1989

Intelligent knowledge acquisition system

Bong-Soo Youn

Rochester Institute of Technology
School of Computer Science

Intelligent Knowledge Acquisition System

by
Bong-Soo Youn

A thesis, submitted to
The Faculty of the School of Computer Science,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Approved by:

Professor John A. Biles 3/10/89

Dr. Stanislaw Radziszowski

Dr. Peter G. Anderson 10 March 89

March 8, 1989

Table of Contents

Abstract

IKAS (Intelligent Knowledge Acquisition System) is an expert system building shell with a machine-assisted concept checker for easy integrated rule-entry. Unlike most expert system shells currently available, IKAS not only performs rule syntax checking, but also maintains semantic integration of the knowledge base by using a Concept Base. The Concept Base is incrementally grown during the cycle of expert system application development and safeguards the whole domain from conceptually conflicting rules by finding possible inconsistencies or duplications of concepts and giving recommendations for possible solutions. Also automatic generalized concept forming using the Inductive Extension Generalization method gives IKAS a capability to discover new inductive hypotheses from the existing knowledge base. IKAS can be an advancement for the current expert system technology in terms of complex knowledge representation and utilization.

Chapter 1 . Introduction

The boundary between current expert system technology and future Artificial Intelligence solutions is composed of two very difficult factors. First is a hardware limitation.. Secondly, many real applications require that information be constantly reevaluated to take environmental changes into account [Harm85]. Such systems will have to be able to learn from their own experience and constantly update their own rules about the domain area as time goes by. There is need, then, for an intelligent expert system shell with a learning capability to facilitate a more structured way of transferring domain knowledge from a human expert to a machine, a more intelligent way to construct the derived knowledge from the existing knowledge. When equipped with a way to discover new facts and theories from observations, the computer then can be left running every night to analyze data and discover useful information [Kamr87]. With the fast growth of computer usage and expert system applications, the field will certainly welcome the possibility of these learning shells.

In this thesis I would like to implement a small expert system shell, with a capability to acquire knowledge intelligently, called **IKAS - Intelligent Knowledge Acquisition System**. Most of the currently available expert system shells do not allow domain knowledge to be changed once it is delivered for use. Preservation of the expert's knowledge from accidental change by not allowing knowledge update justifies the current predominant technology for expert system shells, but if we consider the consistency of the knowledge of the system, even after years of using it, there should be an advanced mechanism to update the old and learn new knowledge of the domain to keep up with the changing environment. To keep the knowledge accurate and integrated, the system should, of course, have a method to detect inconsistency, remove instances of redundancy among rules, and possibly generate derived rules from the existing rulebase.

Chapter 2 through 4 of this thesis discuss the current concepts in the areas of human and machine learning and how they apply to my own needs for IKAS practically. Chapter 2 addresses the issue of **human learning** to motivate a more plausible approach to machine learning. Particular attention will be paid to how human knowledge can be represented, stored and retrieved. In Chapter 3 **Conceptual Dependency Frames** will be discussed for slot-filling, top-down and bottom-up processing theory, which I intend to use in IKAS for preserving consistency of the conceptual knowledge. In Chapter 4, **inductive learning** methods and generalization algorithms are discussed. Several ways to generalize from specific instances are also given, which may be needed for learning from instances. Chapter 5 discusses the **implementation of IKAS**, with its overall knowledge representation scheme and the way to detect and solve conflict or duplication of IKAS knowledge base. Chapter 6 explains briefly the basic **functionalities of IKAS** as an expert system shell. Finally Chapter 7 concludes this thesis with discussion of future work remained. Appendix A gives an explanation of the IKAS version of ARL(Abbreviated Rule Language). Appendix B gives an example knowledge bases of IKAS.

Chapter 2. Human Comprehension

Let's examine human information processing. Figure 2-1 gives a view of humans as information-processing devices based on Atkinson and Shiffrin [Thom85]. Sensory input consists of environmental input such as a "voice" we hear, "colors" we see, etc. The processes of attention and pattern recognition help in the identification and selection of information for further processing. To do this attention and pattern recognition we need to draw on background knowledge from long-term memory. These processes will be discussed in detail in the next section under human comprehension.

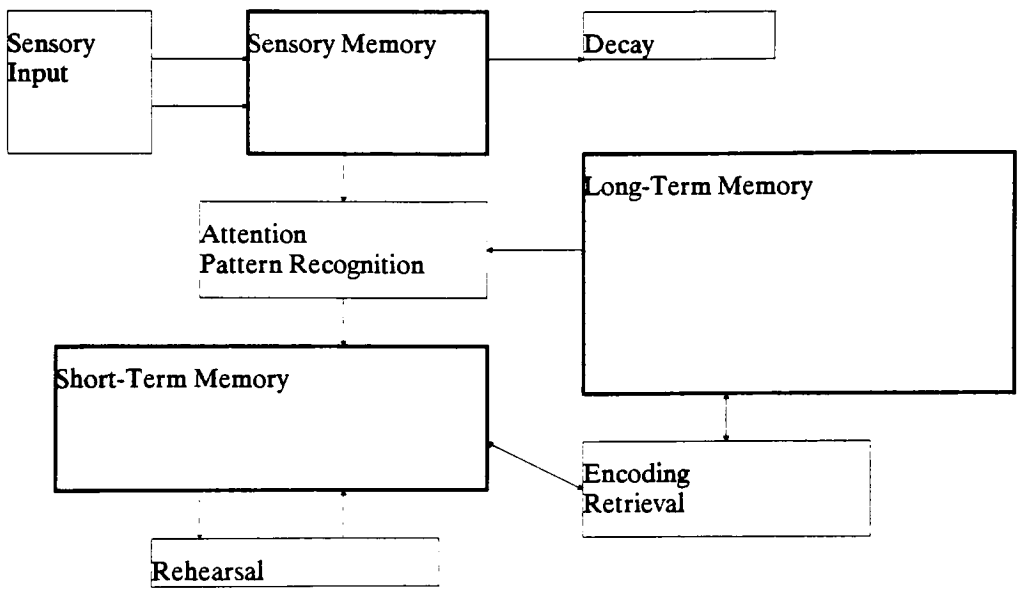

Figure 2-1 Human Information Processing

Information in short-term memory consists of a representation of the external environment from sensory information, and previously learned information retrieved from long-term memory. This information exists only for a very short time (some experiments suggest 15 to 30 seconds) [Thom85]. If we want to remember something for the future, we encode short-term memory to long-term memory. If it is not encoded properly and we do not continue to process i.e. reherse it in short-term memory, it will be decayed. The most obvious factor in decaying is the passage of time. The longer the time interval between learning and recall of information, the more likely it will have been decayed. More importantly, however, it seems that how well the information was learned and encoded originally is a factor in its preservation. In other words, how comprehension about an event was managed will be another factor in determining how well something is remembered [Darl84].

2.1. Human Comprehension as a Theme

Human comprehension is guided by the conceptually driven processing of hypotheses based on expectation. Many psychological studies reveal that an individual may receive several different pieces of information, but all of those with the same theme will be integrated into a single memory representation during the comprehension processes. Later it is difficult to remember the exact input sentences in subsequent memory tasks. After the meaning is comprehended, the exterior structure and exact words are forgotten.

(1)
- The house was in the valley.
- The house was little.
- The house was yellow.

(2) The little yellow house was in the valley.

Figure 2-2 Theme Example

For example, a person originally may have heard the sentences of (1) in Figure 2-2, but he remembers it as (2). The most important conceptually driven force in comprehension is the theme of the material. A comprehender's knowledge about what the passage pertains to can guide the understanding of the material and it is very difficult to understand a passage that seems to have no theme at all.

One's prior knowledge of the general semantic domain of the theme of the material can also affect the comprehension of information. For example, knowledge engineers with a high degree of knowledge about a domain area, understand more and perform better than those with little domain knowledge. These "high-knowledge" engineers are better able to integrate the new information with the goal structure in their stored knowledge about the domain.

Prior knowledge also can come in the form of a point of view, which can help organize information thematically as it is comprehended. Here is an example:

Pichert and Anderson gave subjects a story to read about two boys playing in a house. One group was told to read the story from the point of view of a burglar considering robbing the place; these subjects tended to remember details about valuable objects, isolation from surrounding houses, and other such details of relevance to a potential burglar. The other group of subjects remembered from the point of view of a real estate agent; these subjects remembered details such as size and number of rooms, condition of the house, and quality of the yard [Thom85].

2.2. Schema Theory

An important development in cognitive psychology is the schema theory (Alba & Hasher, 1983). The assumption of this approach is that spoken or written text does not in itself carry meaning, rather it provides directions for listeners or readers on how to use their own stored knowledge to retrieve and construct the meaning. In other

words, the goal of schema theory is to provide the interface between the comprehender and the context. Schema theory goes back to the philosopher Immanuel Kant(1781), who noted that concepts only had meaning insofar as they could relate to knowledge the individual already possessed. The definition of a schema is "a unit of organized knowledge about events, situations, or objects", or technically "a data structure for representing a generic concept in memory" (Moates & Schumacher, 1980). Schema affect how we process new information and how we retrieve old information from memory, by following these three basic principles:

- Selection

From all the information in a given event or message, only some will become incorporated into the memory representation. The pre-existence of an appropriate schema in memory will influence the selection process greatly. If no such schema exists, then both comprehension and encoding will be poor. By identifying the theme, more important and pertinent information of the context can be selected, processed more deeply and better remembered.

- Interpretation

The basic structure of schemata consists of slots where particular information is "filled in" when a schema is instantiated. These slots are used for information acceptance or retrieval of inferences. The inferences drawn during comprehension perform two general functions. First they make connections between information in new material and knowledge already in long-term memory. Secondly, they fill in empty slots implicitly. For example, if you heard "Frank hit the nail," you may infer "Frank used a hammer." In other words, if a strongly implied instrument of some action is not explicitly mentioned, it may be inferred and added to the memory representation. In addition, if no value is provided for a given variable in the certain instance, then a default inference is used to fill in the most typical value which is reserved in the selected schema. For example, if you heard "the hungry snake caught the mouse," the

knowledge of snake and hunger enable you to infer that the snake ate the mouse. Such information was not stated in the stimulus input (sensory input), rather it was filled in by using default inferences and inferring that catching is followed by eating, which may be inaccurate. If you heard that "the little girl caught the mouse," a whole different schema and inference will be used and interpreted quite differently.

- Integration

Schemata are embedded within each other and form a hierarchical structure of schematic information. For example, we may have a schema for "face", which contains nose, mouth and ears. We may also have subschemata for noses, mouths and ears, all subsumed under the global schema "face."

2.3. Conclusion

In human comprehension, phonemes and syllables are perceived, recognized and combined to words. To construct a whole sequence of the words, the human retrieves the knowledge schemata from memory and draw inferences based on this knowledge. We humans typically have top-down (conceptually-driven) and bottom-up (data-driven) processing going on simultaneously. Such processing involves expectation about how to interpret the data based on our previous experience or knowledge in long-term memory. Such expectations control our interpretation of that data and prevent us from doing purely data-driven processing. The most important subjects in this chapter dealt with "filling slots" in schema theory and top-down and bottom-up processing. These are very important aspects of "Conceptual Dependency Frames" by Schank, and will be explained in more detail in Chapter 2.

Chapter 3. Learning Approach using Conceptual Dependency Frame

Automatic concept learning from a large amount of complex input data is a difficult process. This chapter discusses the conceptual dependency frame for comparing human information processing to that of a machine. Conceptual Dependecny (CD) is frequently used for natural language processing techniques. But the ideas of using slots with expectations and combining top-down and bottom-up processing techniques for comprehending the new concepts are precious tool behind conflict/duplication checker in IKAS. We will look at an interesting experiment done by Mallory Selfridge(1980) involving a machine learning program developed to model one young child, Joshua, as he learned to understand simple language at a level between one and two years of age. The program starts with the level of knowledge that Joshua had at age one and it succeeds in demonstrating the understanding and comprehension abilities of a child at age two.

3.1. Conceptual Analysis

Let's consider the following sentence:

- John went to Buffalo.

The analyzer goes through the sentence from left to right, word by word. First, it finds the word JOHN and looks it up in a special dictionary which returns the CD structure. It returns a CD for JOHN as:

(PERSON FIRSTNAME(JOHN))

JOHN is saved in some place, and the analyzer goes to WENT. The dictionary says WENT is the past tense form of the verb "to go" and returns the following CD form:

(PTRANS ACTOR (nil)
FROM (nil)
TO (nil)
TIME (past))

ACTOR, FROM and TO are empty slots to be filled later. TIME is a slot that is filled with value "past". Now the analyzer makes this CD form of PTRANS to be the backbone of the entire sentence to be understood and decides that the meaning of the sentence is a Physical Transfer of a person named John. Notice that before this point it was a bottom-up process, which tried to understand by scanning on a word by word basis. After the analyzer figures out the backbone of the sentence, top-down processing then takes place. The expectations associated with the slots of the PTRANS CD form are used to interpret the rest of the sentence. The next word is TO. By satisfying one of the syntactic expectations for the PTRANS slots it is implied that the possible filling for the TO slot is coming, which in this case is BUFFALO. BUFFALO is a suitable candidate for both the FROM and TO slots, but since it is following TO in the sentence, by syntactic expectation, TO is the right slot for BUFFALO. As a result, the following CD form is constructed:

(PTRANS ACTOR (PERSON FIRSTNAME(JOHN))
FROM (nil)
TO (BUFFALO)
TIME (past))

The significant findings of this CD analysis are:

1) Conceptual analysis is expectation driven. All empty slots in the CD frame have expectations about possible fillers. In human comprehension, as explained in Chapter 1, schemata have slots for interpretation. For example, the concept about California has a slot for life-style in California. If you hear that some one is from California, you will infer that the life-style of the person includes swimming at the

beach or is easy-going, simply from the fact that you heard the person is from California. This is because subconsciously, the slot for life-style in California already has an expectation for an easy-going life style.

2) Conceptual analysis is dictionary based. The dictionary entry of a word contains a great deal of information about meaning and proper usage of the words. The knowledge is stored using special routines that generate conceptual frames.

3) Conceptual analysis is done by using both bottom-up and top-down processing. Also both syntactic and semantic processing are done simultaneously, with the semantic taking precedence over syntactic. Since the goal is to understand the meaning of the sentence, syntax is only used when it helps in semantic analysis. In human comprehension, bottom-up and top-down processing is done in parallel. Pattern recognition is a very general phenomenon of human information processing and it deals with how we recognize environmental stimuli(bottom-up) as exemplars of concepts(top-down) we already have in our long-term memory. In addition to pattern recognition, humans have the capability of generalizing concepts from positive and negative instances.

Using a Conceptual Dependency Frame with slots filled with expectations, generalizations can be achieved. The following section discusses a computer modeling of one child's learning process.

3.2. Modeling of learning Process

Before proceeding, we should note the assumptions the experiment is based upon. First, a child learns language by using a considerable repertoire of previously acquired concepts and world knowledge. This is very similar to the aspects of Schema Theory discussed under interpretation of Section 1.2. Second, children hear language

in certain situations that enables them to infer meaning (selection in Schema theory). Third, to achieve more specific behavior of the program, the model is based on observation of a single child. It is important to describe what kind of knowledge Joshua(Human) had at age one since the modeling program starts at that level, here it is:

- Knowledge of Object

Experiment: Joshua takes a toy cup, and pretends to drink from it.

Clearly, Joshua knew that cups are for drinking purpose. He had the knowledge of the functions and properties of the object.

- Knowledge of Relations

Experiment: Joshua is playing with toy nesting barrels. Joshua takes one of the barrel halves and mouths it. He makes a little stack of barrel halves, two high.

Joshua had the knowledge of the relations that exist among objects. In the experiment above, Joshua put one barrel half onto another. He seems to understand the relationship "on top of".

- Knowledge of Actions

Experiment: Joshua is playing catch with his mother and sister. They are rolling the ball among themselves. Joshua catches the ball and "throws" it to his mother.

Joshua had the knowledge of the understanding actions and functions associated with the given object.

3.3. Program's Initial Stage

We know some of the knowledge Joshua had at age one. Now let's look at the program's knowledge as shown in Table 3-1.

People	Mallory, Child, Parent
Object	Table1, Box1, Ball1, Block1
Relations	(TOP VAL(NIL); on top of (CONTAINED VAL(NIL)); inside of
actions	(DISAPPEAR VAL(NIL)); disapplear (PTRANS ACTOR(NIL) OBJECT(NIL) TO(NIL) ; physical transfer

Table 3-1 Program's Intial Knowledge

The program begins with the similar (not quite the same) knowledge that Joshua began with, but with no language knowledge. The user gives the program a command and an optional simulated visual input. Then the program infers the response to the input and executes that response by printing it. The program uses the CD Frame developed by Schank (1973) to represent Joshua's conceptual knowledge and his understanding of words. As discussed in Section 3.1, a CD frame consists of a symbol representing a concept followed by labeled slots, which carry particular information associated with that instance of the concept or contain pointers to other concepts.

```
CONCEPT      TOP        TEMPLATE    (TOP VAL (NIL))
CONCEPT   PTRANS     TEMPLATE   (PTRANS ACTOR(NIL)OBJECT(NIL) TO(NIL))
```

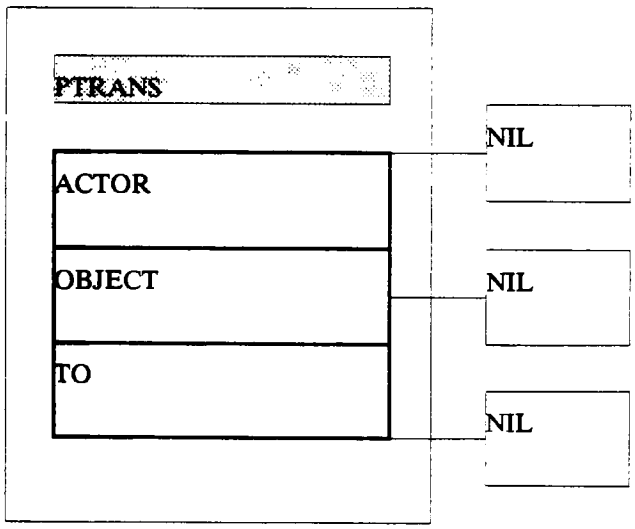

Figure 3-1 Conceptual Dependency Frame

It is our assumption that the program has knowledge of objects, actions and relations equivalent to Joshua's at age one. So somewhere in the knowledge base, the program keeps the templates of concepts consisting of the objects, actions and relationships shown in Fig.3-1.

