Building a Natural Language Interface to Expert Systems

Galina Datskovsky Moerdler

Technical Report #CUCS-306-87

Table of Contents

1. Introduction 1
1.1 Motivation and Domain 2
1.2 Previous work in semantics 6
1.2.1 Expert Systems vs. Data Base Systems 6
1.2.2 Other work in Semantics 6
1.3 Outline of the Proposal 7
2. Requirements for a Natural Language Interface to Expert Systems 8
2.1 Semantic Processing 9
2.1.1 Verb Categories 9
2.1.2 Noun Features 10
2.1.3 Parsing Examples 12
2.1.4 Association Rules 13
2.1.5 Goal derivation from user questions 13
2.1.6 Role Filling and Disambiguation 14
2.2 Director 15
2.2.1 Implementation 16
2.2.1.1 Interpreter 16
2.2.2 Queries and Focus 17
2.2.2.1 User Control of Backward Chaining. 19
2.2.2.2 Maps 19
2.2.2.3 Self Description 20
2.3 To be (Addressed) or not to be 21
2.3.1 Issues that will not be addressed 22
3. Conclusions 23

List of Figures

Figure 1-1: Natural Language - Expert System Interaction 2
Figure 1-2: User - System Interaction Using a Menu 4
Figure 1-3: User - System Interaction Using Natural Language 5
Figure 2-1: Example of Categories 10
Figure 2-2: Partial Tree formed for the *Transfer of possession* category 11
Figure 2-3: Algorithm Used by the Inference Engine 17
Figure 2-4: Summary of the Heuristics for Backward Chaining 17
Figure 2-5: A set of rules from the Dependency Agent 18

1. Introduction

Some expert systems have to gather data from a user in order to solve a problem. In many such expert systems this information gathering is done via a menu interface [Shortliffe 76] [Clancey 79], which asks questions and gives the user a small selection of potential answers, usually in multiple choice format. This form of interface has several limitations [Datskovsky 84]:

- Choice of Input. A user is limited in the choice of input. If none of the choices provided by the system are adequate, the user can not just give an arbitrary answer, however more satisfactory it may be. Moreover, since a menu in effect spans out a tree with many paths, a set of multiple choices a user sees at any given point depends on answers to previous questions. Therefore, if none of the choices presented to a user satisfy his needs, he may end up going down the wrong path, and may find it difficult to back up to the point where the wrong choice was made.
- Additional Information. Sometimes, in order to answer a question a user may need extra information from the system, but there is no facility for him to ask for that information at an arbitrary point. Although the system may be able to provide such information, in order to acquire it the user would have to choose a totally different sequence of multiple choice answers.
- Volunteering Information. A user can not volunteer additional information at an arbitrary point, no matter how important he thinks it is.

Many of these problems can be alleviated by a natural language interface which allows a user to volunteer information at any point during the session and request additional information. A natural language interface that replaces a menu interface must be able to translate user queries into facts, also called predicates, and goals[1] of the underlying expert system.

The idea of building a natural language interface to an existing underlying system is not a novel one; such interfaces have been constructed to data base. These interfaces generally rely on schema that describe the underlying systems. The task of building an interface to an expert system is more difficult because the expert system does not provide such a schema for the semantic interpreter to rely on. Hence, the semantic interpreter must impose a structure on the underlying expert system, while at the same time remaining general and applicable to other systems. Furthermore, a natural language interface places additional requirements on the inference engine of the expert system itself; namely to efficiently utilize the facts entered by the natural language module.

---

[1]Throughout this paper, goal implies the problem for the expert system to solve, or predicates to be proved or disproved, not the goals of the user.

The primary goal of this research is to build a general semantics that will allow the mapping of user statements and questions into facts and goals of the underlying expert system. The semantic interpreter we propose to build will impose a structure on the underlying expert system, while being linguistically based and to some extent transportable. This last criterion requires that it separate domain dependent and independent information. Some of the main features of our semantic approach include linguistically based, hierarchically structured verb categories, a parsing algorithm that is encoded directly into the hierarchies and a mechanism to deal with semantically incomplete input. All of these features are discussed in detail later in the paper. A secondary contribution of this work is to build an inference engine, Director, for expert systems that meets the requirements imposed by a natural language interface. Figure 1-1 shows an overview of the total system as we envision it.

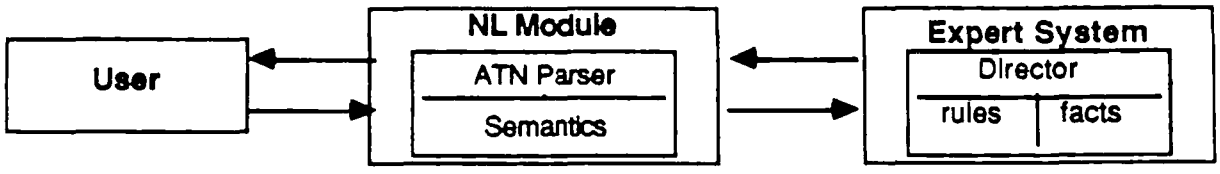

Figure 1-1: Natural Language - Expert System Interaction

1.1 Motivation and Domain

A natural language interface can relieve some of the problems associated with the menu interface by allowing the user to receive advice in the most informative and least time consuming way. That is, overall session length should be shorter using natural language since the system will have to pose fewer questions. This is possible because natural language allows the user to volunteer more than just the requested information whenever the expert system presents a question to the user or the user asks a question of the system.

The semantics and control strategy we propose is aimed at the problem of interpreting natural language input, deriving and making use of any additional facts volunteered by the user, and deriving facts and goals from the user's questions. We are testing our ideas in the domain of tax and financial advising and using a small expert system called Taxpert[2] [Ensor, Gabbe and Blumenthal 85], which

[2]Taxpert is being constructed in conjunction with AT&T Bell Laboratories

deals with personal income tax matters, as our experimental environment. Taxpert consists of a number of agents that cooperate to solve an assortment of tax problems. In particular, we are using two of the agents, the Dependency agent that helps the user determine whether someone qualifies as his dependent and the Filing Status agent that helps the user determine his filing status. The Dependency agent contains over 80 rules and the Filing Status agent over 60 rules.

