Interactive Classification: A Technique for Acquiring and Maintaining Knowledge Bases

TIMOTHY W. FININ, MEMBER, IEEE

The practical application of knowledge-based systems, such as in expert systems, often requires the maintenance of large amounts of declarative knowledge. As a knowledge base (KB) grows in size and complexity, it becomes more difficult to maintain and extend. Even someone who is familiar with the knowledge domain, how it is represented in the KB, and the actual contents of the current KB may have severe difficulties in updating it. Even if the difficulties can be tolerated, there is a very real danger that inconsistencies and errors may be introduced into the KB through the modification. This paper describes an approach to this problem based on a tool called an ***interactive classifier****. An interactive classifier uses the contents of the existing KB and knowledge about its representation to help the maintainer describe new KB objects. The interactive classifier will identify the appropriate taxonomic location for the newly described object and add it to the KB. The new object is allowed to be a generalization of existing KB objects, enabling the system to learn more about existing objects.*

I. Introduction

The practical application of knowledge-based systems, such as in expert systems, requires the maintenance of large amounts of declarative knowledge. As a knowledge base (KB) grows in size and complexity, it becomes more difficult to maintain and extend. Even someone who is familiar with the domain, how it is being represented, and the current KB contents may introduce inconsistencies and errors whenever an addition or modification is made.

One approach to this maintenance problem is to provide a special **KB editor**. Schoen and Smith, for example, describe a display-oriented editor for the representation language STROBE [28]. Freeman *et al.*, have implemented an editor/browser for the KNET language [14], [15]. Lipkis and Stallard are developing an editor for the KL-ONE representation language (personal communication). There are several problems inherent in the editor paradigm, for example:

- The system must take care that constraints in the KB, such as those defined via subsumption, are maintained.
- The system must distinguish at least two different kinds of reference to a KB object: reference *by name* and reference *by meaning*. A reference *by name* to an object should not be effected if the underlying definition of the object is changed by the editor. If one refers to an object *by meaning*, however, and later edits the object referred to, then the reference should still refer to the original description.
- The system must keep track of the origin of the subsumption relationship to distinguish between those explicitly sanctioned by the KB designer and those inferred by the system (e.g., by a classifier).
- Editors tend to be complex formal systems requiring familiarity with the editor and with the structure and content of the KB being modified.

This paper describes another approach to the KB maintenance problem based on a tool called an interactive classifier. This kind of tool is not as general or powerful as a full KB editor but avoids many of the problems described above. The interactive classifier can only be used to make monotonic changes to the KB. New objects can be added to the taxonomy and additional attributes can be added to objects already in the KB. It does not allow objects to be deleted or their existing attributes changed or overridden.

Although this may sound like a severe restriction, we believe that there are many situations where this is just the kind of KB update that is to be allowed. Consider, for example, the *computer configuration* problem which has been the domain of several recent expert system projects [21], [14], [23]. Such a system needs to have an extensive KB describing a large number of computer components and their attributes, including their decomposition and interconnection constraints. An important feature of this domain is that new components are constantly being introduced as the underlying technology advances. Older components still need to be represented in the KB since there are many installations in the field which may still need them. We may, however, want to predicate additional attributes of these older components to distinguish them from newer ones. For example, at some point in time we may add a new *laser printer* to the line of hardcopy devices. At a later time, we may want to add a new model, a *high-speed laser printer*. This might involve adding two new objects: one to represent a generic laser printer with an attribute *printing speed* and another to represent the new high-speed laser printer. The original *laser printer* object would be seen as a specialization of the newly created generic laser printer.

Knowledge-based systems often represent declarative knowledge using a set of nodes, corresponding to discrete

Manuscript received June 20, 1985; revised April 6, 1986.
The author is with the Department of Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389, USA.

0018-9219/86/1000-1414$01.00 © 1986 IEEE

"concepts" or descriptions, which are partially ordered by a subsumption, or inheritance relation. One concept subsumes another if everything that is true about the first is also true about the second. Whenever a new node is added to the knowledge base, either during its initial construction or later maintenance, it must be placed in the appropriate position within the ordering—i.e., all subsumption relationships between the new node and existing nodes must be established. This is called *classification* because a subsuming node can be considered as a representation of a more abstract category than its subsumees. The notions of subsumption and automatic classification are very useful and have been offered as central features of several recent knowledge representation languages (see [5], [26], [9], and [2], for example).