3.4. Response Inferring Rule

The model is based on the belief that the prerequisite for learning the meaning of a new word is an understanding of the utterance. In order to understand the meaning of the utterance, a child infers the desired response to that utterance. Also it is the only way we can guess whether or not the child really understood the

meaning of a new word. Later on, we will see how these response inferring rules are used in the program that receives the command from the user (parent) and prints out the response. Here are some of the response inferring rules used in the example program run in Section 3.5:

- Function and property inference

If an utterance is heard while interacting with an object, that object on the basis of function or property becomes the inference of that utterance.

- Emphasis-Focus

Attend to the louder words in an utterance.

- Event-Name Inference

If an emphasized unknown word is used in an utterance containing no other known words, and a response can be inferred for that utterance, infer that the meaning of the unknown word is the response to the utterance.

- Meaning-Refinement Inference

If part of the learned meaning of a word is not part of the inferred response to an utterance containing that word, then remove that part from the meaning.

- Event-Naming Inference

If a word and an object are simultaneously drawn to attention, infer that the word is the object's name.

3.5. Program Run

The following two sample program runs show how the program learns without having any previous knowledge of language. The first word the program learns is "bye bye" as shown below:

Program Run 1

- parent says: bye bye
- child sees: (mallory leaves)
- child attends to "bye bye" (rule used: emphasis-focus)
- child infers "bye bye" means
"(disappear object (mallory))" (rule used: event-name)

Program Run 2

- parent says: byebye slippers
- child sees: (parent removes slippers)
- child attends to "slippers" (rule used: emphasis-focus)
- child infers "slippers" means "(slippers1)"
(rule used: event-name)
- child generalizes "bye bye" from
"(disappear object (mallory))" to
"(disappear object (nil))" (rule used: meaning-refinement)

Let's stop here to note what is happened. After program run 1 the system is already equipped with the knowledge about the word "BYEBYE" meaning "MALLORY DISAPPEARS". "BYEBYE" becomes the concept and template is stored as:

BYEBYE - (ACTION OBJECT)

A new frame is generated with slots ACTION and OBJECT. "MALLORY" and "DISAPPEARS" fill the OBJECT and ACTION slots respectively. In human comprehension, as discussed in Chapter 1, the schema about "BYEBYE" is kept in

memory regardless of its truth. Here in the program, the frame about "BYEBYE" is generated and "MALLORY" is placed in the OBJECT slot and the child associates only "MALLORY" with "BYEBYE". In program run 2, "BYEBYE" is the key word to recall the CD frame (schema, in humans) and the program expects "MALLORY" instead of "SLIPPERS". So a conflict exists, and the program generalizes the concept of "BYEBYE" from (DISAPPEAR OBJECT(MALLORY)) to (DISAPPEAR OBJECT (NIL)).

3.6. Conclusion

In this chapter we have seen a computer model of the process used by a child to learn a small subset of language. This model began with only non-language knowledge gained through previous experience (social input). An interesting point made here, is that when an uncertain concept is given, the program acts in a similar way as a human in terms of retrieving frames(schema, in humans) and trying to fit new word to existing slots. Also, if there is a conflict between a new input and a learned concept value, the system generalizes and continues learning. In humans, one class of theories of pattern recognition involves feature analysis. The sensory input is analyzed to produce a set of specific perceptual attributes called features. After such analysis, the resultant list of features is examined for a possible match to a previously learned concept. Sometimes during or after feature analysis, the concept matched will be restructured to support new feature lists or to correct wrong listing of features it already learned. The next chapter deals with how we can schematically represent the structure of these concepts and features.

Chapter 4. Inductive Learning

To make a more generalized concept about a fact(event,situation) or a more generalized meta-level rule from existing specific rules, the mechanism of inductive learning is essential. When there is a conflict between a new rule and an old concept, an induction routine will try to resolve the difference or possibly learn supplementary things. There are at least three occasions where this could happen. First, the new rule could be incorrect and can be corrected according to the old concept. Second, the old concept could be incorrect could be changed according to the new rule. Third, if the new rule is correct as well as the old concept, there should be a generalization made based upon both the new rule and the old concept, and the old concept becomes more generalized knowledge about the facts. Of course during this process, human interaction may be required to confirm the changes. In the following sections, inductive learning is discussed along with several methods of generalization discussed in Section 4.4.

4.1. Types of Inductive Learning

Inductive learning is a process of acquiring knowledge by forming new inference rules based on facts gotten from the environment or a teacher. This process involves methods for generalizing, specializing, transforming and refining knowledge representations. Inductive learning can be subcategorized into learning from examples and learning from observation. Learning from examples can be viewed as a search for plausible general concept description (inductive assertion) which explains the given input data from a set of examples and counter examples, it is useful for predicting new data. Learning from observation, or unsupervised learning, includes the theory formation system, classification system and similar tasks without the benefit of external teachers.

In learning from examples(Concept Acquisition), the F(Facts) is a characterization of some object(situation) preclassified by a teacher into one or more classes(concepts). The induced hypothesis can be viewed as a concept recognition rule such that if a new object satisfies this rule, then it is a member of the induced class(concept). Let's see following example about learning from example and observation:

```
(learning from example)
if shape is box and
            has_keyboard is yes and
            color is white and
            has_monitor is yes
then
            object is computer;

(learning from observation)
Most computers are white_color;
```

4.2. Applications of Inductive Learning System

Before we go further into inductive learning, let's look at the potential for application of a learning system. Probably the most important application is the automatic construction of knowledge bases for expert systems. Currently a few commercially existing expert system shells are equipped with so called "induction table", which is used to automatically generate the rule base from the specific examples of the table. VP-Expert and Rule Master™ are the examples of them. These shells allow the user to create the table with variables and attributes, and then, according to the each rows of the table, conjuntive rules are automatically created. Neither of these shell has the capability of generating generalized rules, still this technique provides an excellent alternative to the tedious process of formalizing an

expert's logic. A less direct but potentially promising use of inductive learning systems is in the machine assisted refinement of a knowledge base during expert system development. In this case, a learning system could be used to detect and correct inconsistencies, to remove redundancies and to simplify the knowledge base using conceptual analysis, which is mentioned in Chapter 3 of this thesis.

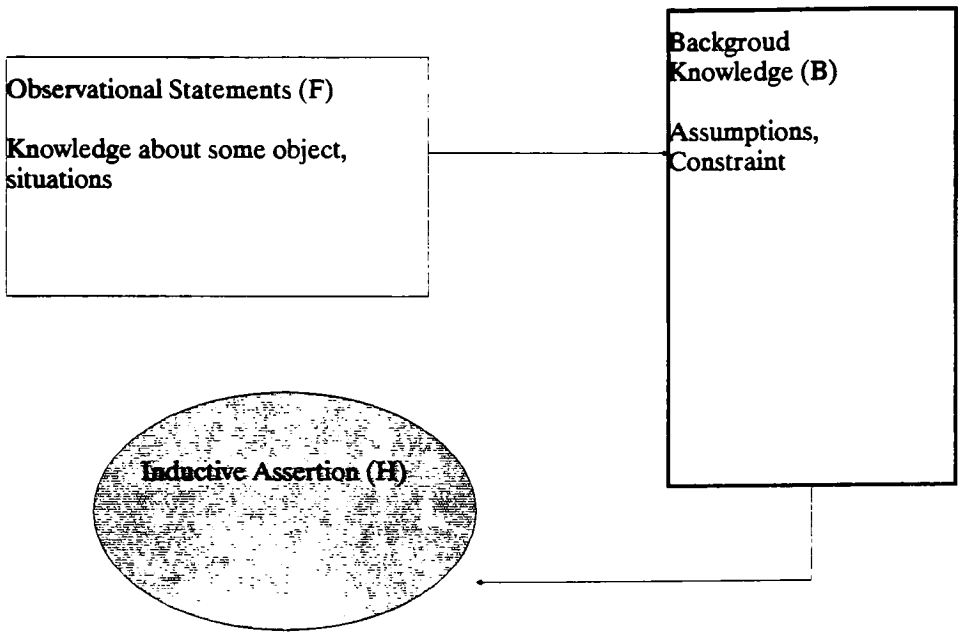

Figure 4-1 Inductive Inference

From the viewpoint of the applications learning system, such as constructing a new rule (derived rule) or checking consistency of a knowledge base using conceptual analysis, the most important decision to make is the description space we are dealing with. In this thesis Conceptual Inductive Learning will deal with the symbolic

descriptions expressed in high-level descriptions typically applied to real world objects, rather than abstract mathematical concepts or computations.

4.3. Inductive Inference Paradigm

In constrast to deduction, premises of induction are specific facts rather than general axioms. The goal of inference is to formulate plausible general assertions that explain the given facts and are able to predict new facts. In Another words, inductive inference attempts to derive a complete and correct desription from specific observations of phenomenon or parts of it. In figure 4-1, Observational statements (facts) F represent specific knowledge about some objects, situations, process, and so on. Background knowledge (B) defines the assumptions and constraints imposed on the observational statements and generated candidate inductive assertions. A, Inductive Assertion (hypothesis) H tautologically implies fact F if F is a logical consequence of H such as if the expression H = F is true under all interpretations, and the following expressions hold.

H | > F (H specializes to F) or F | < H (F generalizes to H)

4.4. Generalization Rules

In the following rule format, "D" stands for some of the arbitrary expressions (context descriptions) that are augmented by additional components. Here is one example of how rule format can be interpreted:

A & B ::> K

Which is read as 'The condition A and B is in class K." and can be turned into a generalization rule:

A & B ::> K | < A ::> K

Which is read as "The condition A and B belongs to class K, which can be generalized as the condition A belongs to class K." Let's say A stands for "apple",B stands for "red" and K stands for "fruit" then the generalized rule above says: "a red apple is a fruit" can be generalized to "an apple is a fruit"

4.4.1. Selective Generalization Rules

If we have the generalization rule:

$$S1 :: > K \quad | < \quad S2 :: > K$$

This rule is called "selective" if S2 has no descriptors other than those used in S1. If S2 contains descriptors that are not in S1 then the rule is called "constructive". The generalization rules listed in Section 4.4.1 is examples of selective generalization rules.

4.4.1.1. Condition Dropping Rule

$$D \,\&\, S :: > \quad K | < \quad D :: > K$$

This rule implies that a concept description can be generalized by simply removing a conjunction. Please look at the example above ("an apple is a fruit").

4.4.1.2. Alternative Adding Rule

$$D1 :: > K \quad | < \quad D1 \,V\, D2 :: > K$$

This rule implies that concept generalization can be achieved by adding logical disjunction. This can be easily explained by extending the scope of permissible values of one specific descriptor. Let's look at the following example:

$$D \,\&\, [color = blue] :: > \quad K \quad | < \quad D \,\&\, [color = blue \,V\, yellow] :: > K$$

4.4.1.3. Interval Closing Rule

D & (L = a) :: > K
D & (L = b) :: > K
| <
D & (L = a..b) :: K

where L is a linear descriptor and a and b are some specific value of descriptor L. The two left rules are associated with a logical conjunction. Let's consider a machine with different temperatures 'a' and 'b'. 'K' represents the normal state of the machine. The left two rules said that if the temperature is 'a' or 'b' then the machine is in a normal state. The right side rule generalizes the left rules by allowing the temperature to be in the interval 'a..b'.

4.4.1.4. Climbing Generalization Tree Rule

D & (L = a) :: > K
D & (L = b) :: > K
D & (L = i) :: > K
| <
D & (L = s) :: K

where L is a structural descriptor representing the highest parent node in the structural hierarchy whose descendants include a,b..i. The following example illustrates the rule:

there exists P, D & (shape(P) = triangle) :: > K
there exists P, D & (shape(P) = rectangle) :: > K
can be generalized to
there exists P, D & (shape(P) = polygon) :: > K

4.4.1.5. Turning Constants into Variables Rule

F[a]
F[b]
F[c]
.
.
F[i]
|<
For All v,F[v]
where F[v] stands for some description dependent on variable v, and a,b,.. are constants

A corresponding rule for concept acquisition is:

F[a] & F[b] & ... ::> K |< There Exists v, F[v] ::> K

Assume that 'a','b' and so on, are parts of class K that have property K. The generalization rule replaces the constant with a variable and states that if any part of an object has property F then the object belongs to class K.

4.4.1.6. Turning Conjunction into Disjunction Rule

F1 & F2 ::> K |< F1 V F2 ::> K

A concept description can be generalized by replacing a conjunction operator with a disjunction operator. Let's see the following example:

F1: the animal flies
F2: the animal lay eggs

F1 & F2 ::> Bird can be generalized to
F1 V F2 ::> Bird.

i.e. if animal flies or animal lay eggs then animal is bird.

4.4.1.7. Inductive Resolution Rule

P & F1 ::> K
not(P) & F2 ::> K
|<
F1 V F2 ::> K
where P is predicated and F is formular

Assume that K is the situation where a person goes to a movie. The person will go to the movie if it is a weekday (P) and they have nothing else to do (F1), or a person will go to a movie if it is a weekend (not(P)) and the movie has recieved a high rating (F2). The generalized rule says a person goes to the movies when either the person has nothing to do or the movie has been highly rated.

4.4.1.8. Extension Against Rule

D1 & (L=R1) ::> K
D2 & (L=R2) ::> not(K)
|<
(L=/=R2) ::> K
where sets R1,R2 are disjoint

This rule is the basic rule for learning discriminant descriptions. From a description of an object which belongs to K(positive example) and a description of an object not belonging to K(negative example) the generalization rules out the negative example.

4.4.2. Constructive Generalization Rules

These rules generate inductive assertions that use descriptors not present in the original(left-side) statement. This means the rule performs a transformation of the

original representation space. Background knowledge remembers relationships among objects and previously learned concepts.

D & F1 ::> K
F1 => F2 |<
D & F2 ::> K

This rule says if a concept description contains a part F1 that is known to imply another concept F2 (through background knowledge) then a more general concept description is obtained by replacing F1 with F2.

there exists P, (color(P) = black)(width(P) & length(P) = large) ::> K

.

suppose the system already had a concept about AREA such as
for all P,((width(P) & length(P) = large) = (area(P) = large)

then the generalization rule is produced as:
there exist P,(color(P) = black)(area(P) = large) ::> K

4.5. Conclusion

Inductive learning systems can be used for the automatic construction of a knowledge base for expert systems by producing production rules and semantic networks. This can shorten the long process of formalizing and encoding an expert's knowledge. Also, inductive learning can be used to refine a knowledge base initially developed by a human expert by detecting inconsistencies, removing redundancies and simplifying the expert-derived rules. Machine Learning itself is still a broad and difficult area in Computer Science, especially for practical usage. There should be good mechanisms for preventing over-generalizations and for keeping concepts and

facts well integrated. This chapter ends my background reasearch over the learning mechanism and from Chapter 5 I will discuss more specific about the IKAS.

Chapter 5. IKAS

IKAS (Intelligent Knowledge Acquisition System) is a general purpose PC based expert system building shell with limited comprehension capabilities. This chapter discusses the theoretical background underlying IKAS and explains the design of the IKAS implementation. Of course, IKAS has other functions and features in addition to comprehension capabilities; however, for the theme of this thesis, emphasis will be placed on the knowledge representation and comprehension aspects. In this chapter some of the theoretical background from the research done in Chapter 2 through Chapter 4 and will be modified slightly to meet the requirement for the practical aspects of IKAS.

5.1. Overview of IKAS

IKAS (Intelligent Knowledge Acquisition System) is an expert system building shell with a machine-assisted concept checker for easy integrated rule-entry. Unlike most expert system shells currently available, IKAS not only performs rule syntax checking, but also maintains semantic integration of the knowledge base by using a Concept Base. The Concept Base is incrementally grown during the cycle of expert system application development and safeguards the whole domain from conceptually conflicting rules by finding possible inconsistencies or duplications of concepts and giving recommendations for possible solutions. IKAS's concept checking unit allows the user to see the hierarchy of concepts graphically and edit the rules pertaining to only the related topics. More functionality of IKAS will be discussed in Chapter 6.

5.2. Knowledge Representation in IKAS

In Chapter 1, we briefly discussed human learning, focusing especially on how humans comprehend, encode and remember facts (situations). As mentioned in

Section 2.1, individuals receive several different pieces of information, but all of those with the same theme will be integrated into a unified memory representation during comprehension. Using a similar method, IKAS uses the list of features to identify objects (facts, situations, themes). For example, if we want to represent and comprehend the sentence "The little yellow house is in the valley," separate facts such as:

- the type_of_object is house
- the location_of_object is in_the_valley
- the size_of_object is little
- the color_of_object is yellow

would be needed for IKAS to understand the situation. In other words, IKAS represents knowledge in the form of symbolic descriptions. Each fact (situation) is represented by the name of the object, a relation_operator and a value. Also, each fact (situation) may be represented by other specific sub-facts. For example,

- the location_of_object is in_the_valley

could be represented by the following specific sub-facts if we assume that if the location is in the valley, there will be many mountains, trees and a river around the object.

