AGNESS: A GENERALIZED NETWORK-BASED EXPERT SYSTEM SHELL*

James R. Slagle
Michael R. Wick
Marius O. Poliac

Computer Science Department
207 Church Street, S.E.
136 Lind Hall
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

AGNESS is an expert system shell developed at the University of Minnesota. AGNESS is more general than other shells. It uses a computation network to represent expert defined rules, and can handle any well-defined inference method. The system works with non-numeric as well as numeric data, and shares constructs whenever possible to achieve increased storage efficiency. AGNESS uses a menu-driven user interface, and has several features that make the system friendly and convenient to use. The system includes eight explanation queries designed to increase the amount of information available to the user, the expert, and the knowledge engineer while remaining simple enough to be included in most of today's expert system shells. AGNESS has been tested on several domains ranging from simplified problems to real world medical analysis.

I. INTRODUCTION

The design of expert consultation systems has been a topic of growing interest in Artificial Intelligence (AI) research during the past decade. Numerous expert systems have been constructed to give consultations in a variety of application areas. Two prominent examples of this are MYCIN [1], a program for the diagnosis of infectious diseases, and PROSPECTOR [2], a mineral exploration system. The common aim of expert system technology is to represent and apply knowledge obtained from a specialist in the problem domain. Early in the history of this technology, people realized that rewriting the entire system for a new domain was both wasteful and unnecessary. Since most of the operational code can be separated from the domain specific knowledge, one program can be written to handle rule bases from several domains. Using this idea, a system can be developed for a new domain by simply changing the rules that the operational system handles. This operational system is called a skeletal system or an expert system shell.

Many expert system shells have been implemented recently with varying degrees of success. The best known of these are KEE(Knowledge Engineering Environment) from Intellicorp, LOOPS developed at the Xerox Palo Alto Research Center, and ART (Automated Reasoning Tool) from Inference Corporation [3].

We have developed an expert system shell called AGNESS standing for **A** **G**eneralized **N**etwork-based **E**xpert **S**ystem **S**hell. AGNESS uses a computation network to represent the domain knowledge as opposed to a production rule base. The network is restricted to be a directed acyclic graph. There are several advantages to using a network-based shell as opposed to a simple rule-based shell. For example, in a network-based system, there is no need for searching for the rules to be fired, as all rules are directly connected to the current node. The AGNESS network increases storage efficiency by sharing common constructs whenever possible. The network also allows for visually pleasing graphical representations of the domain knowledge, and lends itself well to data flow analysis.

PROSPECTOR is perhaps the best known network-based expert system [2]. AGNESS has been implemented as a generalization of the network scheme introduced in PROSPECTOR. In AGNESS, constructs are shared to achieve increased storage efficency. AGNESS also has the ability to manipulate any well-defined data type, not just probabilities. For example, a value in the AGNESS system can be a string, or a frame. Also, AGNESS allows for expert-defined inference methods. This gives the system the abiltiy to handle any value propagation method that the domain expert desires.

AGNESS is a shell aimed at a wide variety of domain applications, however, as with all shells, some application areas are better than others. AGNESS is particularly useful in domains that involve matching entities. The matching problem is really a