A brief comparison of sessions using a natural language interface and a menu interface is now in order. Figure 1-2 presents a hypothetical example from the tax domain in which the user is communicating with the expert system through a menu interface. The user is interacting with the part of Taxpert that is responsible for determining whether one individual can claim another as his dependent. An individual must meet five requirements in order to qualify as another person's dependent. These include the type of support given, relationship between the individuals, citizenship, income, and the type of return filed. With the menu interface, Taxpert needs to ask questions about each of the five requirements for dependency. The system has to pose nine questions before arriving at the conclusion that the individual may claim his child as a dependent.

With natural language, the number of questions can be much smaller, because the user can volunteer extra information. Hypothetical interaction between the system and a user via a natural language interface is shown in Figure 1-3. In this example, the user first wants to know whether his daughter Mary meets the income requirement. At the beginning of the session he volunteers some information, i.e. *potential dependent is his child* and *potential dependent is a student*, posing *potential dependent passes the income requirement* as the goal for the expert system to prove. The user has supplied enough information for the system to prove the goal without any additional questions, because if the child of a user is a student, she automatically meets the income test, and it returns an affirmative reply. Next, the user poses a new goal, *Can he claim Mary as a dependent.* The system proceeds to ask further questions about the three remaining requirements, starting with citizenship. In the answer to the question posed about Mary's filing status, the user not only supplies the information requested, but also the additional fact *user is providing over half of support to the dependent*, thus eliminating the need for any questions about the support test. Without any more questions the system is able to determine that the goal is true. Note that because the user volunteered *dependent is a student*, the system avoided three menu questions (questions 4-6), which try to

1. System: Would you like to talk about:
   a. Child
   b. Parent
   c. Sibling
   d. etc..

   User: a

   Fact added : (?dependent is child_of ?user)

2. System: Is your child a:
   a. US citizen
   b. US resident
   c. resident of Mexico
   d. Etc..

   User: a

   Fact added: (?dependent is US_citizen)

3. System: Is your child planning to file a joint tax return for 1986?
   a. Yes
   b. No

   User: a

   Facts added: (?dependent is filing_joint)

4. System: Is he/she filing a joint tax return to claim a refund only?
   a. Yes
   b. No

   User: a

   Facts: (not ?dependent is filing_joint)

5. System: What is the annual taxable income of your child?
   a. Under $1040
   b. Over $1040

   User: b

   Fact: (?dependent is income >1040)

6. System: How old is your child?
   a. Over 19
   b. Under 19

   User: a

   Fact: (?dependent is age >19)

7. System: Is your child a full time student?
   a. Yes
   b. No

   User: a

   Facts: (?dependent is student)

8. System: How much of your dependent's support did you provide last year? (enter a number only)

   User: 1000

9. System: What was the total amount of support that your dependent received last year?

   User: 1800

   Facts added: (?user is providing_over_half_of_support_to ?dependent)

10. System: You may claim your child as a dependent.

Figure 1-2: User - System Interaction Using a Menu

determine whether the dependent meets the income requirement.

User: My daughter Mary, who is a student, earns $2000 a year. Is her income too large to be my dependent?

Facts:[3] (?dependent is child_of ?user) and (?dependent is student) (?dependent is gross_income ?income)

Goal: (?dependent is passed_income_test)

System: Her income is not too large.
User: Can I claim her as my dependent?

Goal: (?user is able_to_claim ?dependent)
System: Is Mary a US citizen?

User: Yes.

Facts: (?dependent is US_citizen) (?user is providing_over_half_of_support_to ?dependent)

System: Is Mary filing a joint tax return for any reason?
User: Yes, to get a refund. She does not earn enough and I provide more than half of her support.

Facts derived form the answer: (?dependent is filling_joint_return) (?dependent is getting_refund)

System: You may claim Mary as your dependent.

Figure 1-3: User - System Interaction Using Natural Language

The natural language module is responsible for interpreting incoming queries and statements as facts and goals. Since the user can enter any amount of information at arbitrary times with a natural language interface, the underlying expert system must be able to make use of volunteered information to avoid asking unnecessary questions. In the example of Figure 1-3 , note that Taxpert need not ask any questions about the relationship between the individuals since this information was volunteered in the first user statement. Since question asking is normally determined by the sequence of rule firings in expert systems, a control strategy was developed and fully implemented in Director[4] that determines which rules to fire so as to minimize the number of questions, to ensure that the system efficiently utilizes the volunteered information and that the questions are asked in a focused and coherent manner.

[3]During processing the variables *?dependent*, *?user* and *?income* will be instantiated with the appropriate values.

[4]Director currently serves as the interpreter to the Dependency and Filing Status agents. It is implemented in Zeta Lisp on a Symbolics Lisp Machine.

1.2 Previous work in semantics

1.2.1 Expert Systems vs. Data Base Systems

Although natural language interfaces to data base systems have been successfully constructed [Kaplan 79] [Woods, Kaplan and Nash-Webber 72] [Grosz et. al. 85], the task is more difficult in the expert systems domain. A semantic interpreter for a data base system usually relies on the regular structure of the data base, i.e. the schema describing it. No such regularity or description is available in the expert system case, and furthermore, a typical expert system rule base is irregular and flat, so this structure must be imposed by the natural language interface.

Another major difference is in the function of the two systems. A data base system is not expected to know or solve a user's problem, but only supply the information that the user requests. Consequently, an interface to a data base system must simply be able to retrieve information requested by the user. On the other hand, an expert system is designed to be a problem solver. A user consults it about an issue and it must gather information in order to advise him. The interface must be able to derive the problem to solve, as well as facts that can be used for its solution from any given question and add these facts to the data base (or working memory). The action of extracting a goal and adding facts at the same time has no analogy in a data base system, but would be similar to allowing the user to query and update the data base at the same time. The addition of new information with every user statement means that the system has to pose fewer questions and that the natural language interface is now responsible for managing all the new information. These factors make the implementation of a semantic module for expert systems a difficult task.

1.2.2 Other work in Semantics

Recent work in semantics that has influenced our own includes that of Martha Palmer [Palmer 85], Graeme Hirst [Hirst 83] and Steve Lytinen [Lytinen 84]. However, our approach differs from theirs in several fundamental ways. One of the main differences between our work and the work mentioned above is that our semantics imposes a structure on top of an unstructured underlying system, while being linguistically based and using general verb categories.