- the number_of_mountains_around is many
- the river_around maybe yes
- the trees_around maybe yes

The above facts will become expectations whenever IKAS encounters the fact about "the location of object is in the valley." This is analogous to the way Conceptual Analysis in Chapter 3 uses slots with expectations.

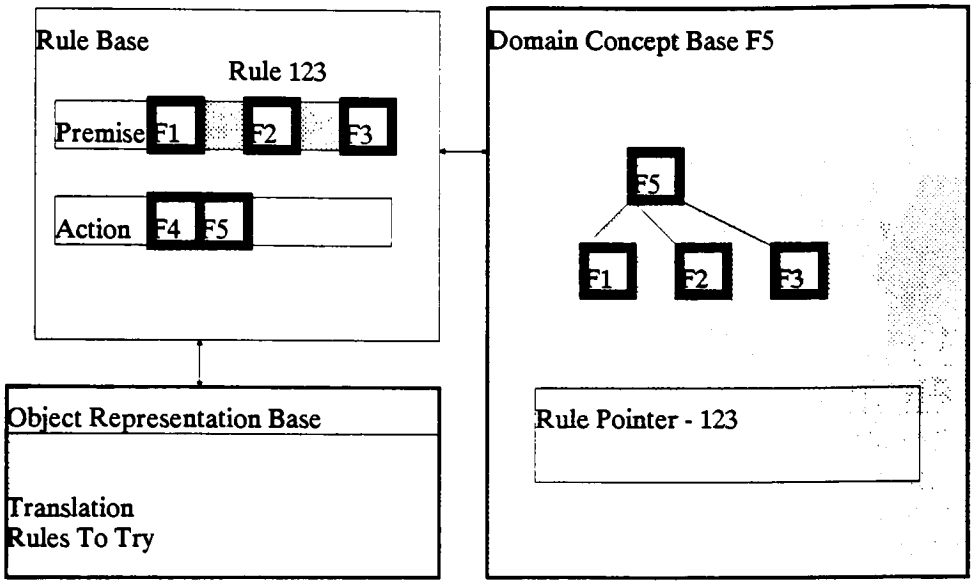

Figure 5-1 IKAS Knowledge Base

To get a more detailed understanding of knowledge representation in IKAS, it is necessary to distinguish the different levels of knowledge structures IKAS maintains. As shown in Figure 5-1, the knowledge base of IKAS is divided into two parts. These are representation-specific and domain-specific meta-level knowledge. Representation-specific knowledge is knowledge about how to utilize the internal knowledge in more structured and efficient ways. For instance, the representation-specific meta-level knowledge, IKAS's internal rule structure, the inference engine, and the rule translation routines allow a user to ask "WHY," "HOW" and "WhatIf," to find out why a question was asked of the user, how a certain conclusion was arrived at, or, if a certain parameter's value were to be changed, how it

would affect the final goal of the system. In Figure 5-1, the Rule Base and the Object Representation Base contain representation-specific knowledge. Domain-specific knowledge maintains the integrity of domain concepts by spotting possible conflicts. Domain-specific meta-level knowledge in IKAS is maintained in the Domain Concept Base and is used for preserving knowledge integration and performing conceptual analysis on incoming rules that influence the parameters (situations). As time goes by and more rules are received, these learned concepts about parameters become background knowledge and are used later as templates for concepts about facts.

The possible use for this background knowledge is very similar to its human usage. When we learn a new fact about some object or situation, we usually compare it with the studied concept about the object in our long-term memory. If it fits well with our old concept about the object, then the old concept is not changed, but if there is a conflict between the new instance and the old concept, then we ask "why" and try to come up with an optimal concept of the object.

5.2.1. Rule Base

The IKAS rule base is the set of rules that describes certain facts, situations or phenomena by Premise (Condition) and Action (Determination) rules. For example, if we want to enter knowledge about "The little yellow house is in the valley" as a fact, which we will call here House_fact_1, we provide the following conditions:

- is(type_object,house)
- is(location_object,in_the_valley)
- is(size_object,little)
- is(color_object,yellow)

House_fact_1 has slots with the attributes given above. Each slot consists of another fact with an object, operator and value. Here location_object becomes object, "is" becomes the operator and in_the_valley becomes the value for the object with the

operator defined. This condition, "is(location_object,in_the_valley)", also can be a fact defined by sub-conditions such as:

- is(number_of_mountains_around,many)
- is(river_around,yes)
- is(trees_around,yes)

By these two facts we can enter the following rules to implement House_fact_1:

```
rule 1: if type_object is house and
                location_object is in_the_valley and
                size_of_object is little and
                color_of_object is yellow
then
                house_fact_1 is yes;

rule 2: if number_of_mountains_around is many and
                river_around is yes and
                trees_around is yes
then
                location_object is in_the_valley;
```

Let's look at another example rulebase, which determines the name of an animal.

```
rule 1: if mammal is yes and
                carnivore is yes and
                color is tawny and
                has_dark_spots is yes
then
                animal is cheetah;

rule 2: if mammal is yes and
                carnivore is yes and
                color is tawny and
                black_stripes is yes
then
                animal is tiger;
```

rule 13: if bird is yes and
does_fly is not yes and
has_long_neck is yes and
has_long_leg is yes and
color is black_and_white
then
animal is ostrich;

rule 15: if bird is yes and
does_fly is yes
then
animal is albatross;

rule 8: if pointed_teeth is yes and
has_claw is yes and
has_forward_eyes is yes
then
carnivore is yes;

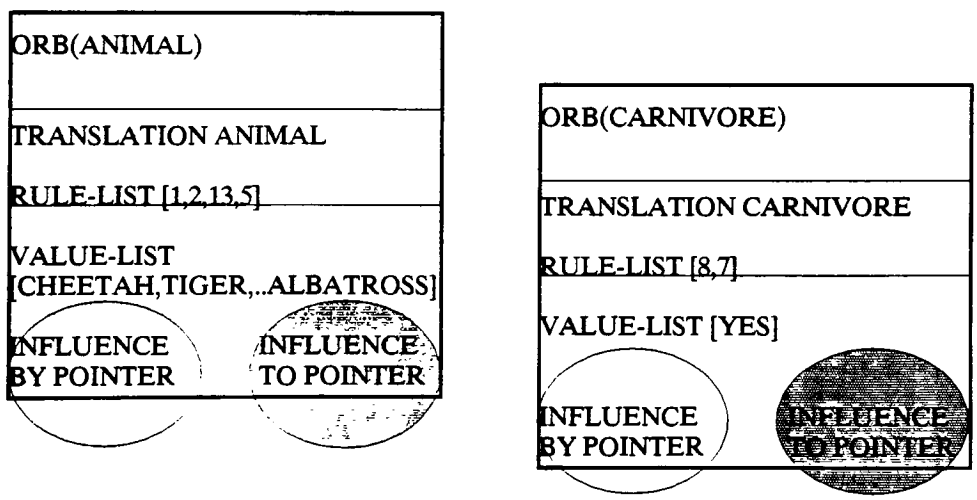

Figure 5-2 Object Representation Base

rule 7: if eat_meat is yes then carnivore is yes;

This animal rule base will be used for examples through the later sections and a complete listing of the rule base is listed in Appendix B.

5.2.2. Object Representation Bases

An Object Representation Base (ORB) is created when a new object is introduced. As shown in Figure 5-2 an ORB contains the object name, the English translation for the object (e.g. the object carnivore's translation is "the animal belongs to the carnivore group") and the list of rules that determine the value of the object (e.g. the animal representation base has pointers to rules [1,2,13,15] and the carnivore representation base has pointers to rules [8,7], etc.). The possible values for the object are also remembered (e.g. animal representation base has a value list: [cheetah, tiger, ostrich, albatross]). The main purpose of the an ORB is to maintain a translation for each object and to maintain lists of rules pertinent to each object so that efficient rule checking can be performed at execution time. Each object has two kinds of pointers to other objects. One is the "influenced by" pointer, which points to the objects that affect this object, and the other is the "influence to" pointer, which points to the objects that are affected by this object. For example, ORB (CARNIVORE) has an "influence to pointer" to ORB(ANIMAL). These pointers are used for maintaining an accurate concept hierarchy.

5.2.3. Domain Concept Base

The Domain Concept Base (DCB) is the set of main-facts and sub-facts represented by a concept hierarchy. The main-facts constitute the action part of a rule and the sub-facts constitute the premise part of a rule. Each sub-fact can itself be a main-fact and could be explained by other sub-facts, and so on, which generates a

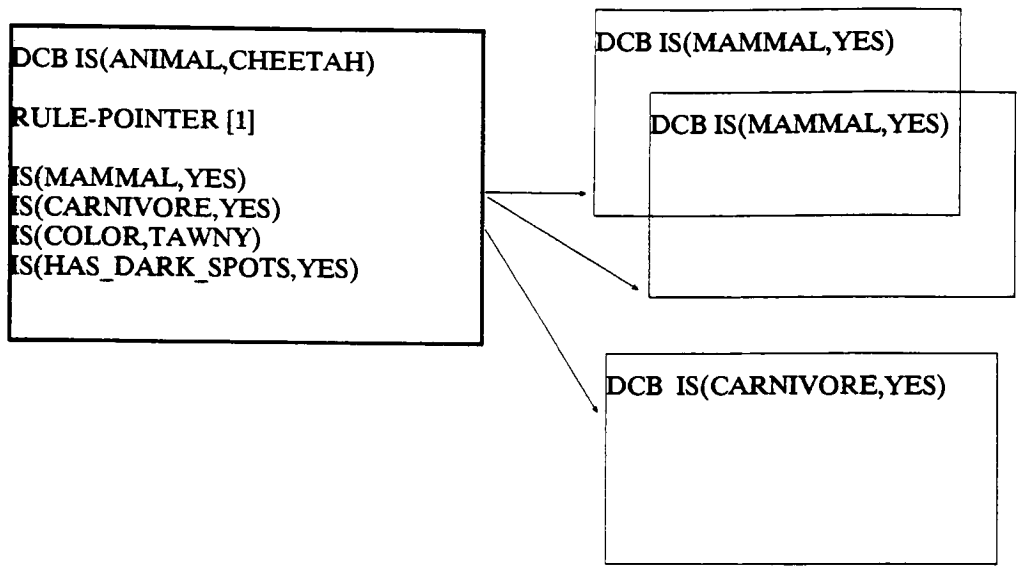

Figure 5-3 Domain Concept Base

concept hierarchy. Each fact consists of an object, operator and value. For example, in the animal knowledge base, "animal is cheetah" is a main-fact. "Mammal is yes," "carnivore is yes," "color is tawny" and "has_dark_spots is yes" can be sub-facts. The fact "carnivore is yes" also can be a main-fact and has sub-facts such as "pointed_teeth" and "eat_meat is yes." Each fact has a pointer to the rule(s) that describe the fact. The main purpose of DCB is to establish a concept hierarchy so that it performs a similar function to that of long-term memory in human comprehension. Whenever a new rule or fact is entered, the conflict-checker unit in IKAS checks the new rule or fact to the learned concept in the DCB and finds any conflicts and/or duplications with existing knowledge. Figure 5-3 illustrates the structure of the DCB, and more detailed role of the DCB and conflict checker algorithm is discussed in Section 5.3.

5.3. Conflict Checker Control Strategy

5.3.1. Conflict Checker Unit

The CCU(Conflict Checker Unit) in IKAS maintains a complete logical-error-free knowledge hierarchy and suggests the removal of any unnecessary duplications of facts among rules. The CCU checks facts for conflicts or duplications, not only at the same level as the fact but also at sub-level(s) of it. By using this technique, the knowledge base in IKAS can be maintained as conceptually correct from start to end. Since each object in the ORB (Object Representation Base) can be checked individually or recursively, incremental knowledge base design is possible without tedious searching of rules for logical errors. The CCU performs a separate level of conceptual analysis according to the type of concept. For simplicity we will categorize concept types as single-condition-facts versus multiple-condition-facts. A single-condition-fact is a fact that is described by a group of facts chained by conjunction operators, such that only one group of facts (sub-facts) exists for determination of the main fact. In the animal knowledge base, each of the facts "animal is cheetah," "animal is tiger," and "animal is ostrich" is a single-condition-fact. A multiple-conditions-fact consists of groups of facts chained by disjunctive operators, such that more than one group of facts (sub-facts) exist for determination. The fact "carnivore is yes" is an example of multiple-conditions-fact because it is determined by the more than one group of facts.

Let's imagine that the following rules are added to the previous animal knowledge base.

```
rule 5: if has_feather is yes then bird is yes;
rule 6: if does_fly is yes and
                lay_egg is yes
then
```

bird is yes;

The CCU finds a conflict between rule 13 and rule 6 in terms of "does fly" and "does not fly." In Rule 13, background knowledge about animal ostrich has the expectation that an ostrich is a bird and an ostrich does not fly. With a general description of birds in rule 6, IKAS realizes the possible incorrectness of its learned concept about either "bird" and/or "ostrich." It does not make any difference whether the concept about bird is entered before the concept of "ostrich." In this case IKAS maintains the description about bird in long-term memory and finds a conflict when the special kind of bird, "ostrich," is introduced with its feature "does not fly." By the same technique CCU finds duplication in terms of "does fly" in rule 6 and rule 15 because rule 6 already implies that birds fly, so there is no need to mention it again in rule 15. This example is only a one-level-deep recursive conceptual analysis; in real expert system applications multiple levels of concepts are represented, and the CCU can be regarded as a valuable device for complex knowledge base maintenance.

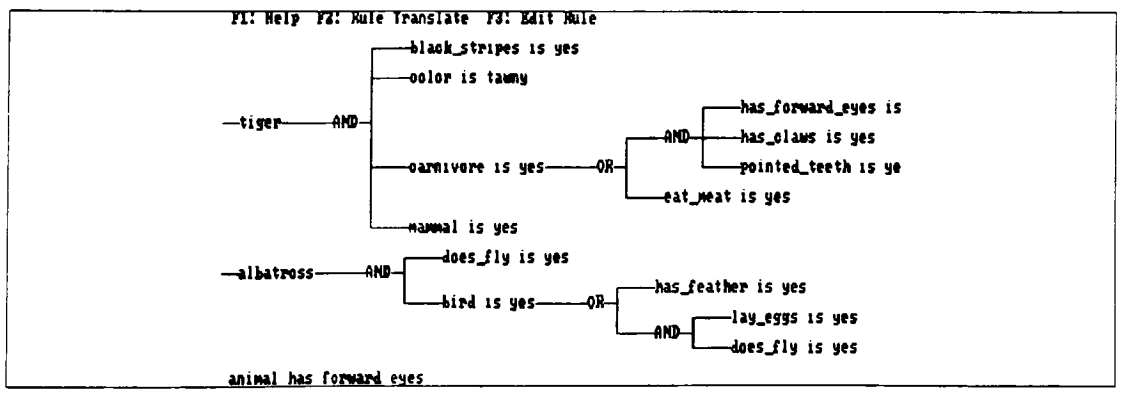

Figure 5-4

5.3.2. Conflict Checker Unit Algorithm

Before we get into the CCU(Conflict Checker Unit) algorithm, let's review and standardize some of terms we have been using throughout this thesis.

- Concept
- Fact
- Object
- Operator
- Value
- Feature list

A Concept has a more abstract meaning and can be described by a group of facts or group of concepts that can be represented by the logical knowledge hierarchy. For example in the animal knowledge base, the concept "animal is tiger" can be represented graphically as shown in Figure 5-4. Each fact consists of an Object, Operator, and Value. When a group of facts represents another fact directly, it is called a feature list. In Figure 5-4 the fact "animal is tiger" has the feature list ["mammal is yes", "color is tawny", "carnivore is yes", "black_stripes is yes"]. With this understanding, let's proceed with the conflict checker algorithm. The CCU can be invoked manually by the user selecting from the list of concepts. This way multiple levels of concepts can be checked and modified individually. Of course, knowledge integrity is not guaranteed unless the checkpoint is at a higher level of the concept hierarchy. Here the checkpoint describes the root of the concept tree, which is the object the user selected. In the animal knowledge base, if a new concept is formed about a bird and it is contradictory to the animal concept, then unless the CCU is called from the higher level of the hierarchy, here the animal node, it is hard to detect a conflict such as shown in Section 5.3.1. For example, Figure 5-5 is an example of detecting conflict and duplication. The checkpoint was the name of the animal, ostrich,

which is a higher level in the concept hierarchy than that of bird. The CCU recursively finds a conflict and a duplication from the feature-list of bird concept.

