in the analyses proposed by Brown and Lindsay. Let us now consider how our system may use graphs like these to characterize the modified domain W_r'.

For every state W in W_r' the "left-top-corner" element, say c', will be unique, and for every y in W the Distance(c' y) will be unique. For a state, W' in W_r but not in W_r', the distance function is not unique for all y in W'. Uniqueness properties like these may be noticed immediately using the D-graphs like the one shown in Figure 9. They may be expressed as a sentence in a formal language as follows (Sentences like the one below will be used in the domain language L_D to state domain constraints):

```
PROPERTY P1:

  [((C)(C member-of {LT, RT, LB, RB, L, T, R, B, I}) =>
       ((EXISTS c)(c occurs-in S) & (c member-of C))) &

                For every equivalence class there is a
                constant in S belonging to that class.

   ((EXISTS c)(c occurs-in S) =>
    ((EXISTS C)(C member-of {LT, LB, RB, RT, L, T, R, B, I})
          & (c member-of C))) &
```

Every constant in S should belong
to some equivalence class.

```
((EXISTS unique x)(x member-of LT) &
 ((Y)(Y member-of {RT, LB, RB}) =>
          ((EXISTS unique y)(EXISTS unique n)
   (y member-of Y) & (Distance(x y) = n))) &
```

Corner elements are unique and are
situated at unique distances from LT.

```
 ((Z)(z)(Z member-of {L, R, B, T, I})
       (z member-of Z) =>
  ((EXISTS unique m)(Distance(x z) = m))))].
```

Every other element should also
be at a unique distance from LT. (3.2)

Let D_r' be the set of descriptions in L_G of the states in W_r'. Then, after examining a sufficient number of sample states of the domain the machine may assert by circumscription,

```
((S)(S member-of D_r') <=> P!(S)).
```
(3.3)

We believe that it is possible to specify mechanisms that could identify invariant properties like these and express them as sentences in a language of the kind shown above. Clearly, in order to assert statements like (3.3) a machine should have both positive and negative instances of states of a domain. An interesting property of machines like these is that they may actively seek for negative instances in order to ascertain a current hypothesis. We have here overlooked the problems involved in constructing the right kinds of graph abstractions, searching the graphs for invariant properties, and formulating them in a language of the above kind. Some general approaches to solving these problems are discussed in section 7 of Part III of [1]. But much work remains to be done.

In this paradigm the searches for domain constraints will always take place over suitably constructed dependency graphs of a domain. We generalize dependency graphs to include function terms also, besides the constants. Every such dependency graph will represent an abstraction of a state that belongs to the domain. Each such abstraction may introduce new symbols into the domain language L_D (relation names, function names and constants). In each such abstraction the search for invariant properties will be specified in terms of recognition of certain recurring graph patterns. These graph patterns may consist of "loops", convergent branches of "relation paths", and other such identifiable features of the graphs. Methods for organizing the search for the relevant patterns that may occur in a given graph, remain yet to be worked out. We also need schemes for organizing sets of constraints, that a machine may have for a domain, in a manner that is useful for problem solving in the domain. We discuss in section 7 of Part III of [1] one possible organization for the constraints. It is quite similar to the one proposed in MDS [10, 11] and implemented in AIMDS [12].

Let me now conclude this discussion with a few general comments on the research objectives and the approaches outlined above. It may be noted that theories of the kind illustrated above have a meta-theory and the meta-theory may be used to identify control structures that enable one to state and solve problems in a domain using theories of this kind, and in addition learn domain heuristics from the problem solving experience. We introduce in [1] the concept of <u>theories with explanatory capabilities</u> and discuss mechanisms that are needed to generate explanations (reasons for failure or success of

given problem solving steps). The ability to generate explanations plays a central role in giving the KBL-machine the capability to use the theory to solve problems and learn from this experience. We conclude now with a few general comments.

4. Concluding Remarks.

The theory formation scheme illustrated above may be characterized in terms of three basic processes which every KBL-machine should have: an abstraction process AB (and others like it), the semantic function Phi and an instantiation process, say I. The nature of these processes and their use in formulating a meta-theory of KBL-machines are both discussed in [1]. Suitably defined general classes of abstractions, truth evaluation procedures and instantiation processes provide the foundations necessary for the construction of KBL-machines. These also set the basis for endowing such machines with the capability to generate explanations and use the explanations for problem solving and planning, as is shown in [1].