Current classifiers require a complete description of the node to be added before they begin. (See [32] for a description of classification, and [20] or [27] for examples of a classifier for the representation language KL-ONE [4].) When the classifier is used directly by a user to add a new node, the user must know the descriptive terms in use in the existing KB and something of its structure in order to create a description which will be accurately classified. If the classifier places the new node in the wrong place, or if the description of the node contains errors or omissions, the user must repeatedly modify the node and redo classification until he is satisfied. The process of adding a node is much more efficient if done interactively, so that immediate feedback based on the contents of the KB is available to the user as each piece of information about the new node is entered.

The rest of this paper describes an interactive classification algorithm, which has been implemented in Prolog. Together with a simple knowledge representation language, this implementation forms a system called KuBIC, for Knowledge Base Interactive Classifier. The system takes a user's initial description of a new node and a (possibly empty) KB and either classifies the node immediately, if enough information has been specified, or determines relevant questions for the user that will help classify it. Thus a user who is familiar with the knowledge base may completely avoid the question/answer interaction with KuBIC, and use it only as a classifier, while someone who has never seen the knowledge base before may use the interaction to be presented with just those portions of the KB which are relevant to the classification of the new concept. The algorithm could be applied, for example, to knowledge representation systems or environments for building expert systems which contain classifiable knowledge bases, such as KEE [18], HPRL [19], SRL [13], or LOOPS [3].

II. The Representation Language

In order to explore the underlying ideas of interactive classification, a simple knowledge representation language was chosen. The KB is constrained to be a tree structure, so each node has at most one parent. Nodes have single-valued attributes which represent components or characteristics that apply to the object or concept described. Values of attributes can be numbers, intervals, symbols, or sets of symbols. The meaning of a set or range with multiple values is *disjunctive*; children of a node with an attribute with multiple values can have any subset or subrange (including single values) of the parent's value. Each node inherits all the attributes of its parent node, but its values can be restrictions of the parent attribute's values. Finally, no procedural attachment is allowed.

The Subsumption Relation: The tree structure of the knowledge base is formed by the partial ordering of its nodes with respect to the subsumption relation. The intended meaning of "*X subsumes Y*" is that whatever is represented by description *Y*, is also represented by the more general description *X*. All of *X*'s characteristics are inherited by *Y*, perhaps with some restriction. Since the subsumption relation is transitive, *Y* also inherits the characteristics of *X*'s subsumers (i.e., all its ancestors in the tree). In KuBIC, subsumption information is used to achieve *economy of description* and to *localize distinguishing information*.

Economy of description is a direct consequence of the inheritance of attributes and attribute values. Each description is considered to be a *virtual* description whose attributes are either local to the real description, or inherited from an ancestor. Only the most restricted value of an attribute appears in the attribute of the virtual description, even if the value occurs in an attribute of more than one ancestor description.

Classification is aided by the structure of the knowledge base. In such a taxonomic database, distinguishing information is localized. Once a new description has been determined to be subsumed by node *X*, only *X*'s subsumees are possible candidates for a more specific subsumer of the new description. The information stored at *X*'s immediate subsumees allows the classifier to select questions which will determine which node is this more specific subsumer.

III. Interactive Classification

The interactive classification process is divided into three phases: acquiring the initial description of the new concept, finding the appropriate parent concept in the existing taxonomy (the most specific subsumer), and finding the appropriate immediate descendants in the existing taxonomy (the most general subsumees).

A. Acquiring the Initial Description

To make the interaction more efficient and minimize the number of questions the user has to answer, the user is allowed to specify an initial description of the new node. Attributes of the new node can be given, and a subsumer can be stated directly if known. Note that the user can say only that a node subsumes the new node, not that it is the **most specific** subsumer. If enough information is given, it is possible to classify the new node immediately without any further interaction. If not, KuBIC must determine what attributes to ask about so that classification can be completed.

If the initial description includes an attribute which is not currently in the KB, then the user is asked to supply certain information about the new attribute. In the simplified representation language used in KuBIC, this information is just the general constraint on possible values that the attribute can take on and a *question form* that the system can use to ask for a value for this attribute.

B. Establishing the Most Specific Subsumer

Because the characteristics of a node are shared by all its descendants, it is most efficient to search the tree for the new node's most specific subsumer (MSS) in a top-down

manner, starting with the root. Two strategies are used to speed the search for the most specific subsumer: classification by *attribute profile* and classification by *exclusion*. The first strategy is used to take the partial description of the new KB object the user initially presents and to identify a likely ancestor as low in the taxonomy as possible. The second strategy, classification by exclusion, is used to *push* the new KB concept lower in the taxonomy, eliciting new information from the user as needed. This second strategy is more basic to the interactive classifier and will be described in detail first.