On the other hand, if the checkpoint is bird alone, the system can find neither a conflict nor a duplication. Therefore, the best way to check for a conflict or duplication in a whole knowledge base is to make a checkpoint for the root goal of the

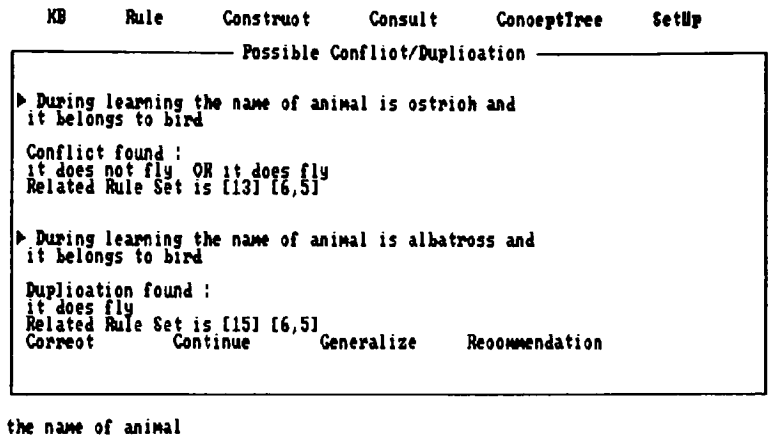

Figure 5-5 CCU Example 1

entire knowledge base. Still, we can find some similarity to human comprehension in that whenever humans form new concepts, either by receiving new instances or changing old concepts, a complete measure is not taken to check for complete knowledge consistency in long-term memory. Checking is done only when the newly formed concept can be related to the situations.

The CCU algorithm is divided into two steps. The first is the Conflict Checker Routine, and the second is the Conflict Solver Routine.

5.3.2.1. Conflict Checker Routine

The Conflict_Check routine looks for a conflict/duplication with the following steps:

(1) Get object checkpoint and call (2) find_value_list(checkpoint).

(2) find_value_list(Object): From ORB get VL (valuelist for Object) and call (3) individual_fact(Object, VL).

(3) individual_fact(Object,[V1,V2,...,Vn]): Repeat calling (4) sub_tree(Object, Vi, TREE) where Vi = {V1,V2...,Vn}.

(4) sub_tree(Object, Vi, TREE):

Get FL (fact list for "Object Vi.");

For each element Fi in Fact List FL = [F1,F2,...,Fn] Begin

If Fi is not represented by sub-knowledge

then

Fi becomes a feature and proceed to (5) conflict_check(Fi);

else

If Fi is a single condition fact then make nodename "AND";

If Fi is a multiple condition fact then make nodename "OR";

Recursively call (4) sub_tree(FObject,FVi,TREE) to get the child node of Fi;

End;

Return TREE with FL as child node;

(5) conflict_check(NewFeature):

If conflict or duplication found

then

proceed to the conflict solver routine;

add NewFeature to "feature_database" and Return;

Here is an explanation of the above algorithm steps in detail.

First, decide the object checkpoint from the user selections. From ORB (Object Representation Base), find out the list of possible values (VL) for the object such as VL = [V1,V2,V3,...,Vn] and call individual-fact(object,VL).

Step (3) individual-fact(object,[V1,V2,V3,...,Vn]) is the node where fact is formed such as "object is V1", "object is V2", "object is V3",...,"object is Vn". Each individual fact can be a concept if it is represented by another knowledge hierarchy, concept checking proceeds to sub_tree(Object,Vi,TREE), where Vi = {V1,V2...,Vn} and TREE is the return value for the tree of "Object is Vi."

Step (4) sub_tree(Object,Vi,TREE) looks in the DCB (Domain Concept Base) to get FL = [F1,F2,...,Fn], which is a fact list for "Object being Vi." In Figure 5-4, for animal being an albatross, the fact "does_fly is yes" is not represented by sub-knowledge, and the fact "bird is yes" is represented by sub-knowledge about bird. For each element Fi in FL a test is performed to find out whether or not the fact Fi is represented by another sub-knowledge hierarchy. If there is a sub-knowledge hierachy then either a single-condition-fact or a multiple-condition-fact, depending on the number of occurrences for Fi in the DCB, will decide the name of the node as either "AND" or "OR". The necessary actions (push and pop) then will be taken to keep the "or_agenda" and "and_agenda" stacks accurate and step (4) sub_tree(FObject, FVi, TREE), where FObject and FVi represent the fact Fi, will be called recursively to construct a TREE for sub-knowledge of Fi. On the other hand if there is no sub-knowledge available, then Fi becomes the leaf of the concept tree and then is regarded as a feature. For example in Figure 5-4, "does_fly is yes," "has_feather is yes,"

and "lay_eggs is yes" become distinct features for "animal being albatross." For each feature, conflict_check (New feature) is performed as step (5).

Step (5) Conflict_check(New feature), where New feature is the format of

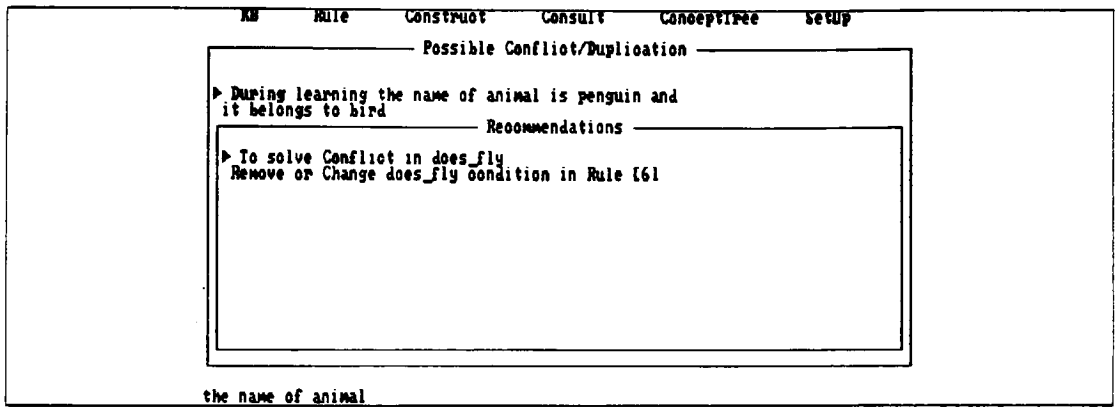

Figure 5-6 Recommendation

(Object, Operator and Value) checks the "feature database," which is created and removed dynamically for each concept for detecting conflict or duplication. The feature database is in the format of feature(Object, Operator, Value, Object stack in and_agenda, Value stack in and_agenda, Object stack in or_agenda, Value stack in or_agenda). For example, in Figure 5-4, there are two "does_fly is yes" features, but each has a different role in supporting "animal being albatross" and are represented by a different internal structure. The first one is a condition for "animal being albatross" and the second one is a condition for "bird is yes" and "animal being albatross." The feature database for the first and second one is as follows:

```
feature("does_fly", "is", "yes", ["animal"], ["albatross"],[],[])
feature("does_fly","is","yes",["animal","bird"],["albatross","yes"],["bird"],["yes"])
```

The CCU algorithm uses the following heuristic; One of the ways to locate conflicting features is to detect for the same Object and Operator but a different Value between two features. One of the ways to locate duplicating features is to detect for the same Object, Operator and Value between two features. In the example above, "does_fly is yes" and "does_fly is yes" is duplicated. Despite conflicting or duplicating features are located, if the "and_agenda" and "or_agenda" for two features is the same, then these two features support each other as a disjunctive condition, and it is not regarded as a conflict or duplication. In the example above, "does_fly is yes" is a duplication since the "and_agenda" and "or_agenda" for the two features are different from each other.

5.3.2.2. Conflict Solver Routine

The Conflict Solver routine recommends how to solve a conflict or duplication with the following steps:

(1) From the "fault database" find out the number of fault occurences by checking the Object, ruleset and fault mode. The Rule set is the list of rules that cause a fault mode that is either "conflict" or "duplication" for the Object. Increase the fault occurences by one.

(2) Find the fault set {Ri,Ii} for the Object from the "fault database" with maximum Ii, where Ri is a ruleset and Ii is the number of fault occurences. Recommend changing Ri.

Whenever the Conflict Checker Routine finds a fault such as a conflict or duplication, the rule set that causes the conflict or duplication is stored in the "fault database" along with the Object that causes the fault, the fault mode, and the number of fault instances, in the format of fault(Object, fault mode, RuleSet, number of fault instaces). Here fault mode is either "conflict" or "duplication." RuleSet is the list of

rules that caused a fault mode in Object. The number of fault instances is used later by the Conflict Solver Routine for recommending how to remove a conflict or duplication, as shown in Figure 5-6. The Conflict Solver Routine assumes that if a conflict occurs (the fault set {Ri,Ii} = {(R1,I1),(R2,I2)...,(Rn,In)} for the Object, where Ri is a ruleset and It is a fault occurrence), then remove the conflict of the Object, by changing or removing the condition that contains the Object from the ruleset (Ri) containing the maximum fault instances. Let's imagine there are 8 balls painted blue and 2 balls painted red. The paint is not dried yet and there will be a hit mark when they collide with each other. Of course when two blue balls collide with each other there will be no mark because they are same color. After a while, the red balls will have more marks than blue balls if each ball collides with every other ball exactly once. The idea is to select a wrong concept from the majority of right concepts.

5.4. Generalization Control Strategy

When we describe either human learning or machine learning, we can conceive of learning as a two step process. The first step is finding a conflict between a new concept and an old concept. The second step is to try to solve the conflict by correcting a wrong concept from either the new concept or the old concept. However, the issue is not as simple as that. While pursuing the second step we are often confronted with the dilemma that there undoubtedly exists a conflict between two concepts; however, each concept could be correct if considered individually. Consequently, correcting either one of these concepts does not solve the problem. For example, in the animal knowledge base as shown in Figure 5-5, the IKAS concept checker routine reports a conflict between rule 13 and rule 6.

rule 13: if bird is yes and
does_fly is not yes and
has_long_neck is yes and
has_long_leg is yes and
color is black_and_white

```
then
        animal is ostrich;
rule 6: if does_fly is yes and
        lay_egg is yes
then
        bird is yes;
```

Rule 6 says that to be qualified as a bird, it should fly and rule 13 says that ostrich is a bird, but ostrich can not fly. Both rules are undoubtably correct conceptually; however, when they are used as a concept representing "animal being ostrich," the conflict is unavoidable because the characteristic of the ostrich's being a bird does not fit with the definition of the bird in terms of "does_fly."

5.4.1. Condition Dropping Generalization

We can think of several solutions to this dilemma. One of them is to generalize the definition of a bird by removing the "does_fly is yes" condition from rule 6, which applies the Condition Dropping Rule in section 4.4.1.1. This implies that a concept can be generalized solely by dropping an element of a conjunction. Even if we use this technique, according to the knowledge represented, we could have several difficulties in accepting the generalized rule as a right concept. One of the difficulties is the result of over-generalization. If we remove the condition "does_fly is yes" from the definition of a bird and entirely rely on the fact "it lay eggs," then it becomes an over-generalized concept about a bird. For example a turtle also lays eggs, but it is not a bird. Another pitfall can be a tradeoff between the generalization of a definition being a bird and abundant specializations of examples of a bird. For example after generalizing rule 6 by dropping "does_fly is yes" condition, every rule that describes any kind of bird should contain the condition "does_fly is yes" if it flies. Still this argument is subject to the knowledge base. As a good example, let's consider the following rules

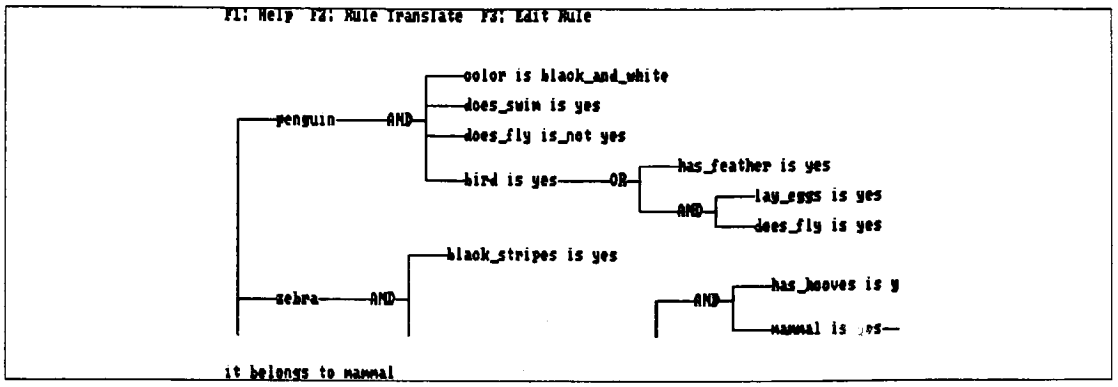

Figure 5-7

rule 1: if object_kind is fruit and
object_color is blue and
object_on_the_table is yes
then
eat_object is yes;
rule 2: if object is apple and
object_color is red
then
object_kind is fruit;

It is natural to say remove "object_color is red" condition from rule 2 to resolve the conflict about the "object_color" condition. Another words, "a red apple is a fruit." can be generalized to "an apple is a fruit." The Conflict Solver Routine, as mentioned in Section 5.3.2.2, has a limited way of recommending which condition in which rule(s) to drop for resolving the conflict.

5.4.2. Inductive Extension Generalization

Now let's discuss another generalization method to resolve conflict. The Inductive Extension Generalization (IEG) method is a new generalization technique developed in this thesis. This method is simillar to the Alternative Adding Rule described in section 4.4.1.2. However, an interesting difference is that the IEG method provides the means for the program to come up with a new derived generalization automatically from the existing knowledge base without manually adding the disjunction rule.

5.4.2.1. Machine Learning Using the Inductive Extension Generalization

For the benefit of illustrating more concise knowledge of naming an animal, from now on the animal KB in the rest of this chapter refers to the complete KB rule listing of animal KB in Appendix B. Let's imagine that rule 5 in the animal KB in Appendix B does not exist and that the only rule to define a bird is rule 6. The goal of the knowledge base (KB) is still to find the name of the animal. The inference engine of IKAS determines the questions to ask according to the rules encoded before the consultation and facts obtained during the consultation (run time). Let's assume that we are expecting penguin to be a name of the animal. Of course we will answer "NO" when we are asked "Does it fly ?" As soon as the answer "NO" is given, the inference engine does not need to check whether it lay eggs or not. Since in rule 6, the conjunctive condition ("does_fly is yes") has already failed, consequently the whole rule 6 fails without checking for the rest of conjunctive conditions in premise part of rule 6. This scenario shows rule 14, which determines animal to be a penguin, never has a chance to succeed to assist the inference engine in determining the name of the animal as a penguin, regardless of the truth that it swims or its color is black_and_white, due to the fact that it is not a bird because it can not fly according to rule 6. In other words, a knowledge deadlock exists such that there is absolutely no

way a penguin can be an answer for the goal no matter how many times this KB is consulted.

Now let's place rule 5 back in the animal KB and consult it again. This time there is a chance for rule 14 to succeed, and the name of the animal can be determined as a penguin as long as we answer "YES" when asked "Has it feather ?" The difference between the previous consultation, which leads to a deadlock, and the present one is that we add the alternative rule to enlarge the description space for the definition of a bird using the Alternative Adding Rule technique as mentioned before. Probably a

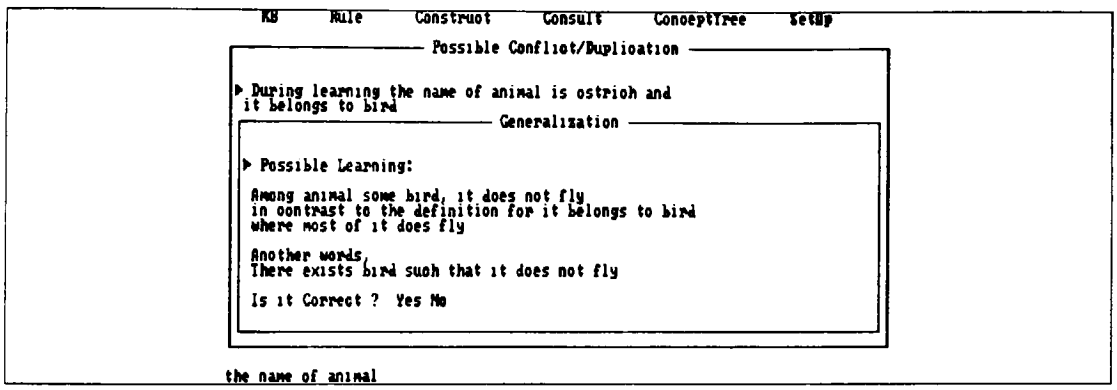

Figure 5-8

better Alternative Adding Rule generalization would be to add one more rule such as the following:

rule 16: if does_fly is not yes and
lay_eggs is yes
then
bird is yes;

The IEG method has a slightly different approach to solve this knowledge deadlock problem. As shown in Figure 5-7, the concept hierarchy for penguin

contains two features regarding "does_fly." The IEG methods consider these two features differently in terms of hierarchy relations. The feature "does_fly is not yes" is a determination factor for the name of the animal, and the feature "does_fly is yes" is a determination factor for the animal being a bird. Also the factor "animal being a bird" is a conclusion factor for the "name of animal being a penguin." In this case "does_fly is yes" is a meta-feature toward "does_fly is not yes," because being a bird determines the name of the animal. The algorithm behind the IEG method is if there exists a conflict between two or more features, find the higher level feature among those conflicting features, treat the higher level feture as a more general case, and treat the lower level feture as more specialized case. Here the general case could be compared to the universal quantifier, and the special case could be compared to the existential quantifier. The new inductive assertion can be formed automatically by creating the existential quantifier, which can be an exception or negation to a universal quantifier. For example the necessary inductive assertion the program needs to avoid deadlock for animal being penguin is "some bird can not fly" or "not every bird can fly." Figure 5-8 illustrates the automatic generalized concept learning using IEG method.