The fundamental point of view I wish to advocate here is that theory formation tasks of the kind illustrated here can take place in realistic domains, if and only if the KBL-machine is initially endowed with the right kinds of generally applicable biases. I believe that such biases exist and they should be a part of any learning system. The study of theory formation itself is fundamental to the design of learning systems and knowledge based systems for the reasons cited below.

I posit tnat complex problem solving should necessarily involve abstractions, planning, and the use of models and explanatory capabilities, both to guide reasoning processes and learn from experience. Humans routinely use these methods in most of their problem solving activities.. It seems the power that humans accrue from using these methods arises not as much from the methods themselves, as from the underlying theory forming abilities that support their use and gives them the necessary flexibility and generality. They are able to use their theory forming abilities to refine and modify their methods, and summarize their own experiences in useful ways. The theory forming capability also supports their ability to generate explanations, perform successfully in the context of incomplete information, and accumulate the pragmatic knowledge that is necessary to use their theories successfully. Indeed, one could assert that the ability to do abstraction, planning, generate and use explanations, and construct and use models, and learn from this experience, are all necessary and inevitable consequences of the innate theory forming capabilities that humans have. The meta-theory gives credence to this hypothesis (see discussion in [1]), because in the meta-theory these various problem solving and learning abilities arise naturally as a consequence of using the intrinsic ability to formulate and use domain theories.

Any attempt to endow machines with similar problem solving capabilities should necessarily rest on a sound understanding of the underlying theory forming methods that are necessary to support such problem solving techniques. Otherwise we shall forever be limited only to emulating the superficial features of such behaviors in given

domains, through programming (knowledge engineering). Every time the domain changes we have to start all over again in our tasks of knowledge engineering. We could not approach general intelligence by taking this superficial route. The rich experience we now have in "knowledge engineering" Brachman [13], Feigenbaum [14], Davis [15], Bobrow [16], Sridharan [17], Davis [18], Roberts [19], Schank [20], Woods [21], Charniak [22], Srinivasan [10, 11] Stallman [23] gives us some perspectives to begin work on a meta-theory of automatic empirical theory formation systems, and inquire into the nature of such theories and the nature of KBL-systems.

Systems like ABSTRIPS [24] use planning within an abstraction hierarchy to cut down search. There are others that use abstraction and generalization to exhibit rudimentary capabilities for learning from examples Winston [2], Vere [25], Hayes-Roth [3], Mitchell [5], Michalski [4, 26]. Lenat [8] used abstraction and generalization very successfully to exhibit a system that could discover new mathematical concepts in an elementary domain of sets and numbers. We also have problem solving systems that use domain knowledge and abstraction within a pre-defined hierarchy to guide their search Schank [27], Bobrow [16], Roberts [19], Davis [15], Goldstein [28], Brachman [13], Soloway [29]. In certain systems we have all these facilities being used in an integrated fashion within specific cleverly designed problem solving contexts Feigenbaum [30], Buchanan [31]. These successful special purpose systems have shown the power of incorporating domain knowledge in problem solving processes. These systems have however two debits: They are very difficult to implement and are inflexible. To overcome these difficulties it is necessary

that we first understand the nature of machines that can formulate their own theories and can learn from the experience of using their theories to solve problems. The biases built into such systems provide the basis for formulating a meta-theory of such theory building and problem solving systems. My research is concerned with the logical foundations of such systems.

5. Acknowledgements.

I wish to thank David Sandford for the discussions I had with him and for his comments.

6. Bibliography.

[abbreviations:
MI = Machine Intelligence, vols. 4 thru 7, Meltzer, B. and Michie, D. (eds.), Halsted Press, Wiley, New York, 1969 thru 1972.
vol. 9, Hayes, J. E., Michie, D. and Mikulich, L. I., (eds.), Chichester: Ellis Horwood, 1979.

AI = Artificial Intelligence (journal),

IJCAI = International Joint Conference on Artificial Intelligence, Proceedings:
IJCAI-3, SRI International, Menlo Park, Ca., 1973;
IJCAI-4, Artificial Intelligence Laboratory, Cambridge, Mass., 1975;
IJCAI-5, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1977]

1. Srinivasan, C. V. and Sandford, D. M., "Knowledge Based Learning Systems: Part I: A Proposal for Research",

Srinivasan, C. V., "Knowledge Based Learning Systems, Part II: An Introduction to the Meta-Theory", and "Part III: Logical Foundations",

DCS-TR-89, Department of Computer Science, Hill Center, Busch Campus, Rutgers University, New Brunswick, N.J. 08903, March 1980.