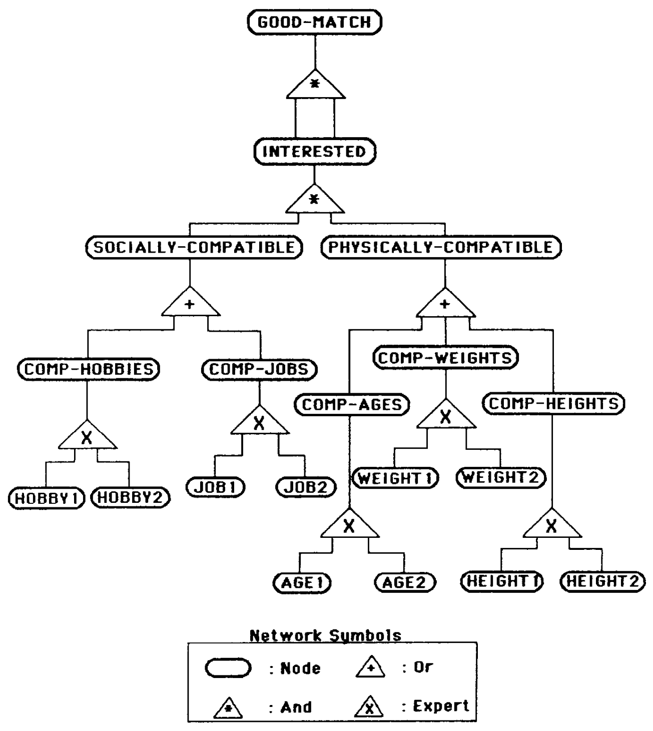

Figure 1. The dating service network

*This material is based partly on work supported by the National Science Foundation, grant no. DCR8512857 and by the Microelectronics and Information Sciences Center of the University of Minnesota.

generalization of the classification problem, and as such, is a widely occurring problem. We will illustrate the AGNESS constructs by means of an example taken from a simplified problem domain of a dating service. The "expert system" we will develop is intended to be used to find the probability that two people make a good match for dating each other. The computation network for this problem domain is presented in Figure 1.

Each general proposition is represented by a node in the network. A value is associated with each node in a given context. For example, in Figure 1, the node **AGE1** may represent the general proposition of the age of a person. Given a context such as **(Steve)**, the node, together with the context, have an associated value, say 24. This is the specific instance of the general proposition **AGE1** representing the fact that Steve's age is 24. In AGNESS, the value associated with a node and context is not required to be a numerical value, but may come from any well-defined data type. For instance, the value associated with the node **HOBBY1** and the context **(Steve)** may be the string "computers" representing the fact that Steve's favorite hobby is computers. The triple made up of the node, the context and the value is called a datum.

The nodes in the network are connected by links called edges representing the possible dependency of one datum's value on that of another. For example, the nodes **AGE1** and **AGE2** are linked (connected by an edge) to the node **COMP-AGES** representing the relationship between the ages of two people and the probability that the two people have compatible ages. The nodes connected by the edges are called the antecedents (**AGE1,AGE2**) and the consequent (**COMP-AGES**) to emphasize their inferential relationship. Associated with each node is a function that takes the value of the antecedent nodes and generates the value of the consequent node. This function is called an inference method corresponding to its function of inferring the consequent value from the antecedent values. For example, the node **COMP-AGES** uses an expert defined inference method that takes the values of the two antecedent nodes **AGE1** and **AGE2** and computes a value for **COMP-AGES**. Given the context of **(Steve)** for **AGE1** and **(Cindy)** for **AGE2**, this would correspond to taking Steve's age and Cindy's age and computing the probability that their ages are compatible and assigning this probability to the node **COMP-AGES** in the context **(Steve,Cindy)**.

A node can be linked to an arbitrary number of other nodes, and it may have an arbitrary number of nodes linked to it. It is important to realize that a node and the value associated with that node (in a given context) are separate entities. The node represents a general proposition whereas the value, stored in a separate database, represents a given instance of a general proposition that occurs during the problem solving process. Domain specific knowledge is relatively fixed, and thus is represented directly in the computation network. User supplied and problem specific knowledge is more volatile and thus is stored in a separate database.

AGNESS has been implemented on a LISP workstation and uses a menu-driven interface. The system operates in several modes and provides a variety of facilities including explanation. AGNESS has also been tested on various problem domains ranging from a simplified expert system on wine to the serial evaluation of ECG exercise tests [4], and has proven to be elegant and powerful.

II. BASIC TERMINOLOGY

A. An Object

The basic element that AGNESS manipulates is called an *object*. An object represents a primitive element to which information may apply. Typically, an object represents a single real-world entity or a group of entities that work together. For example, in our dating service, an object is a given person, such as Steve or Cindy.

B. An Object Type

Objects are grouped into sets referred to as *object types*, such as **male** and **female**. AGNESS supports *basic* object types, defined by enumerating their member objects, and *derived* object types, defined by applying the set-theoretic union, intersection, and difference operators to other object types.