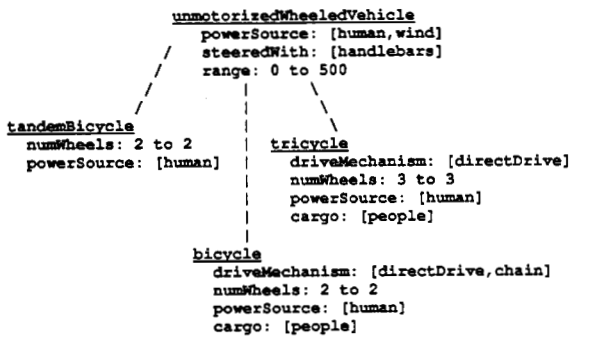
Classifying Using Exclusion: Classifying by exclusion makes use of the fact that every node (except the root) has exactly one immediate subsumer, or parent. At all times during classification, there is one node which has been verified to be a subsumer of the new node, and is the most specific such node (the current most specific subsumer, or MSS). Only subsumees of this node need be considered as more-specific subsumers. Moreover, at most one of the immediate subsumees of this node may be a more-specific subsumer.

Exclusion therefore proceeds by looking for inconsistencies between the current description of the new node and the immediate subsumees of the current MSS. If no subsumees are consistent, the current MSS remains the actual MSS, and classification continues with the search for the new node's subsumees. If only one node is consistent, it must be verified to be a subsumer of the new node. This is done by asking the user, if necessary. If two or more nodes are consistent, attributes must be found to ask the user about which will help exclude as many of them as possible, until less than two nodes remain consistent.

The word "consistent" is a bit inadequate—what is actually meant by "*node S is consistent with the new node N*" is "*node S subsumes the* ***current*** *description of the new node.*" (Note that this consistency relation is not symmetric.) Because the new node's description changes during its interactive classification as the user adds new information, it is possible for *S* to become inconsistent with it. Thus the meaning of the term *consistent* that we are using is similar to that used in discussions of nonmonotonic logic [22].

Verifying Subsumption of a Consistent Node: Because the new description is entered interactively, one attribute at a time, it is incomplete during classification. Suppose that there is only one candidate node in the set of consistent children of the current MSS. This is not enough to ensure that the candidate is a more specific subsumer of the new node since the candidate may have additional attributes that the new node does not have. We are assuming, of course, that the user is giving us a *partial description* of the new KB object. If the candidate has additional attributes, we must verify that it indeed subsumes the new node. For each such attribute, the user is presented with its value in the candidate node and is asked to confirm, deny, or restrict the value as appropriate for the new node (see Fig. 2). This is done to ensure that the values of the new node's attributes are restrictions of the values of the candidate's values. Note that the new node may have attributes which the candidate does not have; this does not affect subsumption. If the user does **not** verify that the single candidate node is a subsumer of the new node, then the current MSS of the new node is established as the final MSS.

An example showing a KB fragment which requires such verification is shown in Fig. 1, and the verification interaction is shown in Fig. 2. For each node in Fig. 1, only the attributes which are either defined locally or locally restricted are displayed at that node. The new description,



Dashed lines mean "subsumes", with subsumer above subsumee.

Fig. 1. KB fragment just before verification.

There is evidence that the new description is a *bicycle*. I will now question you on each unverified aspect of *bicycle*. Please confirm, deny, or restrict the value, for each attribute.

What is the cargo?
cargo = [people]
Enter yes, no, or a restriction of the answer: ***yes.***

What is the drive mechanism?
driveMechanism = [directDrive,chain]
Enter yes, no, or a restriction of the answer: ***[chain].***

I've verified that *bicycle* subsumes *tandemBicycle*

Is this acceptable?: ***yes.***

Subsumer changed from *unmotorizedWheeledVehicle* to *bicycle*.

Fig. 2. Interaction during verification (user's response in ***bold italics***).

tandemBicycle, has been determined to be subsumed by *unmotorizedWheeledVehicle*. Since one attribute of *tandemBicycle* is that it has two wheels, only *bicycle* is a consistent candidate for a more specific subsumer of *tandemBicycle*. Before asserting that *tandemBicycle* is subsumed by *bicycle*, the user is asked to verify that *tandemBicycle* also has a *cargo* attribute whose value is *people* and has a *driveMechanism* whose value is a subset of {*directDrive*, *chain*}. If the user does not agree, *tandemBicycle*'s MSS remains *unmotorizedWheeledVehicle*.

Determining the Next Question: If there are two or more candidate nodes in the set of consistent children of the current MSS, more information about the new node is required to exclude some of them. This is done by selecting an attribute to ask about, getting the answer from the user, and repeating until the set of consistent children has been reduced to zero or one node, or there are no more attributes which will help reduce the set. Two strategies are used to select an attribute to ask about from the set of attributes which apply to the set of consistent children: *explicit attribute ranking* and *maximal restriction*.