Unlike Palmer's work, our interpreter deals with a complex real world domain. It also makes a greater separation between domain specific and domain independent knowledge to allow for a degree

of transportability.

Both Hirst's and Lytinen's works are aimed at representing natural language input in a frame-like formalism and therefore do not adequately address such issues as interpreting semantically incomplete input. While Lytinen's work uses a hierarchical frame-based memory organization, we propose a general hierarchical structure based on verb categories to be used for inferencing.

1.3 Outline of the Proposal

In summary, this thesis makes the following points:

1. In order to build an interface to an unstructured underlying system, some structure must be imposed.

2. The imposed structure must separate domain dependent and domain independent information, in order to be transportable.

3. We propose a linguistically based, general semantics that meets the above requirements, while having value as a semantic approach regardless of the application. The main features of our semantic interpreter include:
   - Linguistically based verb categories that are hierarchically structured.
   - A parsing algorithm that is encoded directly into the hierarchies.
   - A mechanism for dealing with semantically incomplete input.

4. A natural language interface imposes several requirements, such as efficient utilization of volunteered information and minimization of irrelevant questions, on the inference engine of the underlying expert system. We have built Director, an interpreter that answers to these requirements.

The rest of the paper is structured as follows: in section 2 we describe the semantic approach in greater detail; in section 3 we discuss Director; section 4 discusses issues we are planning to address in the future, as well as issues that will not be addressed by this thesis; finally, section 5 presents our conclusions.

2. Requirements for a Natural Language Interface to Expert Systems

The semantic approach we propose relies primarily on verb categorization and hierarchical structuring within each category. The hierarchies are used to impose a structure, similar to the schema in the data base domain, necessary for semantic processing. The leaf nodes of the hierarchies point directly to the facts of the expert system. During parsing, an appropriate hierarchy is selected according to the definition of the verb in the system's dictionary and a selectional restriction based algorithm is used to traverse the hierarchy. The selectional restrictions are based on the noun features. Unlike previous approaches that made use of an assumed structure to define semantics (e.g., data base systems), this approach offers a semantic mechanism for an unstructured underlying system.

Another important factor in the construction of a semantic module is transportability. If the semantics is to be transportable, the imposed structure must be flexible and domain independent. Because the verb categories are general and derived from properties of the verbs, they can be used in any domain where the verbs take on a similar meanings. New hierarchies may have to be added and some hierarchies deleted in unrelated domains. Most of the categories are based on various works in linguistics [Ballmer and Brennenstuhl 81] [Miller 72] [Osgood 79] and computational linguistics [Levin 85] [Webber 71]. Domain dependent and independent information is carefully separated with the former contained in the lower level nodes of the hierarchies and the latter in the top level categories and restrictions on the verbs. Unlike many of the previous approaches, this work captures linguistic generalizations of verb categories while providing modularity and some domain independence.

As already mentioned, the noun features are used as restrictions on the arguments of the verb. They are drawn from the Roget's Thesaurus and are kept as general as possible. Certain noun features also directly imply expert system facts. Nouns with multiple meanings contain more than one feature in the dictionary. In some cases it is clear from other information in the sentence which meaning is implied. At other times, further disambiguation is necessary. Our approach uses a number of association rules that are based on dictionary definitions of the nouns in order to disambiguate the meanings.

2.1 Semantic Processing

The verb hierarchies are designed so that the top level nodes contain the most general information and the nodes become more and more specific as a hierarchy is traversed. The lower level nodes are derived from the more specific, and sometimes domain dependent meanings of the verbs. During parsing, the lower level nodes inherit all the properties of the top level nodes. The leaf nodes point to expert system facts, which constitute the most specific information. The parsing algorithm becomes domain specific only when it reaches the lower level nodes of the parse tree. In some trees the nodes become domain specific as early as level two, while in others they stay general until level four.

2.1.1 Verb Categories

We have looked at over 90 verbs from the tax code and classified them into 12 categories. Analysis of categories of verbs have been done by researchers before. A number of our categories came from previous work in linguistics [Osgood 79] [Ballmer and Brennenstuhl 81]. However, the analysis of verbs was generally done for one category only, such as verbs of motion [Miller 72]. In real world domains, like tax advising, many such categories are necessary. Unlike Schank [Schank 75] [Riesbeck and Schank 76], we do not claim that these categories are adequate for all domains, and that every verb will fall into one of those categories. More categories may be necessary for other domains. We also do not claim that all other roles in a sentence can be anticipated by the definition of the verb.

Figure 2-1 shows the categories and a number of verbs that belong to each one. A dictionary entry for a verb contains the category or categories to which the verb belongs and a plus or a minus, which indicates whether the underlying subject of a sentence is the semantic agent or patient. Each verb category is organized hierarchically. The nodes of a hierarchy are derived from the meanings of the verbs in that category. The leaves of the hierarchies contain expert system facts. A dictionary entry for a verb may also contain a lower lever node of the parse tree if the verb has a more specific meaning.

For example, the verb *to get* has a dictionary entry of *Transfer of possession* <->(the hierarchy for this category is shown in Figure 2-2) indicating that the syntactic subject is generally the patient

CATEGORIES:

- **Transfer of possession:** *Earn, Get, Give, Make, Pay, Provide, Receive, Refund etc...*
- **Relationship:** *Depend, Share, Support, etc...*
- **Change of position:** *Go, Move, Relocate, etc...*
- **Statement of Status:** *State, Assert, Report, Describe, etc...*
- **Change of Status:** *Alter, Change, Disqualify, Modify etc...*
- **Classification:** *Consider, Designate, Regard, etc.*
- **Be Part of:** *Attach, Include, Constitute, etc.*
- **Saving** *Save, File, Store, etc.*
- **Figure out** *Figure, Guess, Find, Determine, Guess, Questions, etc.*
- **Use** *Use, Employ, Apply, etc.*
- **Wait** *Anticipate, Wait, Expect, etc.*
- **Possess** *Have, Own, Possess, etc.*

Figure 2-1: Example of Categories

in a sentence with this verb. In the sentence *John gets $500*, John is the recipient or the patient. The verbs *to pay* and *to earn* have more specific meanings and therefore have lower level tree nodes as entries in the dictionary. The verb *to pay* is defined as *Transfer of possession <+>, money*, because the verb generally indicates the transfer of monetary amounts, and the verb to *earn* as *Transfer of possession <+>, tax*, because it generally indicates the existence of a taxable income.