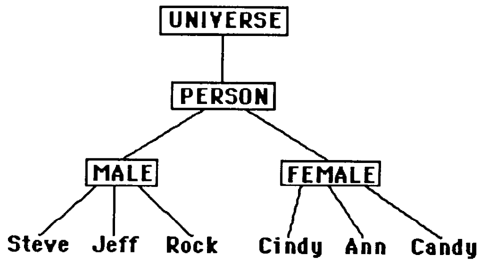

Figure 2. The object type lattice

The basic object types are organized into a structure called the *object type lattice*, representing a partial ordering based on set inclusion. Figure 2 shows the object type lattice for the dating service world. Placement of objects on an object type lattice naturally allows for the representation of fragmentary knowledge. For example, AGNESS will reason about Steve as both a person and as a male. Rules and elementary propositions in a computation network can be made as general as possible to avoid duplication, without diluting the power of the system to reason about specifics.

C. A Datum Function

A *datum function* is a mapping from objects to information about those objects. The English meaning of a datum function is described using a list of words and integers. For example, the datum function **INTERESTED-DF** has the phrase **(The probability that <1> is interested in <2>)**. Each bracketed number in this list, called a *parameter*, corresponds to an element of an ordered list of objects called a *context*. The datum function may be *instantiated* by substituting the elements of the context for the corresponding parameters, resulting in a concrete phrase about specific objects. Instantiating the above datum function in the context **(Steve,Cindy)** yields the phrase "The probability that Steve is interested in Cindy". The use of a datum function enables AGNESS to represent the general antecedent-consequent relationship between un-instantiated concepts instead of the specific relationship between concrete phrases.

D. A Domain Constraint

We have seen how a datum function may be instantiated to yield a phrase. It is important to prevent instantiations that yield meaningless phrases such as "The probability that Steve is interested in Jeff" (assuming a heterosexual dating service). The set of permissible contexts of a datum function is specified using *domain constraints*. A domain constraint is a list of object types that represents the Cartesian product of those types. For example, the above datum function may have the domain constraint **(male,female)** designating that the datum function may be instantiated in any context that contains a male in the first position and a female in the second. Thus a context is a member of the set represented by a domain constraint when each object in the context belongs to the corresponding type in the domain constraint. We say in this case that the context *matches* the domain constraint. A datum function will be instantiated only in contexts that match one of its domain constraints. A datum function may have several domain constraints, allowing **(The probability that <1> is interested in <2>)** to be instantiated in any context matching either the domain constraint **(male,female)** or the domain constraint **(female,male)**.

Multiple domain constraints may be used with the same datum function to divide the domain of the datum function into disjoint parts. This division is useful when the value of the datum is computed differently when instantiated in contexts from different parts of the domain. For instance, the datum function **(The probability that <1> is interested in <2>)**, specifying the probability that one person is interested in dating another person, may be instantiated for any male/female or female/male pair. However, if we allowed the instantiation for any person/course, we would get a completely different idea, namely the probability of a person being interested in some particular course. Obviously, this datum would be derived in a completely different manner than would the earlier datum.

Domain constraints also enable AGNESS to generate the possible contexts for a node, an important consideration when inference is performed (see section 4).

III. KNOWLEDGE REPRESENTATION - THE NODE

In AGNESS, knowledge is represented in the form of a computation network and a database. The network is built from the rules supplied by the domain expert and the database is built from knowledge obtained during the run of the expert system and from the default information. This section describes, in detail, the design and implementation of the AGNESS network and illustrates the ideas with the dating service example.

The main element of the network is the node. A node in AGNESS corresponds to a datum function with domain constraints. That is, a node represents a proposition and the domain in which that proposition is valid. A node is defined as the following 5-tuple: <datum function, constraint-default list, antecedent edges, consequent edges, inference method>.

There are two types of nodes that are of special interest. First, a node with no consequent is called a *top node* and represents a high level topic that is of interest to the system. For example, the top node in the dating service example is **GOOD-MATCH** representing the probability that two people are a good match for dating. A second special type of node is a node with no antecedents, called a *bottom node*. A bottom node represents a topic that the system has no way of inferring from other information, and thus has no associated inference method. An example of a bottom node is **AGE1** or **HOBBY2**. The value of such a node will either be the default value or a value supplied by the user.