2. Winston, P. H., "Learning Structural Descriptions from Examples", The Psychology of Computer Vision, Winston (ed.), McGraw-Hill, New York, pp. 157-209, 1975.

3. Hayes-Roth, F. and McDermott, J., "An Interference Matching Technique for Inducing Abstractions", C. ACM, 21, 5, pp. 401-411, 1978.

4. Michalski, R. S., "A System of Programs for Computer-aided Induction: A Summary", IJCAI-5, pp. 319-320, 1977.

5. Mitchell, T. M., Version Spaces: An Approach to Concept Learning, Ph.D. thesis, Dept. of Electrical Engineering, Stanford University, 1978.

6. Brown, J. S., "Steps Toward Automatic Theory Formation", IJCAI-3, pp. 121-129, 1973.

7. Lindsay, R. K., "Inferential Memory as the Basis of Machines which Understand Natural Language", Computers and Thought, Feigenbaum and Feldman (eds.), McGraw-Hill, New York, pp. 217-233, 1963.

8. Lenat, D. B., "The Ubiquity of Discovery", AI 9, pp. 257-285, 1977.

9. McCarthy, J. and Hayes, P. J., "Some Philosophical Problems from the Standpoint of Artificial Intelligence", MI 4, pp. 463-502, 1969.

McCarthy, J., "Epistemological Problems of Artificial Intelligence", IJCAI-5, pp. 1038-1044, 1977.

McCarthy, J., "First Order Theories of Individual Concepts and Propositions", MI 9, 1979.

10. Srinivasan, C. V., "The Architecture of Coherent Information Systems: A General Problem Solving System", IJCAI-3, pp. 618-628, 1973; also in IEEE Trans. on Computers, vol. C-25, 4, pp. 390-402, 1976.

11. Srinivasan, C. V., "The Meta-Description System: A System to Generate Intelligent Information Systems, Part I, The Model Space", Tech. Report SOSAP-20A, Dept. of Computer Science, Rutgers University, New Brunswick, N.J., 1977.

12. Sridharan, N. S., AIMDS User Manual - Version 2, CBM-TR-89, Department of Computer Science, Rutgers University, New Brunswick, N.J., 1978.

13. Brachman, R. J., "A Structural Paradigm for Representing Knowledge", Ph.D. thesis, Harvard University, 1977, also BBN technical report 3605, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1978.

14. Feigenbaum, E. A., "The Art of Artificial Intelligence: I. Themes and case studies of knowledge engineering", IJCAI-5, pp. 1014-1029, 1977.

15. Davis, R. and Lenat, D., Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, in press, 1980.

16. Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H. and Winograd, T., "GUS, A Frame-Driven Dialog System", AI 8, 2, pp. 155-173, 1977.

17. Sridharan, N. S. and Schmidt, C. E., "Knowledge-Directed Inference in BELIEVER", in Pattern-Directed Inference Systems, Waterman and Hayes-Roth (eds.), Academic Press, New York, 1978.

18. Davis, R., Buchanan, B., Shortliffe, E., "Production Rules as a Representation for a Knowledge-Based Consultation Program", AI, 8, 1, pp. 15-46, 1977.

19. Roberts, R. B. and Goldstein, I. P., FRL Users' Manual, MIT AI Laboratory, Memo No. 408, Cambridge, Mass., 1977.

20. Schank, R. C. and Abelson, R. P., Scripts, Goals, Plans, and Understanding, Lawrence Erlbaum, Hillsdale, N.J., 1977.

21. Woods, W. A., "What's in a Link: Foundations for Semantic

Networks", in Representation and Understanding, D. G. Bobrow and A. Collins (eds.), Academic Press, New York, pp. 35-82, 1975.

22. Charniak, E., "Organization and Inference in a Frame-like System of Common Knowledge", in Theoretical Issues in Natural Language Processing, R. C. Schank and B. L. Nash-Webber (eds.), Mathematical Social Sciences Board, Cambridge, Ma., pp. 46-55, 1975.

23. Stallman, R. M. and Sussman, G. J. , "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", AI 9, 2, pp. 135-196, 1977.

24. Sacerdoti, E. D., A Structure for Plans and Behavior, Elsevier, New York, 1977.

25. Vere, S., "Induction of Concepts in the Predicate Calculus", IJCAI-4, pp. 281-287, 1975.

26. Michalski, R. S., "Pattern Recognition as Rule-Guided Inductive Inference", paper dated September 1979 and accepted for publication, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979.