A selectional restriction like algorithm is encoded into each hierarchy. The restrictions on the arguments of the verbs rely on the noun features. During parsing, the restrictions on the agent, patient, object and modifier of the verb help guide the parse down the hierarchy and derive the appropriate facts. The algorithm starts with the general verb meaning and proceeds down to more specific nodes of the hierarchy, until it reaches an expert system fact, which is most specific.

2.1.2 Noun Features

Since the noun features are mostly drawn from the thesaurus, they are general. If the word in the thesaurus has several features, we choose the one most appropriate for our domain. For example, given the word support, we choose the feature indicating financial support, rather that the feature implying a bearer (i.e. a block supporting another block). The same features can be used in any domain where the nouns take on a similar meaning, thus making the dictionary transportable. If,

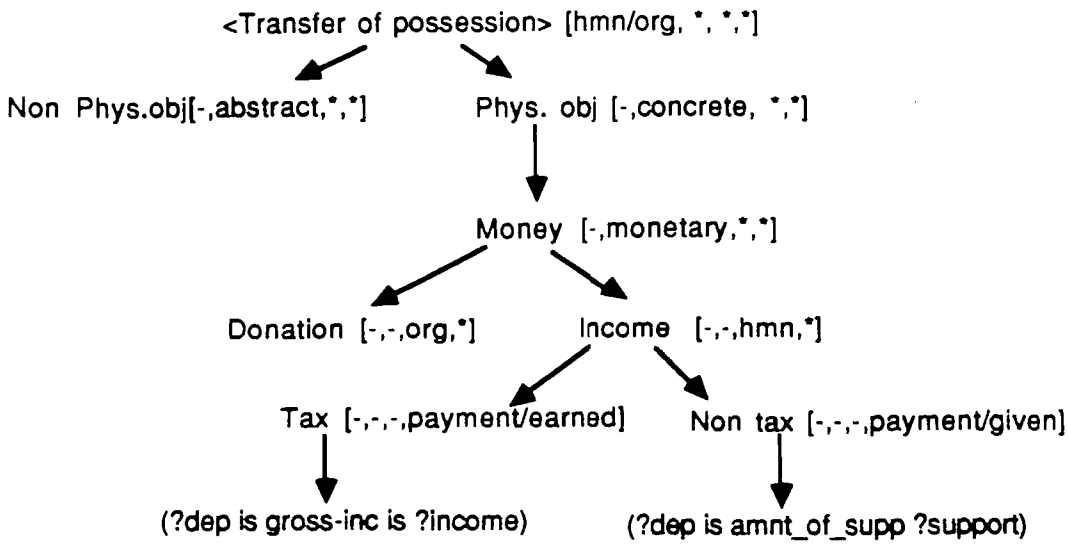

Figure 2-2: Partial Tree formed for the *Transfer of possession* category[5]

however, a different meaning is necessary, it then must be chosen from the thesaurus. We also use the general linguistic features such as as *concrete, abstract, animate, inanimate, human, male, female etc.*.

Let us now look at some examples of noun features. The words son, daughter, step-son, all have the feature *child*. Words like father, mother, step-father, have the feature *parent*. The words child and parent have the feature *relative*. As another example, consider a set of words with the feature *payment*. These include support, salary, dues, etc.. The difference of meaning in these nouns is captured in the features assigned to them. The dictionary entry for support is *payment/given*, while that for salary is *payment/earned*, and for dues, *payment/owed*. The noun features are used as the restrictions on the verb arguments during parsing as well as in deriving facts. For example, the fact (?dependent is child_of ?user) is derived directly from the feature *child*, because in sentences like *Can I claim my son as a dependent*, the word *son* implies the relationship, provided agreement between the subject and the possessive.

[5]In the figure, * stands for wild card, and - means that the feature is inherited from the parent node.

2.1.3 Parsing Examples

In this section we look at two examples. One of them uses a verb that requires the algorithm to start at the top node of the *Transfer of possession* hierarchy, and the second is a sentence with a more specific verb as well as a noun that directly implies a fact. Consider again the partial hierarchy formed for the *Transfer of possession* category shown in Figure 2-2. The arguments in square brackets indicate the restrictions from the agent, patient, object and modifier of the verb. Consider a typical input sentence *I give Mary $500 of support.* The verb *give* is defined in the dictionary as *Transfer of possession <+>*. Thus, during the parse, the *Transfer of possession* hierarchy is chosen based on the definition of the verb in the dictionary. If the verb has a pointer to more than one hierarchy, the one that is considered most frequently used in the domain is chosen first. If the choice proves to be incorrect, the interpreter backs up and tries a new tree. This issue is discussed in more detail later in the paper. Next, the parser has a choice of proceeding down to either *Physical Object* or to *Non Physical Object.* It selects *Physical Object* because *$500*, which is the object of the sentence, fits the *concrete* restriction. At the next level, *concrete* is further restricted to *monetary.* Now, the choice is between *Donation* and *Income.* Here *Income* is selected based on the feature *human* of the patient (Mary), because in this domain a monetary amount given to a human is generally some kind of income, while that given to an organization is a kind of a donation. At the next level, in choosing between *Taxable* and *Non Taxable*, the additional information comes from the modifier (of support) instead of the case roles as before and *Non Taxable* is selected because *support* has the *payment/given* feature in the dictionary. Finally, the fact (?dependent is amount_of_support ?support) is added to the data base (or working memory) of the expert system. Thus, we have gone from the most general node, *<Transfer of possession>* to the specific node, *Non Taxable*, which points to the fact *(?dependent is amount_of_support ?support)*, which is indeed correct because support is considered non taxable income.

Let us look at a second sentence with a more specific verb, *I pay my son a salary of $10000.* The verb *pay* is defined in the dictionary as *Transfer of possession <+>, money*, so the algorithm enters the hierarchy at the *Money* node. It then proceeds down to *Income* because the patient in this case is *human.* Next, because of the modifier *salary* which is defined as *payment/earned*, the node *Tax* is selected, since earned payments are generally taxable, and finally (?dependent is gross-inc ?income) is added to the data base thus specifying the amount of taxable income. Also, the feature

child of the recipient indicates that *(?dependent is child_of ?user)* should be added to the data base.