5.4.2.2. Generalized Concept Utilization

It will be useless if a generalized concept is formed but not used at all. We can identify two major usages for a generalized concept. The first usage is to prevent detecting similar conflicts from related topics. As an example, once generalization is done to generate "some bird can not fly" from the specific penguin instance in animal KB, the IKAS Conflict Checker Routine already has the intelligence not to detect a similar conflict for ostrich in rule 13 in terms of "does_fly is not yes" in the animal KB. The second usage is applying this generalized concept during consultation for the purpose of preventing knowledge deadlock. As we recall in Section 5.4.2.1, before a generalized concept is formed, rule 6 in animal KB could not succeed at all if the

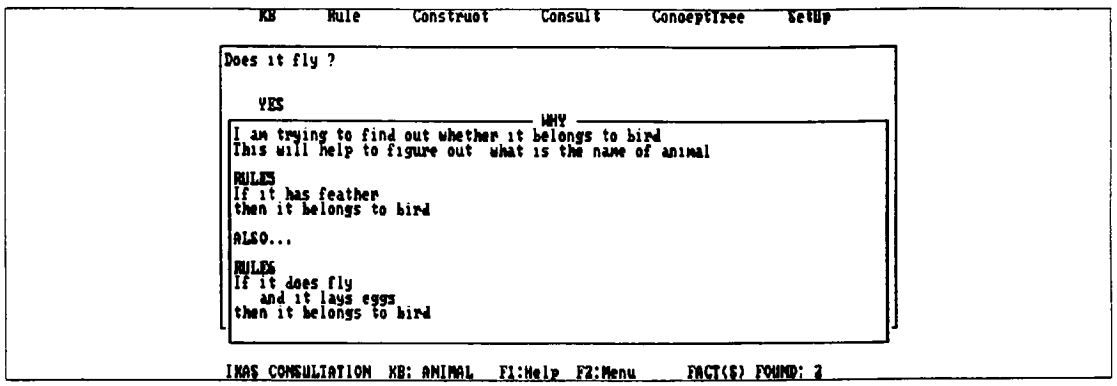

Figure 5-9

answer for "does it fly ?" is "NO." After learning the hypothesis "some bird can not fly" the IKAS inference engine avoids knowledge deadlock as the following manner.

(1) The IKAS inference engine tries rule 6 to determine the fact "bird is yes"

(2) As soon as the response for "does it fly ?" is known to be "NO", as shown in Figure 5-9, the first condition in rule 6, which is "does_fly is yes," immediately failed.

(3) Instead of ignoring the rest of the condition "lay_eggs is yes" and making the whole rule 6 fail, the IKAS inference engine investigates potential ways to make an exception from a normal bird such that even if it does not fly there is a remote possibility of it being a bird. When the inference engine finds the learned hypothesis "some bird does not fly," it ignores the failure of the first condition "does_fly is yes" and continues to explore the possibility for its being a bird by testing the condition "lay_eggs is yes," as shown in Figure 5-10.

(4) After answering "YES" for the questions in Figure 5-10, the inference engine finds the possibility of animal being a bird, and it tries rule 13 for the purpose of determining the animal name to be ostrich, as shown Figure 5-11. The answer is given

as "NO" when asked "Has it long neck ?" and then the inference engine tries rule 14 for determining if the animal name is penguin by asking "Does it swim ?" and the rest of questions shown in Figures 5-12 and 5-13.

(5) Figures 5-14 and 5-15 show the conclusion screen with the listing of facts concluded and how it concluded name of animal as penguin.

The important point to be made here is the flow of utilizing the concept. At first, the IKAS inference engine tries to test a more precise concept to determine the fact. If the fact becomes true by testing a precise concept then the fact is determined, but if not, a more general concept, which is an inductive hypothesis, will be searched and examined to remedy the possible over-specialization or knowledge deadlock.

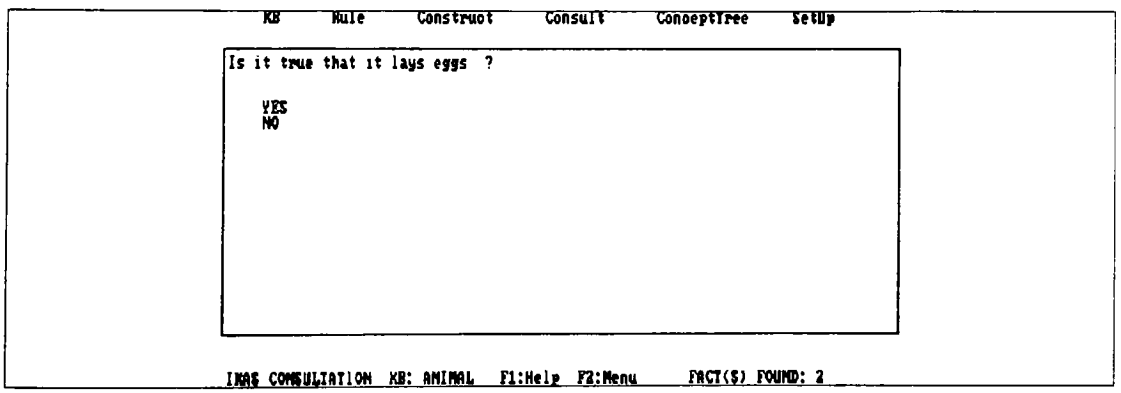

Figure 5-10

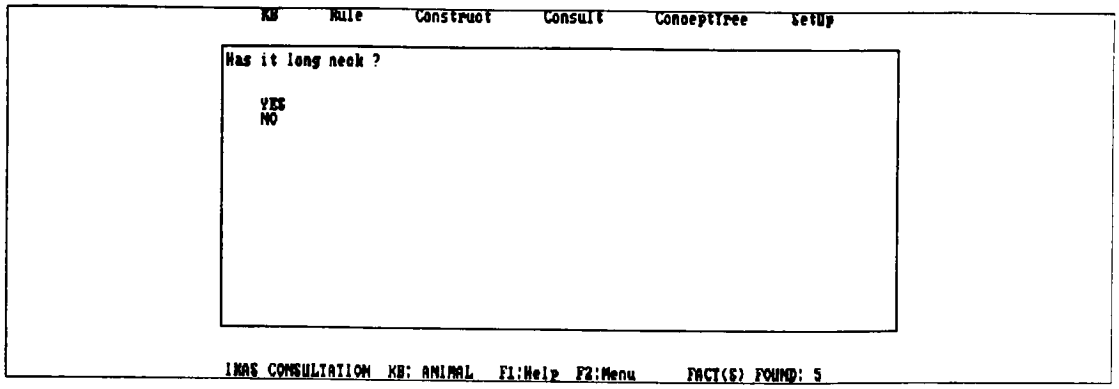

Figure 5-11

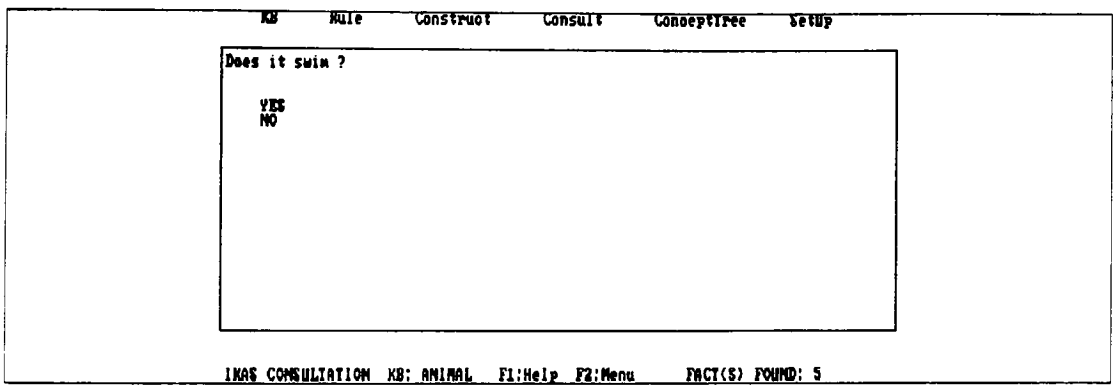

Figure 5-12

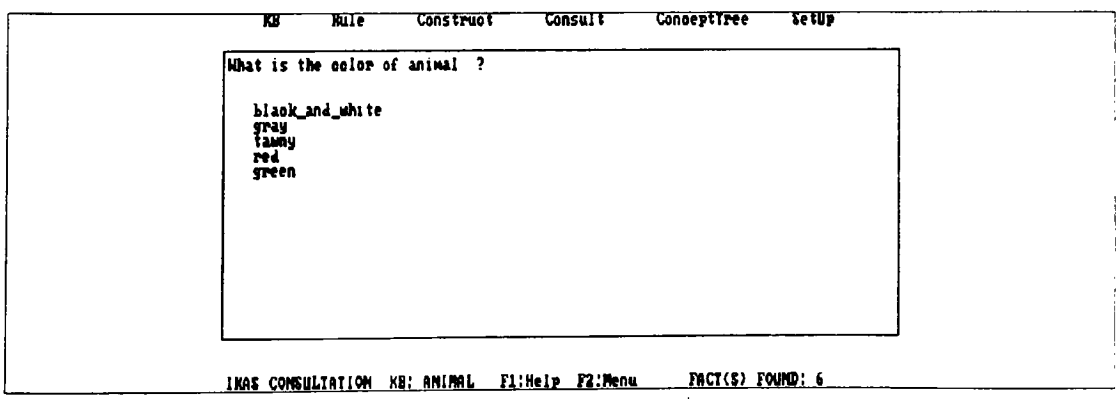

Figure 5-13

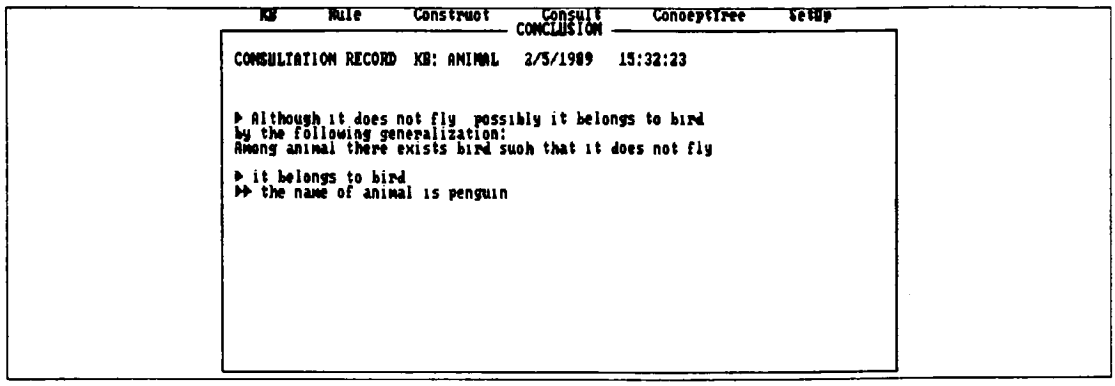

Figure 5-14

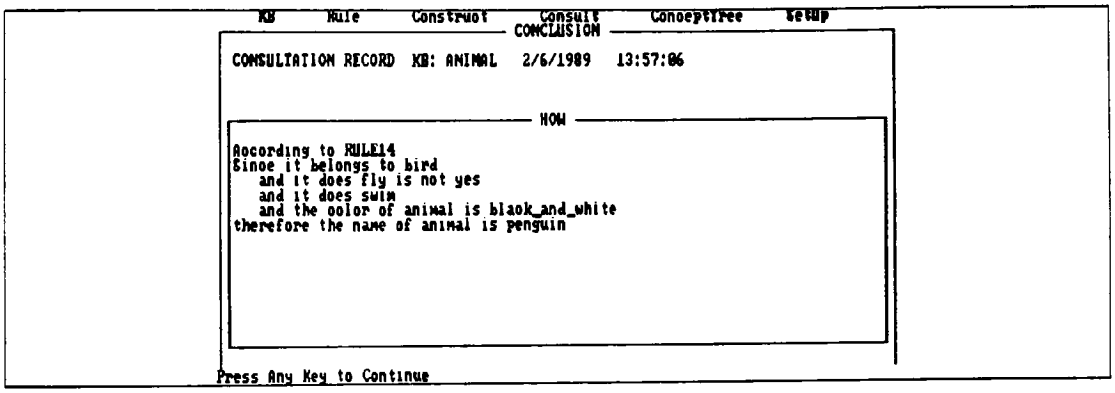

Figure 5-15

5.5. Conclusion

IKAS is a rule based expert system shell with an intelligent knowledge acquisition capability. In this chapter, most of the emphasis was placed on the Conflict Checker / Solver / Generalization Unit since it is the theme of this thesis. The rest of components will be explained briefly in Chapter 6 along with the sample program run. The most important notion developed in this chapter includes the recursive concept hierarchy with a feature list and the way to represent each feature in terms of an "and_agenda" and "or_agenda." Also automatic generalized concept forming using the IEG method gives IKAS a capability to discover new inductive hypotheses from the existing knowledge base. Even though there is no exact indication that human learning uses the same technique to detect knowledge conflicts and learn new concepts, I believe these techniques can be valuable tools for machine comprehension because of the machine's limited ability to represent complex human knowledge.

Chapter 6. IKAS The Expert System Shell

Although throughout the previous chapters, particular attention was placed on the Conflict Checker / Solver / Generalization Unit in IKAS, it is necessary to demonstrate at least briefly the other basic functionalities and features of IKAS as an expert system shell, even though it is not the main topic of this thesis. The major functions are grouped as units according to their roles and explained in this chapter with the help of screen dump of IKAS in execution. Section 6.5 includes a step-by-step demonstration of knowledge base creation using IKAS.

6.1. KB File Handling Unit

The knowledge base (KB) File Handling Unit deals with knowledge base creating, retrieving and storing to and from a file. When prompted to type in the name of a file, hitting return, in many cases, produces a listing of available files to choose from. Here are some of basic functions the KB File Handling Unit manages as depicted in Figure 6-1.

(1) Load : Load the existing knowledge base.

(2) Create : Create a new knowledge base.

(3) Save : Save the current knowledge base to a file so that it can be consulted and edited later.

(4) Delete : Delete an existing knowledge base in the directory.

(6) Translate : Translate All rules in the current knowledge base to English.

(7) Print : Print the current knowledge base to either file or printer

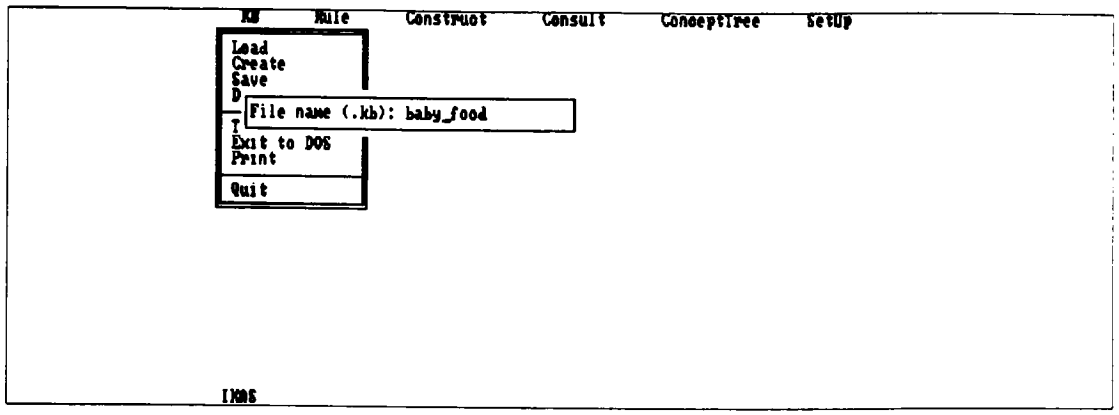

Figure 6-1

6.2. ARL Edit / Compiler / Construct Unit

IKAS uses its own ARL (Abbreviated Rule Language), which is explained in detail in Appendix A of this thesis. After the ARL is entered through the editor, as shown in Figure 6-2, the Compiler Unit automatically checks for possible syntax

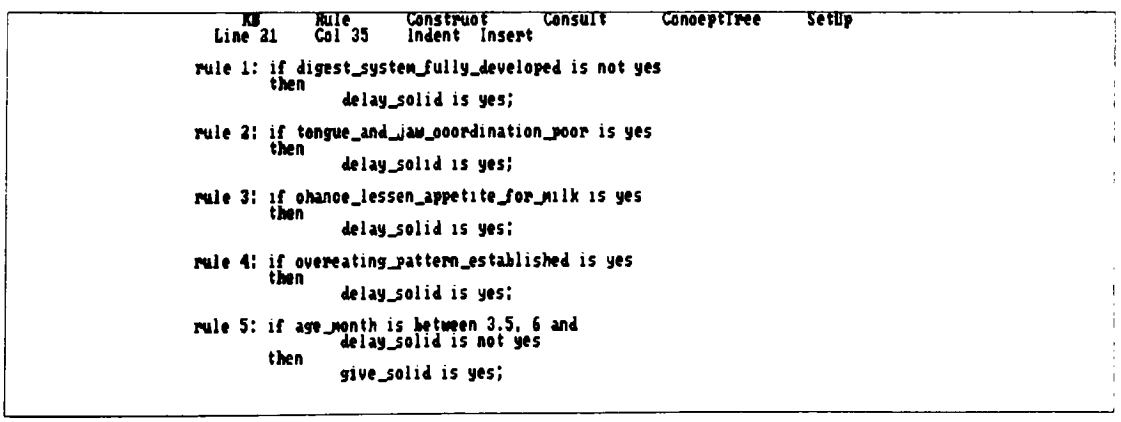

Figure 6-2

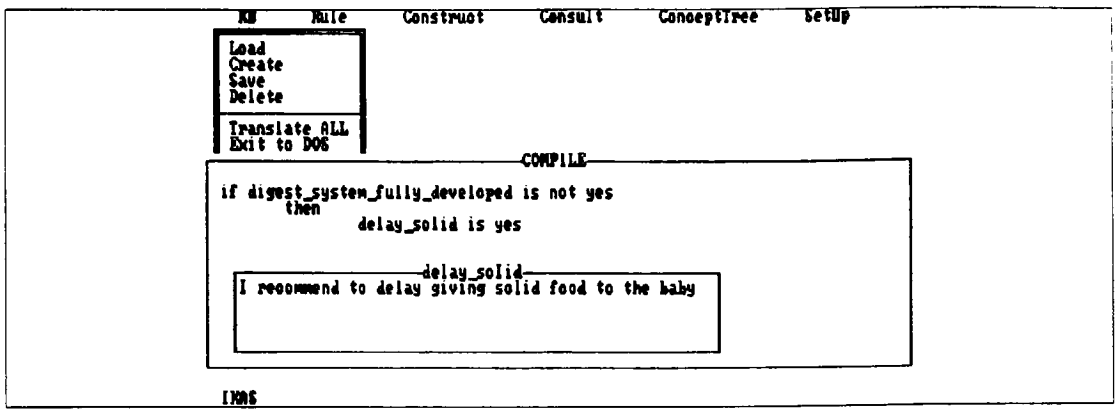

Figure 6-3

errors. After syntax checking is done, if the objects (please refer to Section 5.2.2) from the new rule(s) were not introduced previously, the translation and type of the object will be asked of the user as illustrated in Figure 6-3 and Figure 6-4. The translation value for the object tells how to express the object in plain English, along with the type of object it uses to construct sentences when the IKAS inference engine asks certain questions and gives recommendations during consultation. For this reason, the translation values that contain verbs such as have, does, can, could, shall, and so on is very handy for constructing certain sentences. As an example, some of the possible translation values for the "does_fly" can be "it does fly" or "it flies." When the question is asked, the sentence "Does it fly ?" can be formed from the translation value "it does fly," and the sentence "Is it true it flies ?" can be formed from the translation value "it flies." As an related example, if the rule says "does_fly is not yes," the translation unit translates it as "it does not fly" by inserting "not" after the verb "does."