27. Schank, R. C., "The Structure of Episodes in Memory", in Representation and Understanding, D. G. Bobrow and A. Collins (eds.), Academic Press, New York, pp. 237-272, 1975.

28. Goldstein, I. P. and Roberts, R. B., "Nudge, a Knowledge-Based Scheduling Program", IJCAI-5, MIT, Cambridge, Mass., pp. 257-263, 1977.

29. Soloway, E. M. and Riseman, E. M., "Levels of Pattern Description in Learning", IJCAI-5, MIT, Cambridge, Mass., pp. 801-811, 1977.

30. Feigenbaum, E. A., Buchanan, B. G. and Lederberg, J., "On Generality and Problem Solving: A Case Study Using the DENDRAL Program", MI-6, pp. 165-190, 1971.

31. Buchanan, B. G., Feigenbaum, E. A. and Sridharan, N. S., "Heuristic Theory Formation: Data Interpretation and Rule Formation", MI-7, pp. 267-290, 1972.

1	2	3
4	5	6
7	8	9

1a

1	2	3	4	5
6	-	8	9	10
-	12	-	14	-
16	17	18	19	20
21	22	23	-	25

1b

Figure 1: Two instances of configurations in the world W_r of rectangular arrays. "-" in the arrays denote empty slots.

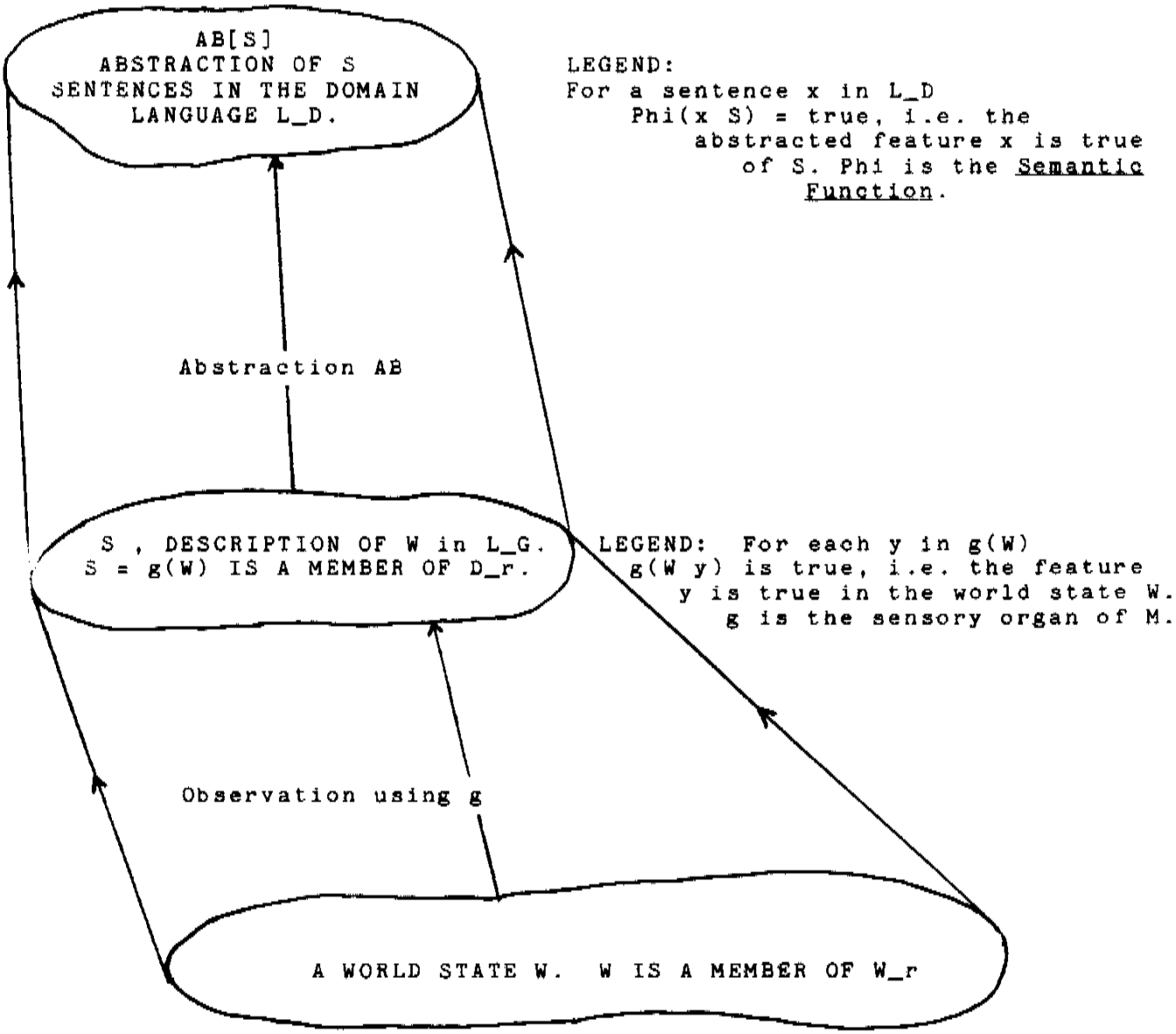

Figure 2: World State, Domain State and its Abstraction.

```
{(c1 left-of c2),(c2 right-of c1),(c3 right-of c2),
 (c1 left-of c3),(c2 right-of c4),(c3 right-of c1),
 (c1 left-of c5),(c2 right-of c7),(c3 right-of c4),
 (c1 left-of c6),(c2 left-of c3), (c3 right-of c5),
 (c1 left-of c8),(c2 left-of c6), (c3 right-of c7),
 (c1 left-of c9),(c2 left-of c9), (c3 right-of c8),
 (c1 top-of c4), (c2 top-of c4),  (c3 top-of c4),
 (c1 top-of c5), (c2 top-of c5),  (c3 top-of c5),
 ..............., ...............,  ...............,
 (c1 top-of c9), (c2 top-of c9),  (c3 top-of c9),

 (c4 left-of --),(c5 left-of --), (c6 top-of --),
 (c4 below --),  (c5 right-of --),(c6 right-of --),
 (c4 top-of --), (c5 top-of --),  (c6 below --),
                 (c5 below --),

 (c7 left-of --),(c8 left-of --), (c9 right-of --),
 (c7 below --),  (c8 right-of --),(c9 below --),