2.1.4 Association Rules

Sometimes a noun has two features which may be applicable in a given case, and further knowledge is needed in order to decide which feature is most appropriate. In our approach, association rules are used for this type of disambiguation. These rules use syntactic and semantic information from the sentence, as well as English dictionary definitions of the nouns. Let us analyze the sentence *Mary is in school.* This generally implies to a human listener that Mary is a student. Indeed, this is the most frequent definition of the word student, and exactly what we would like the natural language module to derive from the sentence, adding *(?dependent is student)* to the data base.

The features of school are *organization, learning, teaching.* The noun *student* has the features *human* and *learning.* In this example the problem is to disambiguate the meaning of the noun *school* by selecting either *learning* or *teaching.* An association rule helps us decide whether *learning* or *teaching* is most appropriate. The rule states that if the agent of the sentence has the feature *human* and the object or modifier has the features *organization, learning* and *teaching,* and the preposition is in, then the feature *learning* should be selected, allowing the derivation of the associated noun *student.* This rule also applies to sentences like *My son is in a university; Can I claim Mary who is in college? etc..* Because the rules are based on dictionary definitions of the nouns, they are mostly domain independent.

2.1.5 Goal derivation from user questions

The natural language interface must be able to derive goals from user queries, as well as any facts that may be entered in a question. Yes/no questions (such as *Can, Do, & Is*) are processed in the same way as the statements, with the derived fact being the goal. For example, the user may enter *Can John claim Mary who is in school?.* Here the system must not only derive the fact (?dependent is student), but also assert the goal as (?user is able_to_claim ?dependent). This can be done by processing the question just like a statement, and stating that the goal is indicated by the main verb of the sentence. In the example above, the system simply analyzes the sentence *John claims Mary who is in school.* The fact derived from the main verb of the sentence is (?user is able_to_claim ?dependent), so this fact is passed to the system as the problem to be solved. The processing of WH

questions is still to be addressed.

2.1.6 Role Filling and Disambiguation

The parser, as we presented it so far, can handle only semantically complete sentences, i.e sentences that contain all four roles that are used for selectional restrictions. In this section we present a technique for dealing with semantically incomplete input, as well as with situations where a wrong parse tree is initially chosen.

If a user is to communicate with the system in natural language, he will certainly not always enter complete sentences. Consider the sentence *My son gets $2000 a year*. Given the input, there is no way of telling whether the $2000 is taxable or non taxable and the system must acquire this information before it can derive the appropriate fact. However, there are other times when the missing information does not play an important role in deriving an expert system fact, and therefore, the system need not ask for it. In dealing with this issue, we have adopted some of Martha Palmer's techniques and terminology [Palmer 85]. Each of our restriction roles is assigned one of three categories: obligatory, essential, and non essential. Obligatory roles are those that are syntactically mandatory and so are always filled. Essential roles are syntactically optional, but must be filled for proper semantic processing. Finally, the non essential roles are both syntactically and semantically optional and therefore can be omitted or derived based on our domain knowledge and previous input. If during tree traversal an essential role is not fillied at a given level of the tree, the system has to fill it before going on. This can be done in two ways; either a domain default can be used, or the system must ask the user for further information. For example, in the sentence *I would like to claim my son.*, the missing modifier **as a dependent** is implied and filled in by the default mechanism. In the sentence *My son gets $2000 a year*, the system has to ask whether this is taxable or non taxable income.

The non essential roles, when not specified can be derived by the system. However, this may lead the algorithm into the wrong subtree. Consider the sentence *I gave a donation to a University.* The parser has no way of knowing whether the donation is abstract or concrete, but follows the most probable meaning, i.e concrete. If next the system finds out that the donation was a copyright to a book, and it guessed the role incorrectly, it must back up to the point where the guess was made and start again. The same strategy applies when an incorrect parse tree was chosen altogether, i.e. when

the verb has more than one pointer to subtrees in the dictionary. The most frequently occurring meaning is always taken first. If the choice was erroneous, the situation would be quickly detected because the essential roles will not match the features of the verb modifiers, so the algorithm will be forced to back up and try a different tree.

2.2 Director

The semantics described above presents facts to the underlying expert system in a more or less unconstrained fashion. Director is an interpreter for rule-based expert systems that was specifically designed to be able to handle such input. The two major requirements for the interpreter are to efficiently utilize information volunteered by the user, while maintaining a focused and coherent interaction.

A rule-based system executes via the evaluation of its rules. These evaluations are controlled by the system's interpreter, choosing which rules to evaluate according to some strategy. System queries to the user are generated as the rules attempt to determine the values of various data. Therefore, in these systems the goal of minimizing the number of questions and providing a focused interaction can be realized through suitable control of rule firings, i.e., through an appropriate interpreter. Common interpreters for rule-based systems are based on sequential statement evaluation (e.g., [Bobrow and Stefik 83]), forward chaining (e.g., [Forgy 81]), or backward chaining (e.g., [Van Melle 81]). Sequential control, often used as the basis for specialized user-programmed control structures, is of little direct assistance in building rule-based systems. Systems that are restricted to forward chaining inference violate the coherence requirement because they focus on deriving inferences from a set of facts, rather than investigating hypotheses. Systems restricted to backward chaining often do not allow a user to volunteer information, ignoring inferences from new information. More than a simple combination of forward and backward chaining is necessary, thus Director is based on a heuristically controlled combination of the two strategies, and so is able to efficiently utilize the facts entered by the natural language module.

2.2.1 Implementation

Since each rule is selected according to the selection procedure contained within the interpreter, this procedure influences the structure of the rules and the control information that must be explicitly encoded into the system. Indeed there is probably no major expert system in which the rules are independent of their interpreter [Duda 84]. In Director, each rule is invoked as a function, whose body is an if-then form in which the premise and the action are restricted Lisp s-expressions.

Each rule premise is restricted to data base (working memory) queries, i.e., the examination of the values of facts. The value of a fact may be added to the data base in only two ways: either through the action of a rule or through user input. Any fact that is not added by the action of a rule has an associated query procedure so that the user can supply its values. This query procedure is invoked if the premise of a rule tries to examine the fact's value, and the value is not present in the data base. Director automatically maintains the mappings between the rules and the query procedures for their associated facts.