A. The Datum Function of a Node

A datum function is associated with each node and is a mapping from objects to information about those objects. It is defined as the following 5-tuple: <arity, phrase, askable, codomain-constraint, self-merit>.

An argument list for a datum function is a list of objects called a context, and the result of instantiating the datum function in a context is called the value. Together a node (with its datum function), a context, and a value are called a *datum*. In this paper we will use data as the plural of datum. Each entry in the AGNESS database is stored as a datum, and retrieved using the node and context as keys.

The *arity* of the datum function is the number of formal parameters. This number is used during the generation phase (called phase I) of the propagation process.

The *phrase* of a datum function is a list of bracketed numbers and text such as **(probability that <1> is interested in <2>)**. The phrase represents the English meaning of the datum function. The instantiated phrase is what the system uses to request or report the value of a datum. This text gives the system some of the advantages of a natural language interface, while retaining the advantages of strictly canned text.

The *askable* flag of the datum function tells the system whether the user may be requested to supply a value for this datum. Use of this field allows the expert to prevent questions that a typical user cannot answer.

The *codomain-constraint* of the datum function is used to verify that a value of this datum is reasonable. That is, the value of any datum that uses this datum function must satisfy the constraint. For example, if the value of the datum is meant to be a probability, the system will use the codomain constraint called **probp** which will return true if the value is between zero and one. This provides the system with a way to screen data that can not possibly be correct.

The *self-merit* of the datum function is a number that is used to calculate Merit [5], a measure of the utility of requesting information from the user. The self-merit associated with each datum function is an expert-defined approximation of the ratio of the expected change in the value of a datum to the expected cost of determining the value. The concept of Merit will be discussed later in relation to the questioning process of the AGNESS shell (see section 5).

B. The Constraint-default list of a Node

The constraint-default list is the second element of a node. Each element of this list is an ordered pair that consists of a domain constraint and a default value. For example, the constraint-default list (**(person) computers**) for a datum function with the phrase **(The favorite hobby of <1>)** defines that computers are the default hobby of every person. The domain constraints in the constraint-default list need not represent disjoint sets of contexts. If a context matches more than one domain constraint in the constraint-default list, the first such constraint and its associated default value apply. For example, a constraint-default list containing (**(male) operating-systems**) and (**(Person) artificial-intelligence**) defines that the default hobby for any man is operating systems, while the default hobby for any other person (simply **woman** in this example) is artificial intelligence.

C. The Edges of a Node

The next two elements of a node are the edges to the antecedents and the consequents. In AGNESS, an edge is explicitly represented as the following 4-tuple: <antecedent, consequent, transformation template, auxiliary information>.

The antecedent of an edge is the node that is used as the "source". For example, the antecedents of the node **COMP-AGES** are the nodes **AGE1** and **AGE2**. It is the data of these nodes that are used in the computation of the consequent datum.

The consequent of an edge is the node that is used as the "destination". It is the datum of this node that is computed using the data of the antecedent nodes. The consequent of **COMP-AGES** is the node **COMPATIBLE**.

A node can have an arbitrary number of antecedents and consequents. The names "antecedent" and "consequent" are chosen from their role in the typical IF - THEN rule.

The *transformation template* of an edge is a list of bracketed numbers that specifies the correspondence between parameters of the antecedent and consequent datum functions. Each bracketed number in a transformation template specifies a single pair of corresponding parameters. The consequent parameter is given by the number's value, while the antecedent parameter is given by the number's position in the transformation template. Every element of the antecedent context must occur in the consequent context. Thus reasoning is constrained to proceed from the general to the specific.

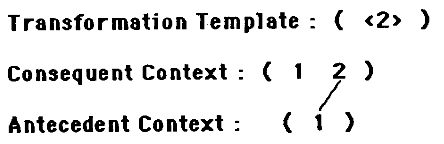
Transformation Template : (<2>)
Consequent Context : (1 2)
Antecedent Context : (1)

Figure 3. Operation of a transformation template

The operation of a transformation template is illustrated in Figure 3. In this example the edge links the antecedent node **HOBBY2** (with a one parameter datum function) to the consequent node **COMP-HOBBIES** (with a two parameter datum function). The first (and only) element of the antecedent context corresponds to the second element of the consequent context because the first element of the template contains <2>. Thus the two parameters must be the same. The first element of the consequent context does not correspond to any element of the antecedent context.