In our simple representation language, one can attach to a concept a list of some of the concept's attributes which are ranked with respect to their importance in classifying by exclusion. If such a ranking has been defined, then the

attributes are selected in the given order. This strategy supercedes the next one, because the ranking contains external information which is not otherwise available to the system. The ranking could be based on numerical weights, but here it is a non-numerical ordering.

If there are no more attributes in the ranked list, the attribute selected to ask about is the one which *maximally restricts* the set of consistent children, in the worst case. In other words, no matter what answer is given as the value of this attribute, the minimum number of consistent children which are excluded by the answer is greater than or equal to the same minimum for any other relevant attribute. If more than one attribute is best, one is selected without regard to other considerations.

The above strategies could be augmented by using information about the particular user. Since not all questions need to be asked to perform one classification, questions which the user is more likely to be able to answer should be asked first. The user's ability to answer can be decomposed into his or her ability to understand the question, determine an appropriate response, and communicate the response to the system. The user model could be created initially by asking the user several questions intended to establish a stereotype of the user, and refined later as the user answers (or does not answer) questions. (See [25] and [10] for examples of this use of stereotypes.)

Classifying Using Attribute Profiles: The second classification strategy is a heuristic for searching the tree more quickly. Given the operations of determining consistency and asking the user to verify subsumption described above, if a guess could be made about possible subsumers of the new node, it would be a simple matter to verify the subsumption. A good guess is necessary, however, because the user must get involved in the verification.

The particular heuristic used in KuBIC examines the set of attributes specified by the user in the initial description to try to restrict the possible subsumers of the new node. The heuristic could also be used whenever volunteered information is allowed. It works by picking an attribute of the initial description, finding the common ancestor of all nodes in the KB which have the attribute, and using this common ancestor as a guess. The guess must be a subsumee (immediate or not) of the current MSS of the new node.

If the user verifies the guess, then it becomes the current MSS and the process continues. The user has been spared from having to answer questions about attributes of concepts which lie between the original MSS and the guess. The deeper the guess in the tree, the more questions avoided. If the user does not verify the guess, perhaps because the attribute has more than one meaning in the current KB, all is not wasted. Questions asked during verification can contribute information to the new node, or, if the attribute in question is not an attribute of the new node, KuBIC knows not to ask the question again. The system can keep guessing, whether a previous guess succeeded or not, until it runs out of attributes, or until the user becomes weary of incorrect guesses.

C. Establishing the Most General Subsumees

The task of classification is half completed once the most specific subsumer of the new node has been established. Finding the most general subsumees (MGSs) is the other half. Fortunately, this half is much less work because of the constraint that the KB form a tree structure.

The only possible candidates for most general subsumees are children of the MSS of the new node—i.e., siblings of the new node. (This assumes that the KB is well-constructed, so that the immediate subsumer of each node is its MSS, and then immediate subsumees are its MGSs.) Thus to find all the MGSs, it is only necessary to check whether the new node is "consistent" with each sibling in turn, and to ask the user to verify that there is no missing information about either node which misled the classifier. Note that by establishing a node as the MGS of the new node, the interactive classifier can implicitly change the descriptions of the MGS and all its subsumees—nodes which were already in the KB—because they inherit new attributes from the new node.

If the subsumption relationship is allowed to define a lattice rather than a tree, then determining the MGSs is more difficult. A newly entered node may not subsume its siblings, but could subsume some of its sibling's descendants. For example, consider a taxonomy for living things which includes a concept *livingThing* with two immediate children: *animal* and *plant*. We could use the interactive classifier to enter a new node *genderedLivingThing* to represent the concept of a *livingThing* with an attribute gender whose values come from the set {male,female}. This concept would initially be an immediate descendant of the concept *livingThing*. Neither *animal* nor *plant*, however, is a descendant of this new concept, since there are genderless animals and genderless plants. Many of their descendants, however, are subsumed by the new concept *genderedLivingThing*.

IV. Interactive Classification in More Expressive Languages

The KuBIC system is limited by the extremely simple nature of the knowledge representation language we have used. Using this simplified language was a conscious research strategy choice. It allowed us to focus on the notion of an interactive classifier in a simple surrounding. The two major shortcomings in KuBIC's representation language are that nodes are organized in a tree rather than a lattice, and that values of attributes must be explicit sets of values or value intervals in the case of totally ordered domains such as integers. Neither of these limitations is a serious obstacle to extending the idea of interactive classification to more general representation languages. This section briefly describes some additional work on two experimental interactive classifiers, CHPRL and KLASSIC, as well as a proposal for an extension to a KL-ONE-like language to better support interactive classification.