The action of a rule is restricted to a single data base assignment. The value to be asserted may be a constant, the value of a datum, or the result of a function evaluation. However, any input/output performed by such a function is beyond the control of Director. No query procedure is automatically associated with the fact mentioned in the action of a rule.

2.2.1.1 Interpreter

Director uses both forward and backward chaining. When a fact is given to the system, all possible inferences from the data in the current data base of facts are made using forward chaining. This means that full consideration is given to newly entered facts. Thus, forward chaining promotes a focus of attention according to the facts offered to the system by its user. When a user query is received, Director establishes a goal, a hypothesis, to confirm or reject. If the goal is not satisfied by simply examining the data base, backward chaining occurs. Backward chaining is guided by heuristics that try to maintain focus of attention according to both the user query and the facts recently mentioned (see Section 4.2.1). During backward chaining the system may have to ask the user for information. At this point, additional data may be entered, and forward chaining is performed to determine all inferences of this new information. This control structure allows Director to efficiently utilize information volunteered by the user, minimize queries, and shift focus and goals

in response to the user's change of focus and goal. See figure 2-3 for the algorithm used by Director.

Given a fact or facts entered by the user:
1. **Forward chain making all possible inferences without asking any questions.**
2. **If Goal is not found -- Backward Chain.**
3. **Forward chain on all additional data.**

Figure 2-3: Algorithm Used by the Inference Engine

2.2.2 Queries and Focus

Because rule evaluation causes question asking, Director must select rules for evaluation in a way that tries to minimize the number of queries posed to the user. This is done through the use of heuristics, as well as by recording information about user inputs. The heuristics, which are summarized in figure 2-4, determine which rule is most appropriate for evaluation based on the number of known facts in that rule as well as on focus considerations.

1. Select the rule with the greatest number of most recently entered facts in the left hand side
2. In case of a tie in situation 1, pick the rule with the greatest number of known facts derived by the system.
3. If the user has not yet entered any input, choose the rule with the greatest number of known facts

Figure 2-4: Summary of the Heuristics for Backward Chaining

Consider the set of rules in figure 2-5, which come from the Dependency agent of Taxpert. Suppose the user enters the following statements: *John gets $500 of support from Fred. Fred is the only one supporting John who is his father. Can Fred claim John?*. These sentences add facts *(?dependent is parent_of ?user)*, *(?User is the only supporter of ?dependent)* and *(?dependent is amount_of_support ?support)* to the data base and state that the goal is to know whether *(?dependent is claimable)*. Director first forward chains to make all the possible inferences given the contents of the data base. In this case rule 1 is evaluated, adding *(Relationship-test is met)* to the data base. Now the system backward chains starting at rule 4. It then determines that in order to prove *(?dependent is claimable)*, it must first prove *(Support_test is met)*, so the system backward chains again with the

new goal. At this point the system may have to ask the user for some information. We want Director to pick the next rule in such a way as to guarantee the most focused conversation. To promote this behavior, Director tries to select the rule with both the goal in its right-hand-side and the greatest number of facts most recently added by the user in its left-hand-side. This implies that Director must differentiate those facts derived by rules and those entered by the user. Furthermore, Director must assign a time-stamp to each fact added by the user. In this example Director would try rule 3 first, because it contains *(?User is the only supporter of ?dependent)* which was entered by the user. Now the system has to ask only one question, to determine whether *(?User gives 50%)* is true before answering the user's question. However, if *(?User is only supporter of ?dependent)* were unknown, the system would choose arbitrarily between the rules 2 and 3. If rule 2 was chosen first, the user might have had to answer two additional queries, namely whether *(?User alone gives over 10%)* is true and whether *(?dependent gets multiple-support)* is true, before going on to consider rule 3 and answering the questions associated with the facts in that rule as well.

1. If (?dependent is parent_of ?user)) Then (Relationship_test is met)
2. If (Relationship_test is met) (?dependent gets multiple-support) (?User alone gives over 10%)) Then (Support_test is met)
3. If (Relationship-test is met) (?User is the only supporter of ?dependent) (?User gives over 50%))) Then (Support-test is met)
4. If (Relationship-test is met) (Support_test is met))) Then (?dependent is claimable)

Figure 2-5: A set of rules from the Dependency Agent

The heuristics are guided by the facts entered recently and thus do not always give optimal behavior. However, if the user mentions relevant facts as he issues queries (as would be expected if the user understood the problem domain), the behavior should be quite natural, giving a focused conversation, and minimizing the number of system questions.

2.2.2.1 User Control of Backward Chaining.

Sometimes the user has semantic knowledge of the queries and can, therefore, better direct the selection of rules. Forward chaining can be controlled simply by the facts that are added to the data base. Backward chaining can be controlled by the facts and the queries issued to Director. An optional mechanism is provided in Director to allow the user to help direct the rule selection process. Normally, expert systems use only information in the right-hand-sides of their rules to initiate backward chaining. Director can also use information in the left-hand-side of rules when selecting rules to use as a starting point of the backward chaining process. If the user supplies this left-hand-side information when making a request, it will be used in the initial rule selection. For example, consider the following set of rules:

1. If (?dependent is a child) (?dependent is a student)) Then (Gross_income_test is met)
2. If (?dependent is a child) (?dependent is ?age < 19)) Then (Gross_income_test is met)

Suppose a user issues the following query: *Do students automatically meet the income test?*

The fact *(?dependent is a student)* and the goal *(Gross_income_test is met)* are derived from the question above and added to the data base. Using the maps, Director identifies that the general goal is implied by rules 1 and 2. The system now selects a rule as the starting point of the backward chaining process, choosing rule 1 according to user control. If this information was not available, or if the system did not take it into account, rule 2 may have been selected first and additional questions may have been generated.

2.2.2.2 Maps

During rule selection the interpreter must know which facts are contained in the left- and right-hand sides of the rules. This information can be obtained by searching the rule set. Naive searches, however, could be expensive computationally and could make the response time of the system unreasonable. To make this searching efficient, Director maintains two maps. These maps are the rules-add-fact map (RF), and the facts-used-by-rule map (FR). The RF map provides pointers from each fact to the rules that can add it to the data base. The FR map provides pointers between each rule and the facts contained in its left-hand-side, thus specifying which facts have to be true in order for that rule to fire. The maps are built up during a preprocessing stage, which has to be performed only once for a given set of rules.