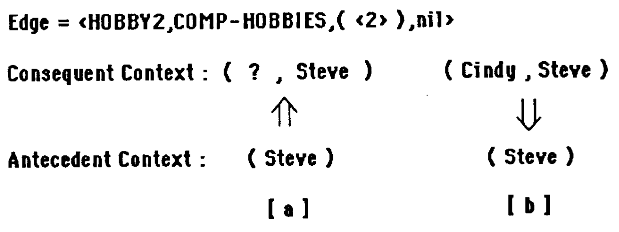
Edge = <HOBBY2,COMP-HOBBIES,(<2>),nil>
Consequent Context : (? , Steve)
(Cindy , Steve)
Antecedent Context :
(Steve)
(Steve)
[a]
[b]

Figure 4. Context mapping during edge traversal

AGNESS uses a transformation template in two ways, corresponding to the two directions in which the edge can be traversed. In proceeding from antecedent to consequent, AGNESS constructs a set of consequent contexts based on the antecedent context. Figure 4a illustrates this traversal. In this example, the transformation template is interpreted from the antecedents' point of view. That is, the template tells the system that the first parameter in the antecedent context (**Steve**) is mapped to the second parameter of the consequent context. Thus the context (**Steve**) for **HOBBY2** is mapped to the context **(?, Steve)** for **COMP-HOBBIES**. Notice, this context is only partially specified. The system will fill in the question mark with all legal objects by using the constraint-default list for **COMP-HOBBIES**. This process will be discussed in section 4. Traversing the edge from antecedent to consequent occurs during the propagation process.

In proceeding from consequent to antecedent, the transformation template tells the system that the second parameter in the consequent context (**Steve**) is mapped to the first parameter of the antecedent. This is illustrated in figure 4b. Thus the context **(Cindy, Steve)** for **COMP-HOBBIES** maps to the context **(Steve)** for **HOBBY2**. Traversing the edge from consequent to antecedent occurs during the questioning process.

The auxiliary information element of an edge holds any additional information a particular inference method might need. For example, the subjective Bayesian inference method requires conditional probabilities. This information can be extracted from the edge and used during the propagation process.

D. The Inference Method of a Node

The last element of a node is the inference method. An *inference method* specifies the relation that holds between the value of a consequent datum and the values of its antecedents. An inference method is defined as the following 3-tuple: <assignment function, antecedent value function, edge merit function>.

Each inference method has an *assignment function* which is a procedure for deriving the value of a consequent datum from the values of its antecedent data. The assignment function is called with one argument for each antecedent, and returns the value of the consequent. The arguments of the assignment function are usually the values of the antecedents. For example, the inference method ***AND*** (probabilistic "and") takes the value of each of the antecedent data, multiplies them together and assigns the resulting value to the consequent datum. In more complicated situations, another function called the antecedent value function constructs the arguments of the assignment function from information present in the computation network.

Thus the second element of an inference method, the *antecedent value function,* is used when the assignment function requires information other than the value of the antecedents. For instance, subjective Bayesian inference methods use conditional and prior probabilities to apply Baye's formula [6]. This information is extracted from the computation network by the antecedent value function. For instance, the conditional probabilities for subjective Bayesian inference are stored in the auxiliary position of the edge. The antecedent value function returns a value suitable as an argument to the assignment function.

The third function making up an inference method is the *edge-merit function.* Merit calculations are used to direct the acquisition of information by identifying questions that are likely to have a large effect on the results of the computation network at a relatively low cost. Since user interaction is frequently the most time-consuming part of expert system use, the intelligent direction of questioning can significantly improve the system's performance. If a questioning mechanism is desired for a computation network built with AGNESS, an edge-merit function must be specified for each inference method.

The current implementation of AGNESS contains inference methods that deduce consequent probabilities from independent antecedent probabilities. Basic logical connectives ("and", "or", and "not") and subjective Bayesian inference have been implemented, as well as Mycin style confidence functions. AGNESS also allows expert-defined inference methods for both general and problem-specific purposes. This feature allows the domain expert to use any inferential relationship that is found to be desirable.