A. The CHPRL Classifier

Our first experiment was to build an interactive classifier for the frame-based representation language HPRL [19].[1] HPRL is a frame-based representation language which relaxes the restrictions found in the simple KuBIC language.

[1]HPRL was developed at the Hewlett Packard Laboratories. The version used in this research ran in Portable Standard Lisp on an HP9836 workstation

In particular, in HPRL concepts can have any number of immediate ancestors and attributes have a much richer structure. A concept in HPRL is called a **frame** and consists of a name, a set of immediate ancestors, and a set of **slots** which correspond to our notion of attributes. In HPRL, a slot has a number of pieces, or facets, of information attached to it. The facets important to classification include a restriction on the type of values a slot can take on, a restriction of the number of values it can have, a description of a default value for the slot, and (possibly) a set of actual values for the slot. We extended the HPRL language slightly to allow the type information to be expressed in one of two ways—either as a procedure (as was normally the case) or as a reference to another HPRL frame.

The CHPRL classifier [24] takes a newly defined HPRL frame and interactively classifies it. One interesting aspect of this system is its ability to handle *exceptions* and *default values*. An *exception* is a specification of a value or a restriction of a value for a slot which is inconsistent with an inherited value or value restriction. A *default value* is an annotation on a slot which specifies a value to be used for the slot if one is needed and there are no local or inherited values for the slot. We have addressed the problems of classification of descriptions involving exceptions and defaults because they are an integral part of the HPRL language, unlike languages in the KL-ONE family. The CHPRL classifier handles defaults by treating them as "virtual values." Thus a slot S_1 with a set of default values D can be subsumed by a slot S_2 only if S_2 has no values, or has a value restriction which subsumes all the defaults in D and either has no defaults of its own or has a set of defaults which is a superset of D.

It is generally held that exceptions introduce many serious problems in a knowledge representation system, particularly with those that include automatic classifiers [6]. It is also widely believed that the general notion of an exception is a useful one, especially in representational systems which attempt to model people's representations. Our attempt to combine the notion of automatic classification and exceptions in CHPRL is based on the following ideas:

- **The classifier never introduces an exception.** All exceptions must by explicitly sanctioned by the user. The classifier is not allowed to hypothesize an exception to enable one concept to be subsumed by another. If a user asserts a subsumption relation between two concepts, any inconsistencies that are detected are marked as exceptions.
- **Exceptions are efficiently indexed.** All exceptional facts are linked to the general facts that they violate. This allows the classifier to efficiently compare the concept being classified to any "exceptional" concepts when appropriate, without searching the entire knowledge base.
- **Exceptions are exceptional.** It is assumed that the number of exceptions in a knowledge base is "small" relative to the size of the knowledge base. If this is not the case, then it is likely that the KB needs to be redesigned. This assumption ensures that the classification process will not bog down in exception checking.

B. The KLASSIC Representation Language

Our second experiment involved building a new representation language with an integral interactive classifier "on top" of HPRL. This language, KLASSIC [17], is very similar to KL-ONE in that it has a more formally defined semantics. One frame $C1$ subsumes another $C2$ if everything which is true of $C1$ is necessarily true of $C2$. In addition, KLASSIC implements some constraints expressed through *role value maps* [7]. Our approach to building KLASSIC was to define the significant components, or units, out of which a description is built (e.g., concept, role, and role value map) as frames in the underlying HPRL representation language. This made it very easy to modify and extend the KLASSIC language. All three classes of units are organized into abstraction hierarchies as well. Thus we can represent the fact that the role **address** specifies a relation between people and places and has two immediate subsumees, **homeAddress** and **officeAddress** as well as one immediate subsumer, **location**.

In building an interactive classifier for KLASSIC, we had to address the additional problems of *primitive concepts* and *role subsumption*. A *primitive concept* is a concept which is only partially defined in the KB. Typically, a primitive concept has some necessary attributes specified but lacks a specification of all of the sufficient attributes. Realistic KBs typically contain many primitive concepts (see, for example, [16]). When the KLASSIC interactive classifier is trying to determine whether or not a new concept C_{new} is subsumed by a primitive concept C_p, it first tries to verify that C_{new} is consistent with C_p by showing that all of C_{new}'s attributes are true of C_p as well. If this is the case, then KLASSIC asks the user to verify that C_{new} is indeed subsumed by C_p.