                 (c8 below --)}.
```

Figure 3: Domain state corresponding to the world state of Figure 1a. "--" denotes the existence of constants.

Left Top corner: LT = {c1}, R(c1) = {left-of, top-of}.

Right Top corner: RT = {c3}, R(c3) = {right-of, top-of}.

Lt. Bottom corner: LB = {c7}, R(c7) = {left-of, below}.

Rt. Bottom corner: RB = {c9}, R(c9) = {right-of, below}.

Top Boundary: T = {c2}, R(c2) = {left-of, right-of, top-of}.

Left Boundary: L = {c4}, R(c4) = {left-of, top-of, below}.

Right Boundary: R = {c6}, R(c6) = {right-of, top-of, below}.

Bottom boundary: B = {c8}, R(c8) = {left-of, right-of, below}.

Interior objects: I = {c5}, R(c5) = {left-of, right-of, top-of, below}.

Figure 4: The Equivalence Classes in Figure 1a.

(LT left-of R), (LT left-of RT), (LT left-of RB),
(LT left-of I), (LT left-of T), (LT left-of B).

For convenience let us write these forms together in one statement, as "(LT left-of (R, RT, RB, I, B, T))". We then will have the following D-consistent forms in the domain W_r:

```
(LT left-of  (R, RT, RB, I, T, B)),
(LT top-of   (L, I, R, LB, B, RB)),
(RT right-of (LT, L, T, I, BL, B)),
(RT top-of   (L, I, LB, B, RB, R)),
(LB left-of  (T, RT, I, B, RB, R)),
(LB below    (LT, L, RT, R, I, T)),
(RB right-of (LT, LB, T, B, L, I)),
(RB below    (LT, T, RT, R, L, I)),
(T  left-of  (RT, R, RB, T, I, B)),
(T  right-of (LT, L, LB, I, B, T)),
(T  top-of   (L, R, B, LB, RB, I)),
(L  left-of  (T, RT, I, R, B, RB)),
(L  below    (LT, T, RT, ...)),
(L  top-of   (LB, B, RB, ...)),
(R  right-of (LT, L, T, I, LB, B)),
(R  top-of   (LB, B, RB, ...)),
(R  below    (LT, T, RT, ...)),
(B  below    (LT, T, RT, L, I, R)),
(B  left-of  (RT, R, RB, ...)),
(B  right-of (LT, L, LB, ...)),
(I  right-of (LT, L, LB, I, T, B)),
(I  left-of  (RT, R, RB, I, T, B)),
(I  top-of   (LB, B, RB, I, L, R)),
(I  below    (LT, T, RT, I, L, R))
```

Figure 5: The dimensions of the domain W_r.