IV. <u>PROPAGATION</u>

Propagation is a procedure invoked each time a new value is added to the data base. The propagation process updates the data base so that values of the consequents of the modified data are consistent with the new values of their antecedents. This means that the value of each consequent datum has been deduced from the values of the antecedent data by applying its inference method. Modifying the values of the consequents thus implies a recursive invocation of the propagation procedure. The recursion is terminated by reaching nodes that have no consequents. The termination condition is insured by requiring that the network be acyclic. The propagation process consists of two phases.

Partially Specified Context : (? , Steve)

Constraint-Default List : (((male, female) 0.8)
((female, male) 0.8))

$\Downarrow$

Matched Constraint : (female,male)

$\Downarrow$

Consequent Contexts : { (Cindy,Steve)(Ann,Steve)(Candy,Steve) }

Figure 5. Context propagation

A. <u>Phase I</u>

The first phase of the propagation process is illustrated in figure 5. First, a partially specified context is generated from the antecedent context using the transformation template associated with the edge. Next, the unspecified parts of the consequent context are filled in using the domain-constraint list of the consequent node. In this example, the partially specified context (**? , Steve)** matches only one domain constraint, namely **(female,male)**. This tells the system that the first parameter of the partially specified context can be filled in with any female object. Doing so gives a set of all the contexts for the consequent that will be affected by the change in the antecedent datum.

B. <u>Phase II</u>

In the second phase of the propagation process, the consequent node is evaluated in each of the contexts produced by phase I. That is, the value for each datum involving the consequent node and one of the given contexts is re-computed using the new value of the antecedent datum. For example, in figure 5, a new value would be computed in each of the three resulting contexts. Once this has been done, the propagation process is re-started once for each node/context pair. Thus all daa in the data base that are affected by the change in the initial antecedent datum will be updated.

An important effect of this propagation process is the downward inconsistency that might arise. If the initial datum that is changed corresponds to any node other than a bottom node, the database will be inconsistent downward. That is, this modified datum has no longer been inferred from its antecedents. The propagation process does insure upward consistency in the database.

V. <u>QUESTIONING</u>

The propagation process discussed earlier can be thought of as the forward chaining mechanism of the AGNESS system. AGNESS also provides a backward chaining mechanism, namely the questioning process. To initialize this process, the user gives the system the initial focus (a node and context). This focus is used as the goal that the system is working towards. If the focus datum is marked as askable, the system will ask for its value. If the user supplies the value, the value is recorded, the propagation process initiated, and the questioning process stops. If the user does not supply a new value, the system generates the antecedent data of the focus datum. The antecedent data act as the initial set of candidate questions. The questioning process then proceeds in three phases: Merit calculation, value retrieval, and candidate updating.

A. <u>Merit Calculation</u>

In this phase, the system calculates Merit values for each datum in the candidate set that is marked as askable. These Merit values represent the ratio of the expected change in the focus datum over the expected cost of suppling the candidate datum. The calculation is based on the partial derivatives of the assignment functions on the path from the candidate's node to the initial focus node. The theoretical foundation of Merit has been presented in previous papers and will not be presented here [5].

B. <u>Value Retrieval</u>

The system now chooses the candidate datum with the highest Merit value and asks the user for the value of this datum. If the user supplies a value, the system will initiate the propagation process to update the data base to be consistent with the new value. The user may, however, not know the answer to the question. In this case, no propagation is performed.

C. <u>Candidate Updating</u>

In this phase of the questioning process, the system updates the set of candidate data. The updating is done as follows. If the user answered the question, that datum is simply removed from the candidate set. However, if the user did not answer the question, the antecedent data are generated and added into the set of candidates. By doing this, the system has added to the possible questions the data that will allow a value for the skipped datum to be computed.

At this point, the system returns to the Merit calculation phase. This questioning process halts when either the user requests the system to stop, or when the Merit of the best available candidate datum falls below a predetermined threshold. The use of the Merit scheme directs the system towards asking next an optimal question. Thus, if the questioning process must be terminated before all the questions are asked, the time has been used to optimal efficiency.