First, let us look at the RF map. Suppose that fact C is in the right-hand-side of rule r: *(If (A ∧ B) then C)*. The RF map entry for this fact would be *(r<-->C)*, indicating that rule r adds fact C to the data base. The information in this map is used during the rule selection portion of the backward chaining phase. For example, if Director is trying to solve goal C, then the RF map provides efficient access to r. This map also allows Director to suppress the firing of certain rules: After a value is assigned to a fact, the system checks the RF map and tries to mark those rules that would assert the same value of this fact. (Rules are not evaluated during the marking process, hence the only rules marked are those that reference this fact by a constant name and assert the same value as a constant.) The marked rules are not evaluated, thus avoiding rule evaluation and the superfluous queries to the user that these evaluations might cause.

Similarly, the facts-used-by-rule map would contain an entry for rule r, *(A,B<-->r)*, indicating that rule r depends on facts A and B. The map would also contain entries for A and B showing that rule r requires their values in order to be evaluated. When some fact A is added to the data base, all those rules that use A in a forward chaining inference are readily found. In the present example, if A and B are in the data base, rule r is found in the FR map to be usable for forward chaining. This map is also used in the rule selection process of backward chaining. Having determined that rule r will be used to infer a needed fact C, the system readily determines that facts A and B need to be known.

2.2.2.3 Self Description

So far, we have described what we call Director's inferencing function. The system has another function, called **Display**. There are many instances when a user wants the system to provide information without providing inferencing. For example, a user may want to see everything the system knows about a certain fact and issue the following request: "Tell me about the citizenship requirements." Our system can handle a query of this sort by using the maps. All the rules that contain *citizenship_test* in the left-hand-sides are found with the help of the FR map. Similarly, all the rules containing *citizenship_test* in the right-hand-side are found with the help of the RF map. All rules containing *citizenship_test* are returned as a response to the above query to the semantic module that translates user questions into requests to Director. Providing this information is done quickly because Director does not perform inferences, but rather only references the maps.

2.3 To be (Addressed) or not to be

In the previous section we outlined the construction of the semantic module and described Director, an interpreter designed to accommodate a natural language interface to an expert system. In this section we look at other issues that will be addressed by this thesis, as well as at issues that will not be addressed.

The construction of the semantic module is not yet complete. We have begun work on several other issues that will be addressed in this thesis.

1. Processing of WH questions. An answer to a WH question is generally more complex than an answer to a Yes/No question. The answer may involve proving more than one goal from both the left and right hand sides of the rules. The amount of information the user desires is generally greater than in a Yes/No question. For example, in the questions *When can I claim my elderly father who lives with me?* the user supplies facts *(?dependent is father of ?user)* and *(?dependent is member_of_household of ?user).* His goal is to know under what conditions he can claim his father. An answer to such a question involves more than just proving a goal, but enumerating all the possible conditions under which the user can claim his father.

2. Instantiation of variables. How to instantiate variables in the expert system facts is another important issue that we must address. Consider the question *Can John claim Mary as a dependent?*. The system must be able to instantiate the variables *?dependent* and *?user* with *Mary* and *John*, respectively. This instantiation depends on the syntactic and semantic information derived from the sentence (i.e. the agent, patient, type of verb (<-> or <+>) etc.). Part of this information comes from the selectional restrictions in the parse trees. The methodology necessary for this instantiation is currently being investigated.

3. Dealing with ambiguous input. Another interesting issue is when to check whether the words a person is using correspond in meaning to the system's definitions of these words. For example, the word *salary* generally implies taxable income, however, a person may use it to refer to non taxable income, such as a graduate stipend. In such a case, the system should ask for the source of the income to insure a correct result. It is unclear how refined such clarification should be, although we suspect it should closely parallel the distinctions available in the previously used menu. We plan to investigate the extent to which such refinement is necessary and develop an adequate mechanism to deal with this issue.

4. Implementation. We have started to implement our ideas, but a lot more work still needs to be done. Currently, Director and the syntactic parser are fully implemented. Two of the parse trees are fully encoded, as well as part of the parsing algorithm. We still have to fully encode all of the semantic trees as well as fully implement the parsing algorithm. Work on this part of the system is currently under way.

2.3.1 Issues that will not be addressed

There are several interesting issues, that for lack of time, will not be addressed by this thesis. These include the automatic construction of the parse trees and automatic classification of facts in these trees. If, for a given domain, a certain tree has to be extended, such extension will have to be done by hand. Also, facts will be hand encoded into the parse trees. Although an algorithm for automatic classification such as the acquisition module used by TEAM [Grosz et. al. 85], where an Expert System Expert can interactively enter all the necessary information about a given expert system, is desirable, it will not be designed as part of this thesis.

Another interesting problem occurs in the area of human factoring. Is natural language interfacing truly superior to a menu approach? Is some combination of the two most appropriate? These questions can only be answered through a series of psychological experiments that compare the performance of the natural language alone, and the menu system alone. Through careful analysis of these experiments, a happy medium may be achieved. Although we are planning to run a psychological experiment of this sort, we do not plan to do the extensive analysis required in order to make a system with a fully developed interface that incorporates the results of these tests and can therefore be used in a tax or financial office.

Finally, language generation issues will not be addressed and the expert system will use canned text to present its solutions to the user.

3. Conclusions

In this proposal we described two important issues that must be addressed when constructing natural language interfaces to expert systems. First we described a semantic mechanism that translates user statements into facts and goals of the underlying expert system. This semantic approach is based on verb categorization. Each category is structured hierarchically, and the parsing algorithm is directly encoded into each hierarchy. The hierarchies provide a structure on top of the expert system, which makes semantic processing possible. The noun features used in the system are drawn from Roget's thesaurus and fairly general. The system uses a number of association rules to disambiguate certain noun features. We have done some work in dealing with partial matching. Our module is also able to deal with verbs that have more than one meaning in a domain and derive user goals from yes/no questions. Some issues in the construction of the complete semantic module are still being investigated. These include instantiation of variables in the facts, derivation of goals from WH questions, as well as complete implementation. The semantics presented is not only useful in the expert systems domain, but also in any domain where the underlying system is not well structured.