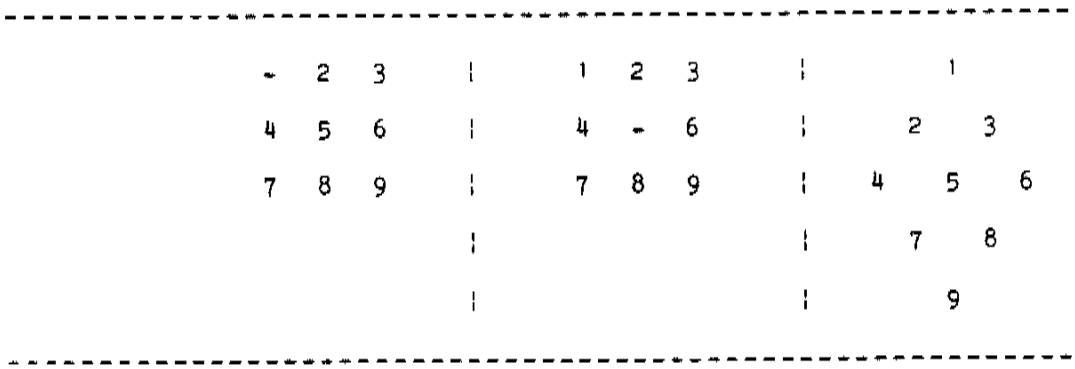

Figure 6: Examples of configurations that are rejected by the theory.

In the tree below

"l" denotes "left-of" and "t" denotes "top-of".