There are many primitive concepts, however, that should never be considered as potential subsumers of any new user-defined concept. For example, many applications require an object to represent the concept of an integer. Such an object typically functions as a "name" for the concept and is not elaborated in any way (i.e., has no attributes or constraints). It is not anticipated that the user will want to introduce new descriptions which are to be classified as specializations. To prevent the classifier from pestering the user with questions about such basic concepts, we have introduced a new type of primitive concept, a *primordial* concept. A primordial concept is one whose descendants are fixed.[2] No user is allowed to introduce new descendants through classification. The interactive classifier will never consider a primordial concept as a potential subsumer of a newly entered concept unless it is explicitly mentioned in the description.

In KLASSIC, as in most languages in the KL-ONE family, a concept's roles are organized into an abstraction hierarchy just as the concepts themselves are. Since a description of a new concept to be added to the KB is an expression containing both concepts and roles, the classifier must be

[2]More specifically, the descendants are fixed with respect to the classifier. There may be other ways for new descendants of a primordial concept to be introduced, such as the syntactic recognition of individual integers.

able to compute role subsumption as well as object subsumption. The basic idea is that, given a new description, its roles must be classified before classifying the concept itself.

C. Classification and Primitive Concepts

The basic idea behind classification, whether interactive or not, is that given any two *concept definitions* it is possible to determine if one subsumes the other. However, it is often the case that many concepts we would like to represent do not seem to have precise definitions. To represent such concepts in a KL-ONE knowledge base requires that they be specified as **primitive concepts** [7]. Primitive concepts may have some information specified for them, but they do not have complete definitions.

Primitive concepts hinder classification since the user must explicitly specify the relationship of new concepts to any primitive concepts in the knowledge base. In real applications, the number of primitive concepts may comprise over half of the concepts in the knowledge base [16]. A user wishing to enter a concept must manually classify the concept with respect to all known primitive concepts to ensure the concept is placed correctly in the knowledge base. For large knowledge bases this can be both difficult and error-prone.

We are exploring an extension to languages like KL-ONE which reduces the burden on the user when adding new concepts to a knowledge base while maintaining the soundness of the knowledge representation language. This extension consists of adding an explicit definitional component to concepts in the knowledge base. Within this component the strictness of concept definitions is itself relaxed. The benefits of this modification are threefold:

- The relaxed form of definitions will reduce the number of primitive concepts in a knowledge base.
- The explicit definitional component can be used by the classifier with concepts that do not have complete definitions.
- The definitional component improves the utility of an interactive classification.

Providing an Explicit Definitional Component: We are designing a representation language which can be seen as an extension to a KL-ONE-like language which permits an explicit **definitional** component for each concept. This definitional component has the form

$$def(X) = N_1 \wedge N_2 \wedge \ldots \wedge N_k \wedge D_1 \wedge D_2 \wedge \ldots \wedge D_l.$$

The N_i are necessary conditions for something being an X. The D_i-terms[3] represent disjunctions of sets of contingent conditions (i.e., non-necessary conditions) and have the form

$$D_i = S_{i1} \vee S_{i2} \vee \ldots \vee S_{in}.$$

The S-terms consist of conjunctions of contingent conditions or C-terms. An N-term or C-term may be either a simple term or a reference to another concept's definition.[4] A simple term is akin to a role and its value restriction in KL-ONE. Our notation also allows for a negated simple term, whether it is a necessary or contingent attribute. Finally, a D-term can be a covering disjunction for the concept it defines. Additional details are given in [16].

Consider the following hypothetical example of the definition of a concept X:

$$def(X) = N_1 \wedge N_2 \wedge N_3 \wedge ((C_1 \wedge C_2) \vee (\neg C_1 \wedge C_3)).$$

In this case N_1, N_2, and N_3 are necessary conditions for X. In addition to these conditions C_1, C_2, and C_3 play a role in the definition of X, but are not in themselves necessary. In fact (together with N_1, N_2, and N_3), the clauses $(C_1 \wedge C_2)$ and $(\neg C_1 \wedge C_3)$ form two sets of sufficient conditions for being an X.

Ramifications for Interactive Classification: The proposed extension makes it much easier for the classifier to determine subsumption on its own. Assuming the creators of the knowledge base take full advantage of the extended definitional capability for concepts, the classifier should be able, in many cases, to find a sufficiency set which fits the new description. Even if a perfect fit cannot be found, the classifier can look for the best matching set.

In the case of primitive concepts, the gain is even greater. Whereas the interactive classifier previously had to check *every* primitive concept with the user, it can now autonomously decide about subsumption in many of the cases. The user will be called upon only in cases where the new description could be a *new exception*. If the user sanctions the exception, it will be reflected as a change to the contingent features of the definition of the existing subsuming concept.