The type of object is characterized by the following considerations:

- Single : There is only one value for this object.
- Multiple : There can be more than one value for this object.
- Yes/No : The value for this object is either yes or no (boolean).
- Number : The value for this object is number (integer, real).

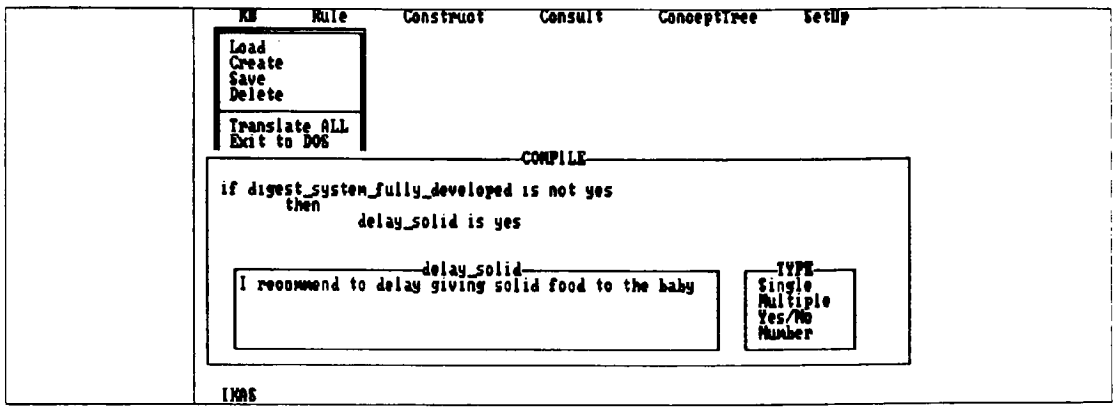

Figure 6-4

According to the type of object, the inference engine works differently. If the type is single, number or yes/no, as soon as inference engine finds a value for the object, tracing is stopped even though there are ways to figure out the other values for the object. For the multiple type of object, the inference engine tries to determine as many values as possible for the object even after the first value is determined. If the intent of the KB is to determine as many conceivable solutions or recommendations as possible from the given facts, then defining objects as multiple type objects is recommended. The object "recommendation" in the Diagnosis KB in Appendix B, which asks all the imaginable software/hardware defects and gives remedial solutions, is an example that illustrates the power of a multiple type object. The Construct Menu of IKAS handles the following functions as depicted as Figure 6-5:

(1) Object Translation : Change the translation of a parameter already introduced.

(2) Object Type : Change the type of a parameter already introduced.

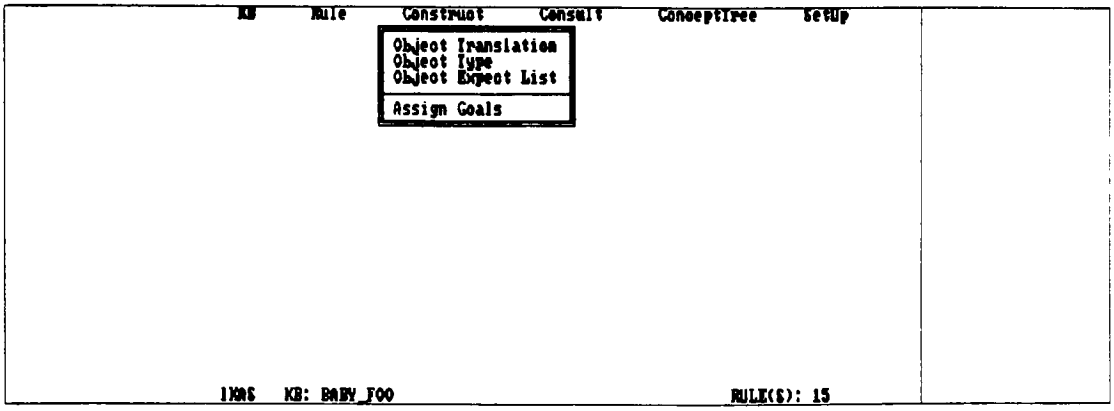

Figure 6-5

(3) Object Expect List : The Expect List is used during Consultation. The User can assign the choices to choose from for the object if applicable, as shown in Figure 6-6. An example of this would be: for the object color, the user may have the expect list : red, yellow and red_and_white. When asked "What is the color ?" during a consultation, the user would be given a choice of these three choices.

(4) Assign Goal(s) : The user can assign the goal(s) for the current knowledge base, as shown in Figure 6-7. The Goal is the name of the object the user wants to find out during the consultation. The IKAS Consultation Unit has the ability to summarize the supplementary facts found during tracing the goal(s); therefore, it is recommended that the user enter only the main goal(s).

6.3. Inference Engine / Consultation Unit

The IKAS inference engine is a goal-oriented, backward chaining facility. From the rules and goal(s) encoded in the KB, the inference engine figures out what rules to try in what sequence. Also, using the translation unit, the inference engine asks questions in the course of determining the value of a goal, and after concluding the

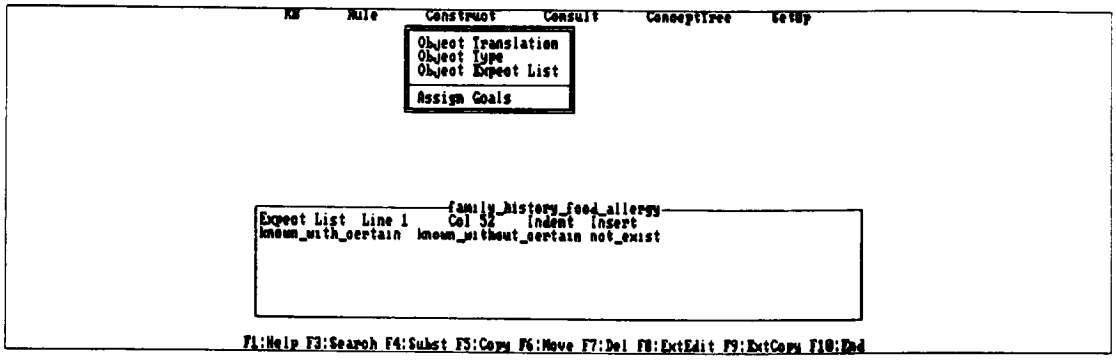

Figure 6-6

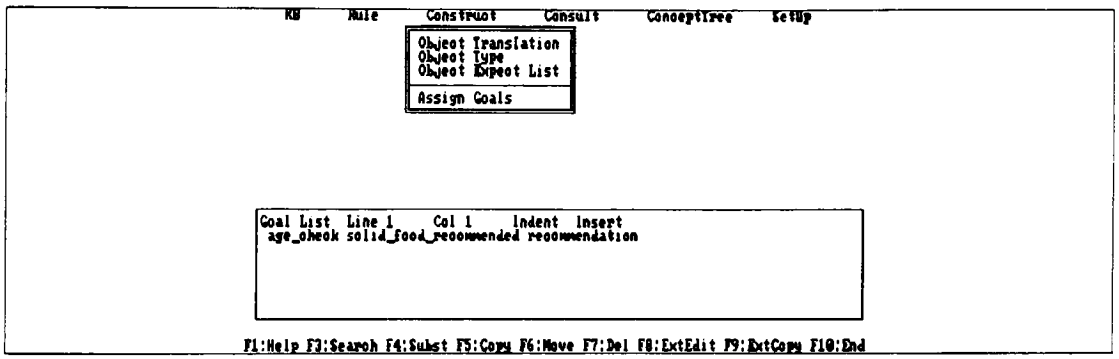

Figure 6-7

goal, it explains what value of the object has been determined and how it was determined. The Consult menu option in main menu executes the expert system. it is essential for user to enter the goal list before the consultation.

6.3.1. Commands During a Consultation

While executing an expert system, the user has the ability to ask "WHY" a question is being asked, to examine the facts that were already determined and to ask

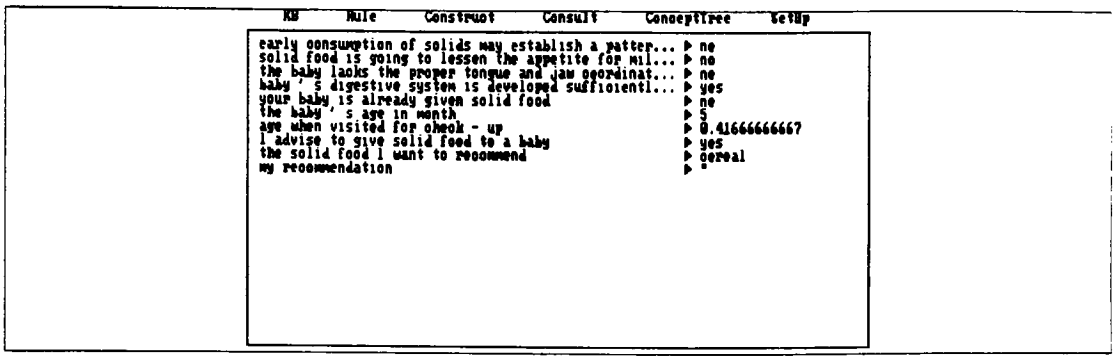

Figure 6-8

"HOW" conclusions were arrived at. To do this, press the F2 key and choose either the WHY or the HOW option. The user can move to the value that the user wishes to examine as shown in Figure 6-8. Hitting return will display either "how" the value was determined or "why" a question is being asked, as shown in Figure 6-9.

6.3.2. Commands After a Consultation

After executing an expert system, the user will be placed in a display window showing the file containing the conclusions that were determined as shown in Figure 6-10. This is the Consultation Record file and user can quit by typing ESC or F10 key. There will now be another menu displayed as follows:

- How : This option is the same as the F2 key and the "how" selection mentioned above. Just choose this option and choose the value the user wishes to look at.

- What If : This allows user to go back and change a value or values and see how these changes affect the final results. This is similar to the whatif concept in spreadsheet applications. Choose a value or values the user wishes to change with RETURN key, and press F10 to start the new consultation.

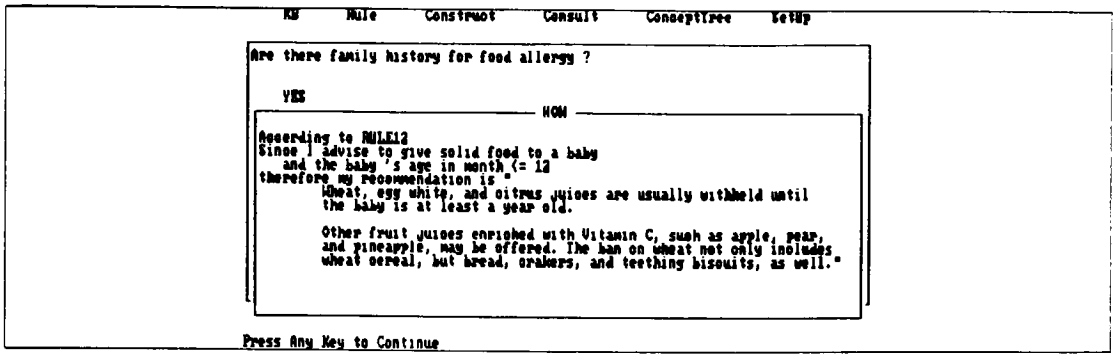

Figure 6-9

- Restart : Allows user to start the consultation over from the beginning.

- Save : Save the consultation file with .rec extension.

- Append : Append the consultation file to another file. This is useful when records for a particular user should be stored together. Let's say client A visits a bank and has a consultation about life insurance. Next time client A has a consultation about C.D.'s. The bank may wish to append the results of this consultation to A's previous consultation. Again the user will be asked for a filename, and .rec is the default extension.

- Quit : Quit the consultation menu and return to the Main menu.

6.4. Conflict Checker / Solver / Generalization Unit

This unit is accomplished by selecting ConceptTree menu, which examines the relationships between objects. The user can enter or modify the rule(s) about the chosen object. If there exists a conceptual conflict or duplication with regards to the

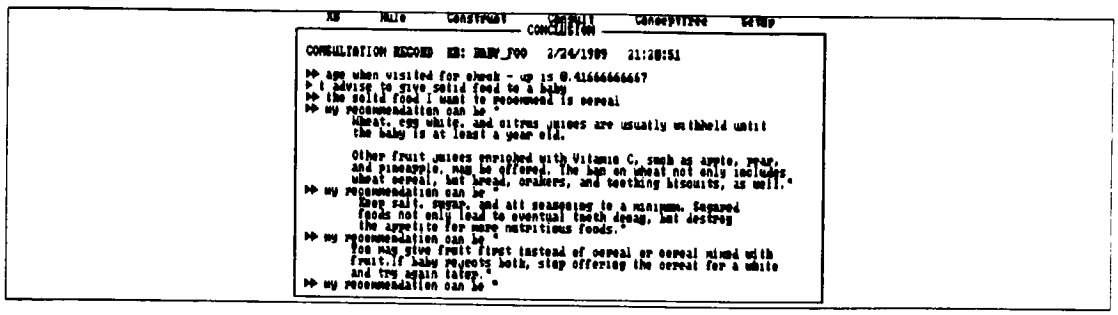

Figure 6-10

chosen object, the remedial step is taken, as demonstrated in detail in Chapter 5. If no conceptual fault is detected, then the concept hierarchy is displayed on the screen. The user can select either "influence by" mode or "influence to" mode by setting the environment as shown in Figure 6-11. In "influence by" mode, the parent node of each tree is determined by the child nodes. This is a top-down (goal driven) knowledge representation and shown in Figures 5-4 and 5-7. In "influence to" mode, the parent node of each tree influences the value of the child nodes. This is a bottom-up (data driven) knowledge representation and shown in Figure 6-12. To enter the editor via a node of a tree, simply move the cursor with the arrow keys to highlight the object the user wishes to enter a rule about and press F3. If "return" is pressed when an object is highlighted, the tree root is changed to that object. Please note that the user also can translate the rules to English by pressing F2 here. Notice that only the rules that determine the highlighted object appear in the editor or translation when F2 or F3 are pressed. This method of entering the editor eliminates the need to search through the rulebase to find the rules that user is interested in. The ConceptTree also can serve as a very useful debugging aid. If the user can determine the area of the tree that a bug is in, the user will have a means to display only the rules that determine each object. It also displays how each object affects and is affected by the others so logic can be checked. The left and right arrow keys are set up to jump the cursor across several

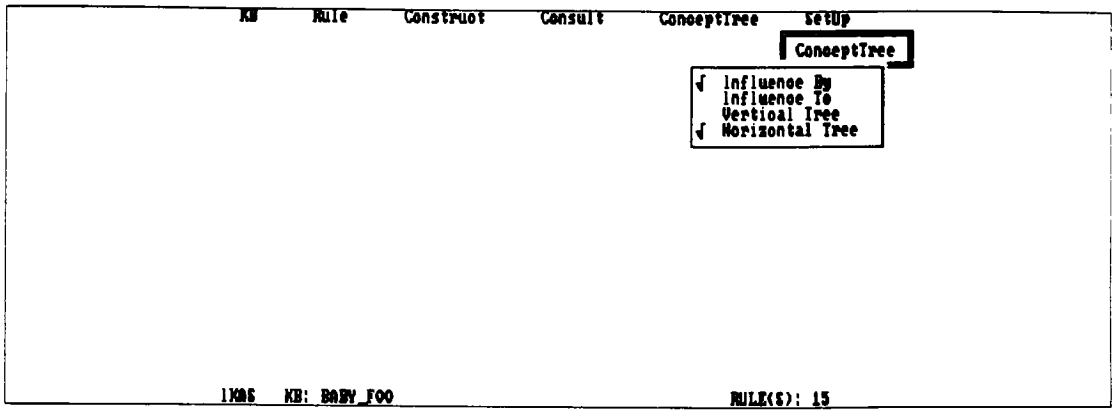

Figure 6-11

spaces. If it should become necessary to move just one space at a time, please use Ctrl in combination with the arrow keys.