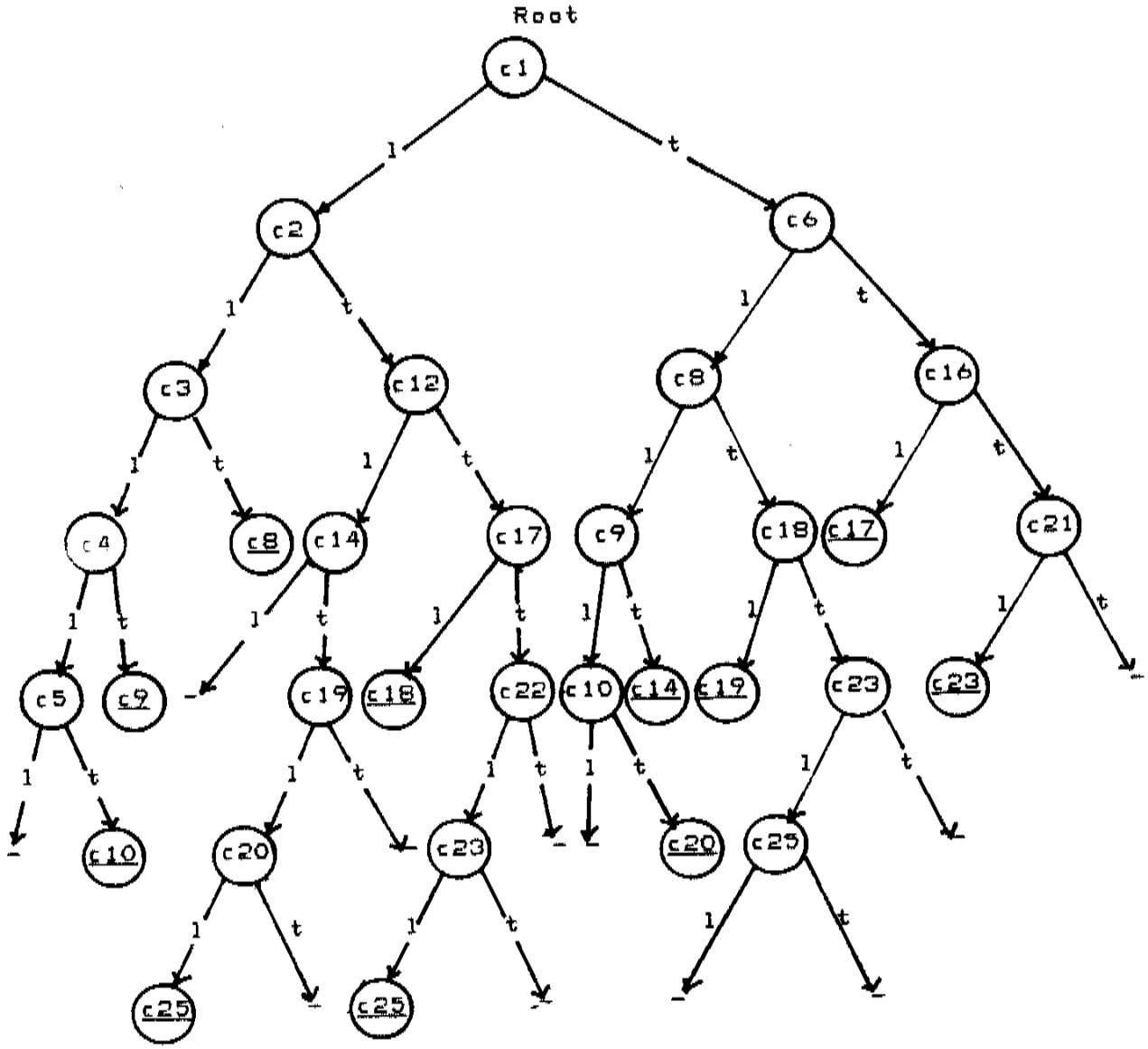

Figure 7: The connection tree, C-tree(c1), relative to the "left-of" and "top-of" relation names.

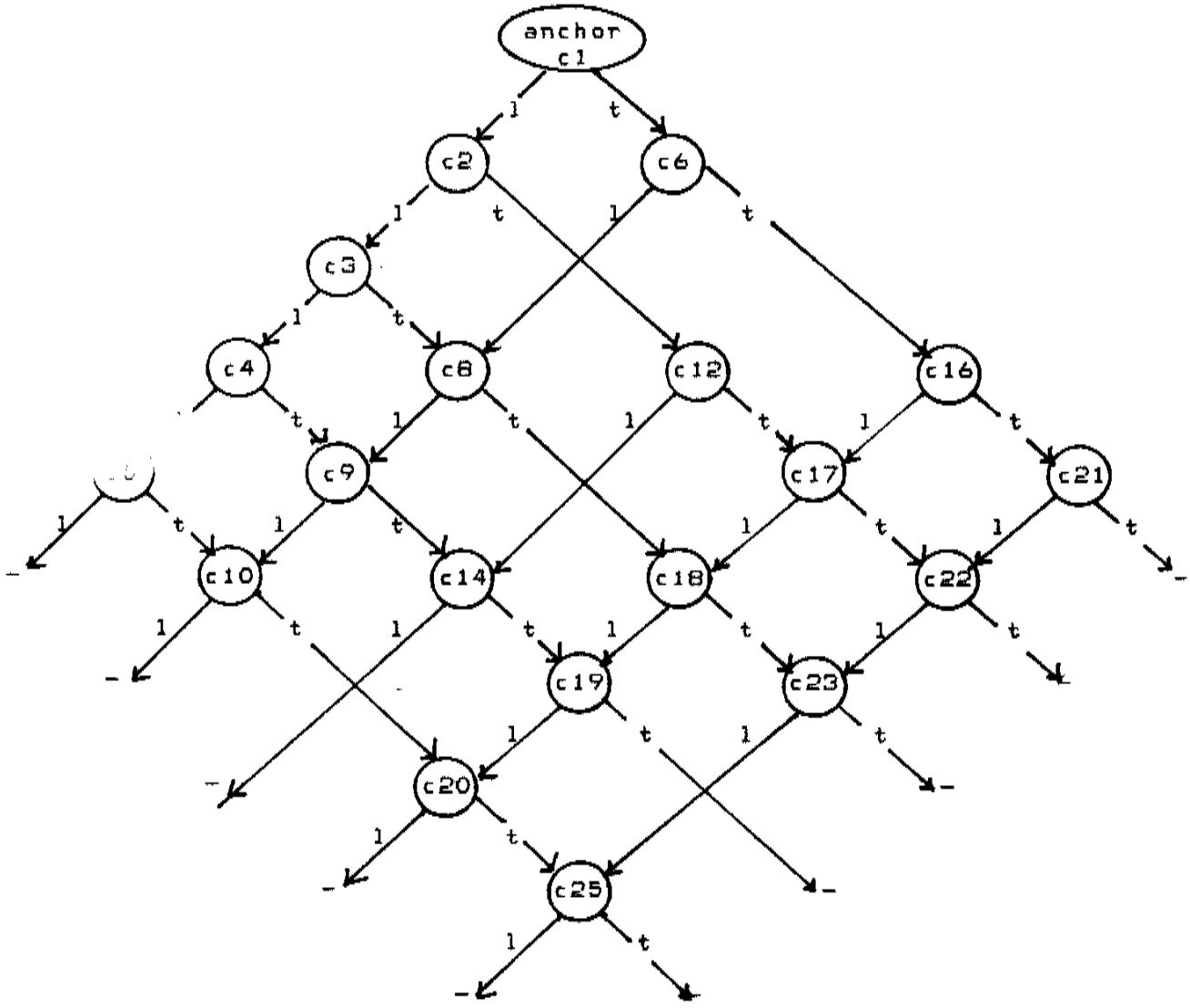

Figure 8: Connection graph, C-graph(c1).