The natural language module adds facts to the data base of the underlying expert system in an unconstrained manner, thus placing extra requirements on the underlying expert system. We discussed Director, an inference engine that uses a combination of forward chaining and backward chaining so as to efficiently utilize facts entered by the natural language interface. It makes available descriptions of its rule base and allows for a limited form of user control over its backward chaining mechanism. This facility allows the user to ask questions about the information contained in the rules but not normally supplied by expert systems. These attributes of Director allow a knowledgeable user to arrive at a solution to his query in the most efficient and least time consuming way, while maintaining a focused dialogue. Director is also useful in domains where decisions have to be made quickly, or where user queries are expensive, such as expert systems designed for use by busy professionals, such as accountants and doctors, as well as systems that work in hazardous environments, such as nuclear reactors.

References

[Ballmer and Brennenstuhl 81]
Levelt, W.J.M. (editor).
Springer Series in Language and Communication, volume 8: Speech Act Classification.
Springer-Verlag, 1981.

[Bobrow and Stefik 83]
Bobrow, D.G., Stefik, M.
The LOOPS Manual
Xerox Corp., Palo Alto, Ca., 1983.

[Chafe 70] Chafe, W.L.
Meaning and the Structure of Language.
University of Chicago Press, Chicago and London, 1970.

[Clancey 83] Clancey, W.
The Epistimology of a Rule-Based Expert System - a Framework for Explanation.
Artificial Intelligence 20 , 1983.

[Clancey 79] Clancey W.
Dialogue Management for Rule-Based Tutorials.
In *Proceedings of IJCAI.* Japan, 1979.

[Datskovsky 84] Datskovsky, G.
Menu Interfaces to Expert Systems: Evaluation and Overview.
Technical Report, Columbia University, New York, 1984.

[Datskovsky 85] Datskovsky, G.
Natural Language Interfaces to Expert Systems.
Technical Report, Columbia University, New York, 1985.

[Davis 78] Davis, R.
Knowledge Acquisition in Rule-Based Systmes-Knowledge About Representation as a Basis for System Construction and Maintanance.
Pattern Directed Inference Systems.
Academic Press, 1978.

[Duda 84] Duda.
Presentation at the IEEE Workshop on Principles of Knowledge-Based Systems.
1984

[Duda, Gasching and Hart 79]
Duda, R., Gasching, J., Hart, P.
Model Design in the Prospector Consultant System for Mineral Exploration.
In Michie, D. (editor), *Expert Systems in the micro-electronic age.* Edinburgh University Press, 1979.

[Ensor, Gabbe and Blumenthal 85]
Ensor J., Gabbe J., and Blumenthal R.
Taxpert -- A Framework for Exploring Interactions Among Experts.
1985.
in preparation.

[Forgy 81] Forgy, C.
OPS5 User's Manual
Carnegie-Mellon University, Pittsburgh, Pa., 1981.

[Grosz et. al. 85] Grosz, B.,Martin, P., Appelt, D., Pereira, F.
Team: An Experiment in the Design of Transportable Natural Language Interfaces.
Technical Report, SRI International, 1985.

[Hirst 83] Hirst, G.
Semantic Interpretation Against Ambiguity.
PhD thesis, Brown University, 1983.

[Kaplan 79] Kaplan, S.J.
Cooperative Responses From a Portable Natural Language Data Base Query System.
PhD thesis, University of Pennsylvania, 1979.

[Levin 83] Levin, B.
On the Nature of Ergativity.
PhD thesis, MIT, 1983.

[Levin 85] Levin, B.
Lexical Semantics in Review: An Introduction.
In Levin, B. (editor), *Lexical Semantics in Review.* MIT, 1985.

[Lytinen 84] Lytinen S.L.
The Organization fo Knowledge in a Multi-lingual Integrated Parser.
PhD thesis, Yale University, 1984.

[McKeown 82] McKeown K.R.
Generating Natural Language Text in Response to Questions About Database Structure.
PhD thesis, University of Pennsylvania, 1982.

[McKeown 86] McKeown, K.
Communication with Expert Systems.
In *Proceedings of JCC.* 1986.

[Miller 72] Miller, G.A.
English Verbs of Motion: A Case Study in Semantics and Lexical Memory.
Coding Processes in Human Memory.
V.H. Winston and Sons, 1972.

[Osgood 79] Osgood, Charles, E.
Focus on Meaning Volume I: Explorations in Semantic Space.
Mouton Publishers, 1979.

[Palmer 83] Palmer, M.
Inference-Driven Semantic Analysis.
In *Proceedings of the AAAI.* 1983.

[Palmer 85] Stone Palmer, M.
Driving Semantics for a Limited Domain.
PhD thesis, University of Edinburg, 1985.

[Pollack 83] Pollack, M.E.
Generating Expert Answers Through Goal Inference.
Technical Report, SRI International, 1983.

[Riesbeck and Schank 76]
Riesbeck, C. K. and Schank, R. C.
Comprehension by computer: Expectation-based analysis of sentences in context.
Studies in the Perception of Language.
John Wiley and Sons, Chichester, England, 1976.

[Schank 75] Schank, R . C.
Conceptual Information Processing.
North Holland, Amsterdam, 1975.

[Schank and Riesbeck 81]
Schank, R.C. and Riesbeck, C.K.
Inside Computer Understanding.
Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

[Shortliffe 76] Shortliffe, E.H.
Mycin: A rule-based computer program for advising physicians regarding anitimicrobial therapy selection.
PhD thesis, Stanford University, 1976.

[Van Melle 81] Van Melle, W.
The Emycin Manual.
Technical Report Stan-cs-81-885, Stanford University, Stanford, Ca., 1981.

[Webber 71] Nash-Webber, B.,
Verbs of Composition.
1971.
Harvard University, 1971.

[Woods 73] Woods, W.A.
An Experimental parsing System for Transition Network Grammars.
In Rustin (editor), *Natural Language Processing.* Algorithmic Press, 1973.

[Woods, Kaplan and Nash-Webber 72]
Woods W., Kaplan R., Nash-Webber B.
The Lunar Sciences Natural Lnaguage Information System: Final Report.
Technical Report 2378, BBN, Cambridge, Mass, 1972.