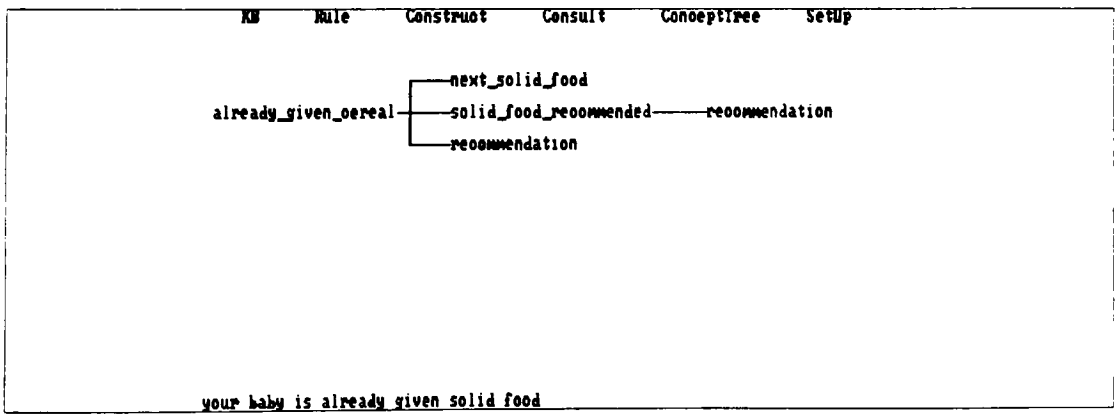

Figure 6-12

6.5. Step by Step IKAS Knowledge Base Creation

This section explains how to create a KB.

(1) First select "Create" from the "KB" menu.

(2) Enter a filename as animal.kb , with or without an extension (.KB)

(3) When the editor comes out, type following rules (NOTE: This is a full editor. Press F1 for a list of all commands available)

```
rule 1: if mammal is yes and
carnivore is yes and
color is tawny and
dark_spot is yes
then
animal is cheetah;

rule 2: if has_hair is yes or
give_milk is yes
then
mammal is yes;
```

(4) You can press ESC or F10 to exit the rule editor.

(5) The Compile window will pop up showing the rule you just entered and will ask the translation and the type of any object that was used in this rule but was not previously used in the rulebase. Since the KB did not previously exist, the translation and type for all parameters will be asked. When asked the translation of the object, type the translation and press ENTER. When asked the type of the object, use the up/down arrow and RETURN key to move and select the type from Single, Multiple, Yes/No and Number.

Please enter the following for each object when it is asked:

Object	Translation	Type
mammal	animal belongs to the mammal family	yes/no
carnivore	animal belongs to the carnivore family	yes/no
color	the color of the animal	single
dark_spot	it has dark spots	yes/no
animal	the name of the animal	single
has_hair	it has hair	yes/no
give_milk	it does give milk to her offspring	yes/no

Note: if you made a mistake in naming the object in the rule editor and IKAS asks the translation and type of that object, you can press ESC to skip, and the system will go to the next object. Later you can correct the name when you get in the rule editor.

(6) When entry is all done, go to Construct menu, select Assign Goal, and type: "animal" in the box provided. Press F10 or ESC key after entering the goal (animal). Now you have made a knowledge base that contains two rules, and you assigned the object "animal" as a goal object so that system will try to determine the name of an animal. When you want to edit the rules relating only to certain object, you can use the ConceptTree editor, and at the same time the concept / duplication checker can be used to keep your knowledge base conceptually correct. Selecting "Consult" from the main menu will execute the knowledge base.

6.6. System Performance

IKAS is tested on the following configuration:

- Hardware: IBM-AT 286, IBM-PC
- Operating System: MS-DOS 3.0, MS-DOS 2.1
- Base Memory: 640K
- Display: EGA, CGA, Monochrome Monitor

The performance of IKAS is summarized in Table 6-1.

KB Name (See Appendix B)	Animal	Computer Diagnosis Help Desk
Number of Objects	22	15
**Number of Rules	15	17
*Compile Time	3 seconds	4 seconds
*Time between Questions in Execution	1 second	1 second
*Concept Checking Time from Goal	2 seconds	1 second

Table 6-1 IKAS Performance

* Measured in 12MHz Machine

**Maximum Number of Rule Entry Tested in 640K Base Memory: 50

Chapter 7 Conclusion

I have shown a way to represent knowledge with hierarchical and related concepts, where a concept can be used to construct other, more complicated concepts. I defined three heuristic steps for concept learning in IKAS. The first step is to find refuted concepts by detecting conflict or duplication from the goal-oriented concept tree, where a concept is expressed by conjunctions and disjunctions of the feature lists, and each of these feature lists is represented by and_agenda and or_agenda.

The second step is to generalize the inconsistencies among conflicting concepts by using an Inductive Extension Generalization method, where a conflicting feature can be found and identified as a higer level feature and lower level feature. Enlarging the permissible description space of higher level feature by creating exceptions or negations of higer level features leads to the automatic generalization of the conflicting concept.

The third step is to use these learned concepts to avoid detecting similar conflicts and preventing possible knowledge deadlock. During IKAS consultation, the sequence of utilizing a special concept prior to the learned general concept amplifies the positive description space for determining the fact. This framework suggests a very practical approach in machine learning if we can resolve the following issues of applying machine learning to expert system technology.

7.1. Issues

- Most intelligent learning is the combined means of top-down and bottom-up processing. IKAS's method of learning can be described as only top-down learning, since conflict checking is done from the root of the concept tree by examining the feature list for each concept. This suggests a possible dilemma for defining true

constructive learning in most of the rule based and goal oriented expert systems. Each rule is divided into a premise and action part, and a certain arrangement is already made for the conclusion, which can be the objective of the constructive learning. Therefore, without implementing a discovery learning module, it will be only selective learning and will heavily rely on the occurrence of conflicts. The discovery learning module could be implemented such that it only receives the new feature lists (examples) without the conclusion (action) part, and then predicts the new conclusion(s) or new hypothesis using the feature lists and background knowledge. If this new conclusion is not accurate after asking the human expert, then a remedial step could be taken, including generalization of the background knowledge. I believe that by combining a new discovery module with the IEG methods described in this thesis, more factual machine learning will result.

- The description language implemented the machine learning in expert system technology is very important. Most current expert system shells employ a rule language that consists of a few operands and operators such as "is" and "is not". This type of description language tends to inhibit dealing with real world learning situations. On the other hand, the growing number of operands and operators in a description language causes the problem of an enormous search space unless special provisions are made such as defining synonyms.

- The description space for learning can be restricted to only symbolic descriptions. When checking for conflict or duplication from a mathematical relational operator such as ">", "=", "< =", it is required to wait until run time for all the operands to get their values, this is known as binding time issue. For example if we have a rule : if a + b / c = d * 0.52 , we need the value of a,b,c and d to compare the mathematical meaning of this rule to the other rules.

• In the case of reporting the conceptually conflicting rules and confirming possible generalizations, more precise and natural sentences with easily understood logic are needed. When a human finds incorrect generalizations, there could be intelligent ways to correct or teach correct concepts using natural language processing input.

• Most of the expert system projects I have personally been involved in require miscellaneous routines beyond those for the knowledge base building. For example, the interface to graphics display routines for pinpointing a troubled area in a diagnosis systems, transferring numerical data to and from a spreadsheet for separating number crunching from the symbolic processing, and storing and retrieving data in a database are often required. Current expert system shells either allow the knowledge engineer to program user defined functions to deal with such assorted functions, or they have rule languages specially designed for such tasks. In either case, if a learning strategy is essential to maintain the whole expert system as time goes by, the question arises of how to preserve such functions or rules.

• Probably the final objective of applying machine learning techniques to expert system technology is to eliminate the role of the knowledge engineer. Currently the knowledge engineering process involving the interviewing of the expert for formalizing the expert's logic occupies a good percentage of the time spent for building an expert system. The idea behind the shells that use induction tables, learning by discovery, conflict / duplication concept checking and generalization is to allow expert(s) to build their own expert systems without going through the tedious interviewing process with a knowledge engineer. I have felt the controversy in this human-computer social relationship. Of course the following claims are solely based on the experience I have had, and are purely driven by social factors rather than available technologies. First of all, there are not as many experts who feel comfortable with the so called "innovative computer technology" as we might expect. For these people, inputing the data in an

induction table or writing the most simplified rule base, even if it is equipped with intelligent generalization unit, is a big hurdle because some of them are not accustomed to even pressing a control key combination on a computer keyboard. Unless complete artificial intelligence technology, including voice recognition, machine vision, natural language processing and machine learning is implemented such that the expert can teach the program by talking natural language or showing data or pictures, this problem cannot be solved completely. Second, most experts are very busy with other work; it is even hard to interview those experts once a week because of their busy schedules. It is out of the question to expect these experts to spend a considerable amount of time engaging with the computer and building expert systems by themselves.

7.2. Future Work Remaining

- As mentioned in the last section, the learning by discovery module, along with the current IEG method would enhance the learning capabilities of IKAS by combining top-down and bottom-up processing. For example, after a skeleton of background knowledge is stacked, we can input the list of features. Then the learning by discovery module will examine these features with the related background knowledge, and generalized hypotheses, and it will come up with new derived rules.

- As careful readers might have noticed, this thesis is loaded with logic. There may be better approach to find conflicting concepts with complexity and making generalizations using mathematical logic. Also there are other generalization methods that could be implemented through IKAS's Concept Base other than Condition Dropping Rules and Inductive Extension Rules as mentioned in this thesis. Chapter 4: Inductive Learning shows a list of generalization techniques. Different approches, which will result in more mathematical and heuristic machine learning methods, are welcomed and could be a good thesis topic for other students.

• More description operators such as "is" are recommended for representing complex knowledge. Among them are "has" , "has_part_of", "belongs to" and so on. The best method can use natural language sentences and interpret them as description operators and operands. In addition to the learning part, this will enhance the expert system consultation. For example, instead of answering questions constructed from the inference engine, the end user has the control to ask questions such as "Is a penguin a bird ?","give me all the features of a tiger."

• After an IKAS consultation, the determinations made using rules encoded could be incorrect. Rather than go back to the rule editor and change rules manually, there could be a new mechanism to update or generalize the rule from the conclusion screen by pointing to what determinations are incorrect. Consequently from user input, the rule(s) that caused incorrect determinations could identified and used for learning by mistake. This learning by mistake, along with learning by discovery mentioned above and learning by generalization, will lead to a more convincing approach to human learning.

In conclusion, the primary objective of this thesis was to practically apply machine learning techniques, especially in the area of machine assisted refinement of a knowledge base during expert system development, to acquire descriptions containing conceptual errors that ultimately lead to making new generalizations. The additional explanations and needs of constructive inductive learning in expert systems were covered only superficially. There is no doubt that as artificial intelligence research evolves, learning theory will remain a core concept in machine intelligence. Still, the most significant research in machine learning is toward understanding how humans comprehend when there is no exact answer for a question. We can only make a guess at how human learning is done and only predict a heuristic future for machine learning.

Appendix A. IKAS ARL Instructions

Rules should have following format:

rule RULENUMBER : PREMISE then ACTION ;

* RULENUMBER should be an integer value to distinguish the rules from each other.
* 'rule' ':' and ';' are needed to avoid format errors.

1.1.PREMISE

Premise can consist of conditions and logical operators to combine the conditions. Conditions can consist of clauses and/or numerical comparison operators.

1.1.1.and

TYPE: Logical Operator
USAGE: condition1 and condition2 [and condition3..]
DESCRIPTION:
Condition on both sides of 'and' should be true for the if's part of the rule to succeed.

EXAMPLE:

```
rule 1: if triangle is yes and
        a = 1 and b = 1 and c = 1 then d = 5
        else d = 6;
```

TYPE: Logical Operator
USAGE: condition1 or condition2 [or condition3..]
DESCRIPTION:
One of the conditions should be true for the if's part of the rule to succeed.

EXAMPLE:

```
rule 1: if a = 1 or b = 2 or c = 3 then d = 7;
```

1.1.2.is

TYPE: CLAUSE
USAGE: Parameter_Name is Value1
DESCRIPTION:
Test whether Parameter_Name's value is Value1.
case 1) If the parameter has been determined to be Value1 the clause succeeds.
case 2) If the parameter has been determined to be value other than Value 1 then the clause fails.
case 3) If the parameter has not been determined then the system starts to trace the parameter and
applies case 1 or case 2 given above.

EXAMPLE:

```
rule 1: if color is white and shape is box and
          has_keyboard is yes
        then object is computer;
```

1.1.3.is not

TYPE: CLAUSE
USAGE: Parameter_Name is not Value1
DESCRIPTION:
Test whether Parameter_Name's value is not Value1.
case 1) If the parameter has been determined to be Value1 the clause fails.
case 2) If the parameter has been determined to be a value other than Value 1 then the clause
succeeds.
case 3) If the parameter has not been determined then the system start to trace the parameter and
applies case 1 or case 2 given above.

EXAMPLE:

```
rule 1: if whether is not cloudy and
          day is Sunday then
```

go_picnic is yes;

1.1.4.is known

TYPE: CLAUSE
USAGE: Parameter_Name is known
DESCRIPTION:
Test whether Parameter_Name is determined. If parameter has been determined then clause succeeds
and if not determined it fails without tracing the parameter.

EXAMPLE:
rule 1: if tax_number is known and address is notknown
then return_application is yes;

1.1.5.is notknown

TYPE: CLAUSE
USAGE: Parameter_Name is notknown
DESCRIPTION:
Test whether Parameter_Name is not determined. If parameter has been determined then the clause
fails without tracing the parameter and if not determined it succeeds.

EXAMPLE:
rule 1: if tax_number is known and address is notknown
then return_application is yes;

1.1.6. = , < > , > , < , > = , < =

TYPE: Numerical Comparison Operator
USAGE: numerical statement :
numerical value :
Parameter_Name

Numerical Comparison Operator

numerical statement :
numerical value :
Parameter_Name
DESCRIPTION:
> Greater than
> = Greater than or Equal to
< Less than
> = Less than or Equal to
= Equal to
< > Not Equal to

EXAMPLE1
rule 1: if a + b + c d + 1 then e = 5;
EXAMPLE2
rule 1: if a 1 then e = 2;
EXAMPLE3
rule 1: if a c then e = 6;

1.1.7.is between Numeric_Value1 , Numeric_Value2

TYPE: Numerical Comparison Operator
USAGE: Parameter_Name is between Numeric_Value1 , Numeric_Value2
DESCRIPTION:
Parameter_Name's value is greater than or equal to
Numeric_Value1 and less than Numeric_Value2.
EXAMPLE:
rule 1: if age is between 1 , 10 then age_group is child;

TYPE: Numerical Operator
USAGE: Same as Mathematical Usage
DESCRIPTION:
+ Plus
- Minus
* Times
/ Divide

1.2.ACTION

If the PREMISE part succeeds then execute the ACTION part. .

TYPE: chaining operator
USAGE: statement1 and statement2 [and statement3..]
DESCRIPTION:
Execute statement1 then statement2 ... etc.

EXAMPLE:
rule 1: if a 1 then b = 1 and c = 2 and d = 5;

1.2.1.is

TYPE: Statement
USAGE: Parameter_Name is Non_Numeric_Value
DESCRIPTION:
Parameter is determined to be a Non_Numeric_Value.
EXAMPLE:
rule 1: if fuel_filter_bad is yes then
recommendation is "replace the fuel filter";

1.2.2. =

TYPE: Statement
USAGE: Parameter_Name =
numerical statement :
numerical value :
Parameter_Name
DESCRIPTION:
Parameter is assigned to be a numerical statement or a numerical value or the value of another parameter.

EXAMPLE1:
rule 1: if a 1 then c = a + 1;
EXAMPLE2:
rule 1: if a = 1 then c = 2;
EXAMPLE3:
rule 1: if a is between 2,3 then
c = a;

1.2.3.system

TYPE: Statement
USAGE: system [" Dos Command "]
DESCRIPTION:
Execute the Dos Command from IKAS. Dos Commands can be exactly the same as they are when at the DOS
command line prompt.
EXAMPLE:
rule 1: if date is notknown then
system "date";

1.2.4.print

TYPE: Statement
USAGE: print [" String "]
DESCRIPTION:
Print the String in the full window. This statement can be used when displaying the introduction
screen, or for pop-up screens when it is necessary to display a semi conclusion/note to the end user.