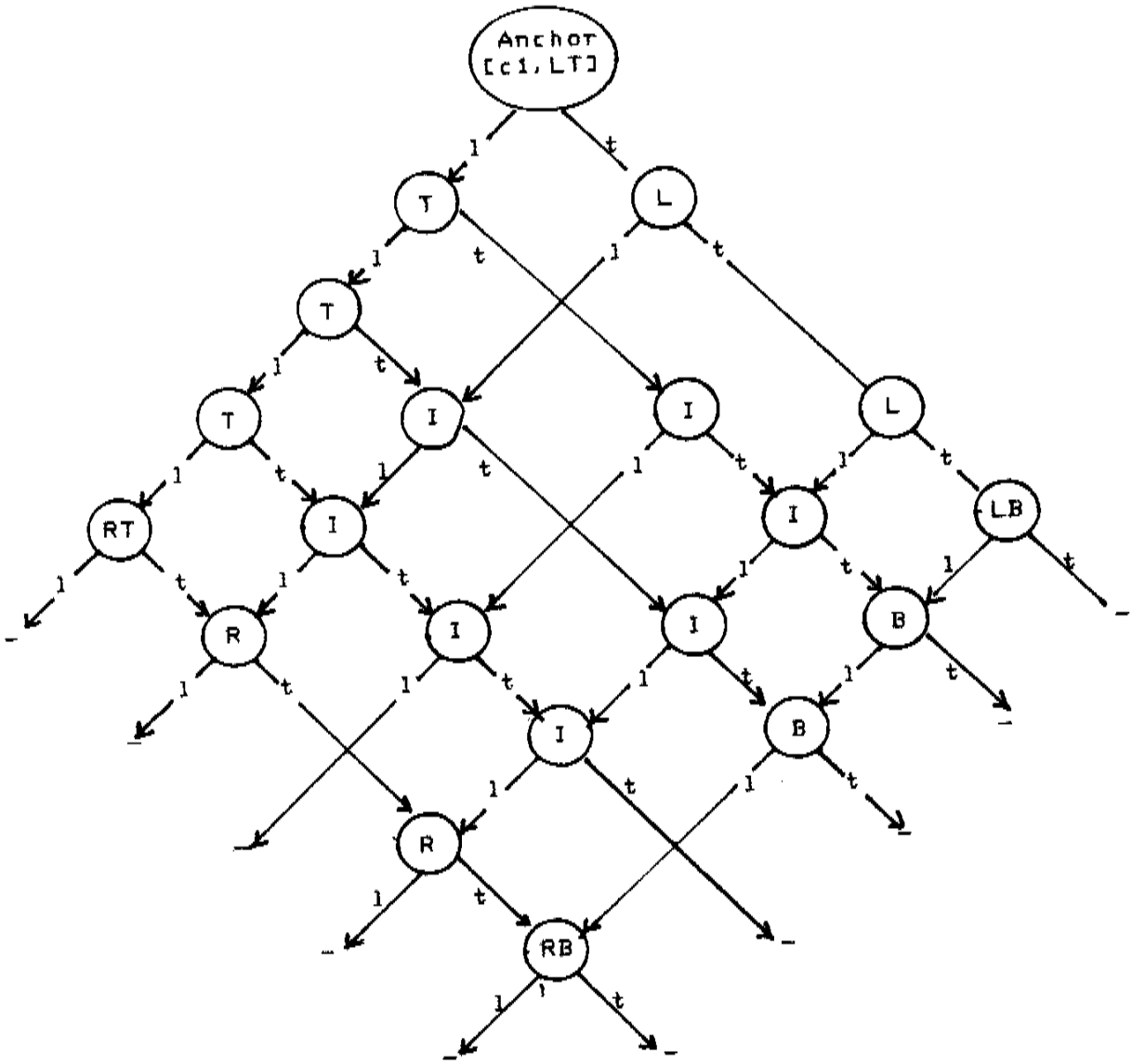

Figure 9: Dependency Graph, D-graph(c1 LT).