V. Providing a Model of the User

If an interactive classifier is to live up to its promise as a tool for building and maintaining large and complex knowledge bases, it must be good at interacting with its users. We are currently working on the incorporation of a more sophisticated model of the user to support this interaction. Such a model can be used to select attributes to ask about next and also to provide the user with appropriate help and guidance in answering questions. This is related to work in the context of interfaces to expert systems (see [30], [31], [11], for example).

There has been some previous research on how expert systems get information from their users. For example, Fox [12] considered integrating reasoning with knowledge acquisition from a resource management perspective. Aikins [1] addressed the seemingly random question-asking behavior of systems which pursued lines of reasoning opportunistically, jumping around to whatever line looked

[3]Note that a definition may have several D-terms, each representing a range of possible attributes. For example:

def(employee)=
preson∧. . .∧
(EmplStatus-FullTime∨EmplStatus-PartTime)∧
(pay-salaried∨pay-hourly∨pay-commissioned).

[4]Referencing another definition is simply a matter of convenience. The fully expanded form could be substituted for the reference.

most promising and asking for whatever information they needed at that point. This randomness annoyed and confused users. Aikins suggested an organization for reasoning that would result in related questions being asked together. Brooks [8] considered the amount of information systems may end up requesting from their users and found that a large number (30 or more) of requests is generally considered unacceptable. He suggested ways of cutting down on the amount of information requested, by enriching systems' models of their domains. These same sorts of considerations can be employed in the context of interactive classification.

For an interactive classifier, the system's goal should be to classify a new concept while burdening the user as little as possible. There may be several outcomes to the system asking a question of the user:

- The user may be unable or unwilling to answer the question.
- The user may need to invoke a subdialogue with the system in order to get additional information to enable him to answer the question. This additional information may be provided in the form of definitions of terms, question paraphrases, or other kinds of help.
- The user may provide an uncertain answer. In general, we might assume that any answer can be qualified or hedged by associating a *degree of belief* with it.

Thus we can define the *effectiveness* of a query as a function of the amount of information returned in the answer and the amount of "user interaction" required. A sensible heuristic for choosing the next attribute to ask about is to choose the most effective query.

Our estimate of the amount of interaction required to get an answer from the user and of the certainty of the answer we will obtain will have to depend on our model of the user. We have developed some general domain-independent tools for building user models [10] that we will use for this purpose.

VI. Summary

This paper presented the design and implementation of an interactive, incremental classifier which is used to add nodes to a hierarchical frame-oriented knowledge base. A knowledge representation language was defined, complex enough to resemble in certain aspects representations of current knowledge-based systems, yet simple enough to allow focusing on interactive classification (for more detail and the Prolog implementation of KuBIC, see [29]). The problem of classification was described as determining most specific and most general subsumption relationships between the new node and nodes already in the knowledge base. Two components to the classification strategy were presented. Classification using exclusion uses a special "consistency" relation and asks questions to exclude whole portions of the KB at a time. Classification using attributes uses a heuristic based on what attributes the user says the new node has in order to take shortcuts in the search. Both of these serve to establish the most specific subsumer; the most general subsumees are then relatively simple to find.

We have built several additional experimental interactive classifiers in representation languages of differing points of view. In one, CHPRL, we looked at some of the issues involved in a language with a system of defaults and in which exceptions are allowed. In another, KLASSIC, we examined the problems of interactive classification in a language in which the abstraction hierarchy forms a lattice rather than a tree and includes definitions of primitive concepts. Current work is focused on extending the concept of an interactive classifier to a more powerful representation language that includes explicit mechanisms for specifying definitions and incorporating a more sophisticated user model.

Bibliography

[1] J. Aikins, "Prototypes and production rules: A knowledge representation for computer consultations," Heuristic Programming Project HPP-80-17, Stanford University, Stanford, CA, Aug. 1980.

[2] H. Ait-Kaci, "Type subsumption as a model of computation," in *Expert Database Systems*, L. Kerschberg, Ed. Menlo Park, CA: Benjamin/Cummings Publ., 1985.

[3] D. G. Bobrow and M. Stefik, "The loops manual," Xerox PARC Tech. Rep. KB-VLSI-81-13, 1981.

[4] R. Brachman, "A structural paradigm for representing knowledge," Bolt Beranek and Newman Inc., Tech. Rep. 3605, May, 1978.

[5] R. Brachman, R. Fikes, and H. Levesque, "KRYPTON: A functional approach to knowledge representation," Tech. Rep. 16, Fairchild Laboratory for AI Research, 1983.

[6] R. Brachman, "I lied about the trees," *AI Mag.*, vol. 6, p. 3, 1985.

[7] R. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cogn. Sci.*, vol. 9, pp. 171–216, 1985.