EXAMPLE:
rule 1: if carburetor_status is flooded
then print " Note: Hold the gas pedal to the floor and try starting ";

Appendix B. Example IKAS Knowledge Base

1.1.Animal

IKAS Knowledge Base Source : KB :ANIMAL 1/24/1989 13:32:42

PARAMETER LISTING

= = = =
bird
= = = =
TYPE: Yes/No
TRANSLATION: animal belongs to bird
= = = = = =
animal
= = = = = =
TYPE: Single
TRANSLATION: the name of animal
= = = = = = = =
ungulate
= = = = = = = =
TYPE: Yes/No
TRANSLATION: animal belongs to ungulate
= = = = = = = = =
carnivore
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal belongs to carnivore
= = = = = =
mammal
= = = = = =
TYPE: Yes/No
TRANSLATION: animal belongs to mammal
= = = = = = = = =
does_swim

= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal does swim
= = = = = = = = =
does_fly
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal does fly
= = = = = = = = = = = = = = =
has_long_neck
= = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has long neck
= = = = = = = = = = = = = =
has_long_leg
= = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has long leg
= = = = = = = = =
chew_cud
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal chew cud
= = = = = = = = = = =
has_hooves
= = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has hooves
= = = = = = = = = = = = = = =
pointed_teeth
= = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has pointeed teeth
= = = = = = = = =
has_claws
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has claws

= = = = = = = = = = = = = = = = =
has_forward_eyes
= = = = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has forward eyes
= = = = = = = = =
eat_meat
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal eats meat
= = = = = = = = =
lay_eggs
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal lays eggs
= = = = = = = = = = = =
has_feather
= = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has feather
= = = = = = = = =
give_milk
= = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal gives milk to her offsprings
= = = = = = = =
has_hair
= = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has hair
= = = = = = = = = = = = = =
black_stripes
= = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has black stripes
= = = = =
color
= = = = =

TYPE: Single
TRANSLATION: the color of animal
= = = = = = = = = = = = = = =
has_dark_spots
= = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: animal has dark spots

RULE LISTING

rule 8: if pointed_teeth is yes and
has_claws is yes and
has_forward_eyes is yes
then
carnivore is yes;
rule 1: if mammal is yes and
carnivore is yes and
color is tawny and
has_dark_spots is yes
then
animal is cheetah;
rule 2: if mammal is yes and
carnivore is yes and
color is tawny and
black_stripes is yes
then
animal is tiger;
rule 3: if has_hair is yes
then
mammal is yes;
rule 12: if ungulate is yes and
black_stripes is yes
then
animal is zebra;
rule 13: if bird is yes and
does_fly is not yes and
has_long_neck is yes and
has_long_leg is yes and

color is black_and_white
then
animal is ostrich;
rule 4: if give_milk is yes then
mammal is yes;
rule 11: if ungulate is yes and
has_long_neck is yes and
has_long_leg is yes and
has_dark_spots is yes
then
animal is giraffe;
rule 7: if eat_meat is yes
then
carnivore is yes;
rule 9: if mammal is yes and
has_hooves is yes
then
ungulate is yes;
rule 10: if mammal is yes and
chew_cud is yes
then
ungulate is yes;
rule 14: if bird is yes and
does_fly is not yes and
does_swim is yes and
color is black_and_white
then
animal is penguin;
rule 15: if bird is yes and
does_fly is yes
then
animal is albatross;
rule 5: if has_feather is yes
then bird is yes;
rule 6: if does_fly is yes and
lay_eggs is yes
then
bird is yes;

1.2.Solid Food Advisor

IKAS Knowledge Base Source : KB :BABY_FOO 1/24/1989 13:31:16

PARAMETER LISTING

= = = = = = = = = = = = = = =

recommendation

= = = = = = = = = = = = = = =

TYPE: Multiple

TRANSLATION: my recommendation

= =

solid_food_recommended

= =

TYPE: Single

TRANSLATION: the solid food I want to recommend

= =

week_after_given_cereal

= =

TYPE: Number

TRANSLATION: the number of weeks passed after the first cereal is given as a solid food

= = = = = = = = = = =

give_solid

= = = = = = = = = = =

TYPE: Yes/No

TRANSLATION: I advise to give solid food to a baby

= = = = = = = = = = =

delay_solid

= = = = = = = = = = =

TYPE: Yes/No

TRANSLATION: I recommend to delay giving solid food to baby
= = = = = = = = =
age_check
= = = = = = = = =
TYPE: Number
TRANSLATION: age when visited for check - up
= = = = = = = = =
age_month
= = = = = = = = =
TYPE: Number
TRANSLATION: the baby ' s age in month
= =
family_history_food_allergy
= =
TYPE: Yes/No
TRANSLATION: there are family history for food allergy
= = = = = = = = = = = = = = = = = = =
already_given_cereal
= = = = = = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: your baby is already given solid food
= = = = = = = = = = = = = = =
next_solid_food
= = = = = = = = = = = = = = =
TYPE: Single
TRANSLATION: the solid food I recommend
= =
chance_lessen_appetite_for_milk
= =
TYPE: Yes/No
TRANSLATION: solid food is going to lessen the appetite for milk and rob the body of essential nutrients
= = = = = = = = = = = = = =
reject_cereal
= = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: baby does reject cereal

= =
overeating_pattern_established
= =
TYPE: Yes/No
TRANSLATION: early consumption of solids may establish a pattern of overeating that will lead to obesity in later life
= =
tongue_and_jaw_coordination_poor
= =
TYPE: Yes/No
TRANSLATION: the baby lacks the proper tongue and jaw coordination to eat from spoon rather than nipple
= =
digest_system_fully_developed
= =
TYPE: Yes/No
TRANSLATION: baby ' s digestive system is developed sufficiently to accept solid food

RULE LISTING

```
rule 15: age_check = age_month / 12;
rule 7: if already_given_cereal is no and
            give_solid is yes
            then
            solid_food_recommended is cereal;
rule 1: if digest_system_fully_developed is not yes
            then delay_solid is yes;
rule 2: if tongue_and_jaw_coordination_poor is yes
            then delay_solid is yes;
rule 3: if chance_lessen_appetite_for_milk is yes
            then delay_solid is yes;
rule 4: if overeating_pattern_established is yes
            then delay_solid is yes;
rule 5: if age_month is between 3.5, 6 and
            delay_solid is not yes
            then give_solid is yes;
rule 9: if already_given_cereal is yes and
```

give_solid is yes and
week_after_given_cereal = 1
then solid_food_recommended is fruit;
rule 11: if already_given_cereal is yes
and give_solid is yes
and week_after_given_cereal is between 5,9
then
solid_food_recommended is meat;
rule 10: if already_given_cereal is yes and
give_solid is yes and
week_after_given_cereal is between 1,5
then
solid_food_recommended is vegetables
and recommendation is " The Vegetables, a chief source of carbohydrates, may be added to the menu about two weeks to a month after fruit. Serve vegetables separately at the midday feeding, following the same rule of introducing one at a time at regular intervals so you can watch for the baby's reaction.
Begin by serving yellow vegetables with a mild flavor - squash or sweet potatoes, for example. The progress to carrots, beans, peas. Beets and spinach may follow. Some vegetables may be unpopular, but you should acquaint your baby with all varieties.";
rule 12: if give_solid is yes and age_month < 12
then recommendation is
" Wheat, egg white, and citrus juices are usually withheld until the baby is at least a year old.
Other fruit juices enriched with Vitamin C, such as apple, pear, and pineapple, may be offered. The ban on wheat not only includes wheat cereal, but bread, crakers, and teething biscuits, as well.";
rule 14: if age_month is between 3.5, 6 and
delay_solid is yes
then recommendation is " I found good reason to delay solid food, therefore do not give solid food at this time. Nutritional studies have shown that normal infants can be well nourished and grow appropriately by consuming a variety of milks, with or without addition of solid food.";
rule 13: if give_solid is yes and
family_history_food_allergy is yes
then recommendation is " Keep salt, sugar, and all seasoning to a minimum. Sugared foods not only lead to eventual tooth decay, but destroy the appetite for more nutritious foods.";

rule 8: if give_solid is yes and
solid_food_recommended is cereal and
reject_cereal is yes then recommendation is " You may give fruit first instead of cereal or cereal mixed with fruit.If baby rejects both, stop offering the cereal for a while and try again later.";
rule 6: if give_solid is yes and
solid_food_recommended is cereal and
age_month < 12 and family_history_food_allergy is yes
then recommendation is " Rice is the mildest and most readily digested cereal. However because of your family food allergy history, I advise NOT to give Wheat and mixed cereals until your baby becomes one year old.";

1.3. Computer Diagnosis Help Desk

IKAS Knowledge Base Source : KB :DIAGNOSI 2/25/1989 12:01:22

KB GOAL ["cause","action","recommendation"]

EXPECT LIST
goal ["install_software","software_diagnosis","selftest"]
print_quality ["good","medium","poor"]
printer ["dot_matrix","laser","laser_jet","plotter"]
user_monitor ["monochrome","CGA","EGA","VGA"]
message ["disk_full","segment_read_error","can_not_read_disk_A","block_read_error"]

PARAMETER LISTING
= = = = = = = = =
sub_goal
= = = = = = = = =
TYPE: Multiple
TRANSLATION: possible test area
= = = = = = = = = = = = = = = = =
recommendation
= = = = = = = = = = = = = = = = =
TYPE: Multiple
TRANSLATION: recommendation
= = = = =
cause
= = = = =
TYPE: Multiple
TRANSLATION: the cause of abnormal operation
= = = = = =
action
= = = = = =
TYPE: Multiple
TRANSLATION: action to take
= = = =
goal
= = = =

TYPE: Single
TRANSLATION: the subject
= = = = = = = = = = = = = = = = = = = =
software_version_old
= = = = = = = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: your software is less than 2.0
= = = = = = = = = = = = =
none_display
= = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: nothing is displayed on the monitor
= = = = = = =
printer
= = = = = = =
TYPE: Single
TRANSLATION: the printer type you have
= = = = = = = = = = = = = =
print_quality
= = = = = = = = = = = = = =
TYPE: Single
TRANSLATION: the print out quality
= = = = = = = = = = = = = = = =
printed_reverse
= = = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: your printer output is reversed color
= = = = = = = = = = = = =
user_monitor
= = = = = = = = = = = = =
TYPE: Single
TRANSLATION: the type of monitor you have
= = = = = = = = = = =
snow_effect
= = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: the program display is flickering

= = = = = = = = = = = = = = = = =
sub_goal_success
= = = = = = = = = = = = = = = = =
TYPE: Yes/No
TRANSLATION: previouse step has been succeeded
= = = =
note
= = = =
TYPE: Single
TRANSLATION: system
= = = = = = =
message
= = = = = = =
TYPE: Single
TRANSLATION: error message

RULE LISTING

rule 10: if sub_goal is video_display and
user_monitor is monochrome and
software_version_old is yes and
none_display is yes
then cause is old_version_software and
recommendation is " Your current monochrome monitor is not compatible wit
software version number less than 2.0. Please call the following number for exchan
for the new software. (777) 234-6789 Attn. Exchange Dept. ";

rule 12: if sub_goal is video_display and
none_display is yes
then cause is wrong_setup and
recommendation is " Please check the system configuration option again durin
software install for the right type of display equipment. If still does not work, call yo
hardware vendor for service.";

rule 16: if cause is wrong_setup then
action is install_again;

```
rule 17: if cause is hardware_exhaust
                then
                action is call_service;

rule 3: if sub_goal is file_transfer and
                sub_goal_success is no and
                message is can_not_read_disk_A
                then
                cause is software_damaged and
                action is refund_software;

rule 1: if sub_goal is file_transfer and
                sub_goal_success is no and
                message is disk_full
                then
                cause is disk_full and
                action is clean_file_space;

rule 2: if sub_goal is file_transfer and
                sub_goal_success is no and
                message is segment_read_error
                then
                cause is disk_driver_fault and
                action is call_service;

rule 5: if sub_goal is file_transfer and
                sub_goal_success is no and
                message is block_read_error
                then
                cause is disk_driver_fault and
                action is call_service;

rule 11: if sub_goal is video_display and
                user_monitor is CGA and
                snow_effect is yes
                then
```

```
        cause is wrong_setup and
        recommendation is "     Please choose CGA ON option when installing software.
        will slow the display speed and your snow effect
        will be disappeared.";

rule 13: if sub_goal is printer_type and
        printed_reverse is yes
        then
        cause is wrong_setup and
        recommendation is "     Your set up program has option to reverse black and whit
        image. Choose Yes for the following question:
          Reverse Black/White color ? (Yes) ";

rule 14: if sub_goal is printer_type and
        printer is dot_matrix and
        print_quality is poor
        then
        cause is wrong_setup and
        recommendation is "     Your set up program has option to select printing quality.
        Choose Quality instead Draft mode will produce fine output.";

rule 15: if sub_goal is printer_type and
        printer is laser and
        print_quality is poor
        then
        cause is hardware_exhaust and
        recommendation is
        "           Check your toner catridge of laser printer.";

rule 6: if goal is software_diagnosis
        then
        sub_goal is video_display;

rule 7: if goal is software_diagnosis
        then
        sub_goal is hard_disk_type;
```

rule 9: if goal is software_diagnosis
then
sub_goal is mouse_type;

rule 8: if goal is software_diagnosis
then
sub_goal is printer_type;

rule 4: if goal is install_software
then
sub_goal is file_transfer and
print
" Disk install program will transfer files from A: to C: driver.";

RULE TRANSLATION

RULE10
If possible test area is video_display and the type of monitor you have is monochrome and your software is less than 2.0 and nothing is displayed on the monitor then the cause of abnormal operation is old_version_software and recommendation is " Your current monochrome monitor is not compatible with our software version number less than 2.0. Please call the following number for exchange for the new software. (777) 234-6789 Attn. Exchange Dept. "

RULE12
If possible test area is video_display and nothing is displayed on the monitor then the cause of abnormal operation is wrong_setup and recommendation is " Please check the system configuration option again during software install for the right type of display equipment. If still does not work, call your hardware vendor for service."

RULE16
If the cause of abnormal operation is wrong_setup then action to take is install_again

RULE17
If the cause of abnormal operation is hardware_exhaust then action to take is call_service

RULE3
If possible test area is file_transfer and previouse step has not been succeeded and error message is can_not_read_disk_A then the cause of abnormal operation is software_damaged and action to take is refund_software

RULE1
If possible test area is file_transfer and previouse step has not been succeeded and error message is disk_full then the cause of abnormal operation is disk_full and action to take is clean_file_space

RULE2
If possible test area is file_transfer and previouse step has not been succeeded and error message is segment_read_error then the cause of abnormal operation is disk_driver_fault and action to take is call_service

RULE5
If possible test area is file_transfer and previouse step has not been succeeded and error message is block_read_error then the cause of abnormal operation is disk_driver_fault and action to take is call_service

RULE11
If possible test area is video_display and the type of monitor you have is CGA and the program display is flickering then the cause of abnormal operation is wrong_setup and recommendation is " Please choose CGA ON option when installing software. This will slow the display speed and your snow effect will be disappeared."

RULE13
If possible test area is printer_type and your printer output is reversed color then the cause of abnormal operation is wrong_setup and recommendation is " Your set up program has option to reverse black and white color image. Choose Yes for the following question: Reverse Black/White color ? (Yes) "

RULE14
If possible test area is printer_type and the printer type you have is dot_matrix and the print out quality is poor then the cause of abnormal operation is wrong_setup and

recommendation is " Your set up program has option to select printing quality. Choose Quality instead Draft mode will produce fine output."
RULE15
If possible test area is printer_type and the printer type you have is laser and the print out quality is poor then the cause of abnormal operation is hardware_exhaust and recommendation is " Check your toner catridge of laser printer."

RULE6 If the subject is software_diagnosis then possible test area is video_display

RULE7 If the subject is software_diagnosis then possible test area is hard_disk_type

RULE9 If the subject is software_diagnosis then possible test area is mouse_type

RULE8 If the subject is software_diagnosis then possible test area is printer_type

RULE4 If the subject is install_software then possible test area is file_transfer and note to the user that " Disk install program will transfer files from A: to C: driver."

Bibliography

1. HLN Thomas H. Leahey, Richard J. Harris, Human Learning, Prentice-Hall, Englewood Cliffs, New Jersey , pp 130-161, 1985.

2. PSY Darley, Glucksberg, Kamin and Kinchla, Psychology Second Edition, Prentice-Hall ,Englewood Cliffs, New Jersey , pp 196-197, 1984.

3. GES Donald A. Waterman, A Guide to Expert Systems, Addison-Wesley Publishing Company, Massachusetts, pp 24-31, 1985.

4.COM Computer World Spotlight NO.31, Machine Learning:The Next Step by Kamran Parsaye, November 23, 1987.

5.ESS Paul Harmon, David King, Expert Systems, A Wiley Press Book, New York, pp 58-60, 1985.

6. Thomas G. Dietterich, A comparative review of selected methods for learning from examples: Machine Learning An Artificial Intelligence Approach Volume I, tioga publishing company, Palo alto,California, pp 41-56, 1983.

7. Karl G. King, The Artificial Intelligence and Expert Systems for CPAs, American Institute of Certified Public Accountants, Inc., New York, New York, pp 1-32, 1987.

8. Ryszard S. Michalski, A theory and methodology of inductive learning: Machine Learning An Artificial Intelligence Approach Volume I, tioga publishing company, Palo alto,California, pp 83-112, 1983.

9. John R. Anderson, Knowledge Compilation The General Learning Mechanism: Machine Learning An Artificial Intelligence Approach Volume II, Morgan Kaufmann Publishers, Inc., Los Altos, California, pp 289- 308.

10. Susan J. Scown, The Artificial Intelligence Experience, Digital Equipment Corporation, Maynard, Massachusetts, pp 15-74, 1985.

11. Norman Hass, Gary G. Hendrix, Learning by being told; Acquiring knowledge for information management, Machine Learning An Artificial Intelligence Approach Volume I, tioga publishing company, Palo alto, California, pp 405- 427, 1983.