[8] R. Brooks and J. Heiser, "Controlling question asking in a medical expert system," in *Proc. Int. Joint Conf. on Artificial Intelligence* (Tokyo, Japan, 1979), pp. 102–104.

[9] F. Corella, "Semantic retrieval and levels of abstraction," in *Expert Database Systems*, L. Kerschberg, Ed. Menlo Park, CA: Benjamin/Cummings Publ., 1985.

[10] T. Finin and D. Drager, "GUMS$_1$: A general user modeling system," in *Proc. 1986 Conf. of the Canadian Society for Computational Studies of Intelligence* (Montreal, Que., Canada, CSCSI, May 1986).

[11] T. Finin, A. Joshi, and B. Webber, "Natural language interactions with artificial experts," *Proc. IEEE*, vol. 74, no. 7, pp. 921–938, July 1986.

[12] M. S. Fox, "Reasoning with incomplete knowledge in a resource limited environment: Integrating reasoning with knowledge acquisition," in *Proc. 7th Int. Joint Conf. on Artificial Intelligence* (University of British Columbia, Vancouver, B.C., Canada, Aug. 1981).

[13] M. S. Fox, J. Wright, and D. Adam, "Experiences with SRL: An analysis of a frame-based knowledge representation," in L. Kerschberg, Eds., *Expert Database Systems*. Menlo Park, CA: Benjamin/Cummings Publ., 1985.

[14] M. Freeman, L. Hirschman, and D. McKay, "A logic based configurator," SDC, A Burroughs Co., Tech. Memo LBS 9, May 1983.

[15] ——, "KNET–A logic based associative network framework for expert systems," SDC, A. Burroughs Co., Tech. Memo LBS 12, Sept. 1983.

[16] R. Kass, R. Katriel, and T. Finin, "Breaking the primitive concept barrier," CIS, Univ. of Pennsylvania, Tech. Rep. MS-CIS-86-36 (LINC LAB 11), May 1986.

[17] R. Katriel, "KLASSIC," unpublished report.

[18] T. P. Kahler and G. D. Clemenson, "An application development system for expert systems," *Systems & Software*, Jan. 1984.

[19] D. Lanam, R. Letsinger, S. Rosenberg, P. Huyun, and M. Lemon, "Guide to heuristic programming and representation language. Part 1: Frames," Application and Technology

Lab., Computer Res. Ctr., Hewlett-Packard Corp., AT-MEMO-83-3, Jan. 1984.
[20] T. A. Lipkis, "A KL-ONE clasifier," USC/Inform. Sci. Inst., Consul Note 5, Oct. 1981.
[21] J. McDermott, "R1: A rule-based configurer of computer systems," Carnegie-Mellon Univ., Pittsburgh, PA, 1980.
[22] D. McDermott and J. Doyle, "Non-monotonic logic I," *Artificial Intell.*, vol. 13, pp. 1–2, 1980.
[23] J. McDermott, "XSEL: A computer salesperson's assistant," in *Machine Intelligence 10.* Chichester, UK: Horwood, Ltd., 1982, pp. 325–337.
[24] R. Petterson, "The CHPRL classifier for HPRL," unpublished report, 1984.
[25] E. Rich, "User modeling via stereotypes," *Cogn. Sci.*, vol. 3, pp. 329–354, 1979.
[26] J. G. Schmolze and D. Israel, "KL-ONE: Semantics and classification," Bolt Beranek and Newman Inc., Cambridge, MA, Rep. 5421, 1983.
[27] J. G. Schmolze and T. A. Lipkis, "Classification in the KL-ONE knowledge representation system," in *Proc. Int. Joint Conf. on Artificial Intelligence* (Karlsruhe, West Germany, 1983).
[28] E. Schoen and R. Smith, "IMPULSE: A display oriented editor for STROBE," in *Proc. Nat. Conf. on Artificial Intelligence* (AAAI, Washington, DC, Aug. 1983), pp. 356–358.
[29] D. L. Silverman, "An interactive, incremental classifier," Univ. of Pennsylvania, Philadelphia, Tech. Rep. MS-CIS-84-10, Apr. 1984.
[30] B. Webber and T. Finin, "In response: Next steps in natural language interaction," in *Artificial Intelligence Applications for Business*, W. Reitman, Ed. Norwood, NJ: Ablex, 1984.
[31] ——, "Expert questions—Adapting to user's needs," Computer and Information Sci., Univ. of Pennsylvania, Philadelphia, Tech. Rep. MS-CIS-84-19, 1984.
[32] W. Woods, "Theoretical studies in natural language understanding: Annual report," Bolt Beranekand and Newman Inc., Cambridge, MA, Tech. Rep. 4332, 1979.