VI. USER INTERFACE

The AGNESS system is capable of using two different user interfaces. The system can run in a batch interface, reading and executing commands from a file. This user interface is useful for applications that require many independent runs of the AGNESS system. The second and more interesting interface is a menu-driven user interface. Through a series of menus, the user can pick activities, nodes and contexts with a minimum of typing. AGNESS provides a variety of facilities including construction and explanation.

A. Construction

This facility provides the expert with a user-friendly interface for building the computation network. Through this facility, the expert can add, delete, and modify nodes, datum functions, and edges. Also, the expert can examine the structure of the network through a graphical representation, examine the values of a given set of nodes and contexts, and essentially access and manipulate all elements of the network and data base. This activity allows the expert to experiment with slight additions to the network, test the need for some nodes by temporarily removing them from the network, and even experiment with new and different inference methods.

B. Explanation

One of the most important features of an expert system, and thus an expert system shell, is the explanation facility. In early systems, the explanation usually took the form of answering "why" a question was being asked. This form of explanation gave the user a way to follow the reasoning of the system by viewing the series of rules the system used to reach its conclusions. We use the term "user" to refer to the end user, the domain expert, and the knowledge engineer. By exposing the user to this information, the designers increased the confidence in the final system. Many early systems also included a second explanation query, namely "how". This query was designed to allow the user to ask questions about the conclusions of the system. As the system listed the conclusions, the user was allowed to ask "how" each conclusion was reached.

Designing an improved set of explanation queries has become an increasingly important area of research. Most of this research has moved away from the simple notion of presenting the rules used by the system towards more sophisticated explanation systems. Some researchers are concentrating on the causal relationships that exist in the domain knowledge [7]. Another branch of research on explanation concentrates on the natural language feature of the user interface. Although this names only two of the numerous areas of research on explanation, it does serve to illustrate that the explanation of tomorrow's systems will be sophisticated, depending on more than the simple rules used by the expert system in reaching its conclusions.

In an expert system shell, the value of the explanation facility is increased significantly. A shell is designed to be used over and over again in various domains, and as such should include friendly and useful interface facilities. Most of the expert system shells available today host an impressive graphic and menu-driven user interface. However, these shells have seemingly forsaken explanation as part of their elaborate interface. For example, some of the most visually sophisticated and useful expert system shells such as ART, KEE, and LOOPS do not include explanation as a feature [3]. These systems do provide a means of programming the explanation function into the final system, however it is not provided as part of the actual shell. Other shells that do provide explanation facilities such as INSIGHT, M.1, and Personal Consultant only provide the basic "why" and "how" queries that were found in the earliest systems [3]. Although the state-of-the-art explanation facilities are far to complicated and domain sensitive to be reasonably included in today's expert system shells, the set of permissible explanation query types should be much larger than the simple "why" and "how" that is found today.

AGNESS provides eight types of explanation that give the user a more complete set of queries. These query types also give the system designer rule tracing and debugging facilities. Each query type uses only slightly more knowledge than the standard "why" and "how" queries, and yet significantly increases the information available to the user . Each has been designed to improve the explanation available to the user while using only technology that is already in use in most of today's expert system shells. Thus, these query types are not meant to challenge the state-of-the-art explanation technology, but instead, to act as an intermediate set that can be included in commercially available systems with little or no increase in the cost or complexity. Also, each of these query types can be added to existing expert systems directly without significant effort.

In the AGNESS system, the explanation queries fall into three categories: queries about the past, queries about the present, and queries about the network structure. Each of the eight queries is recursive. That is, as the system answers the query, the user is allowed to ask for explanation of the answer.

Queries about the past allow the user to ask the system (1) why a datum was derived, (2) where a datum was used, and (3) how a datum was computed. These three queries give the user the ability to move through the database, examining the features that lead to certain data.

With respect to the present, AGNESS provides the user with the ability to ask (1) why a question is being asked, (2) where a datum will be used, and (3) how a datum will be computed if left to the system. By using these queries, the user can follow the reasoning process of the system as it happens. They also provide the user with information about the effect of answering a question, or leaving the computation up to the system. Thus the user can always ask for an explanation of the system's actions, and an explanation of the results of the user's actions.

AGNESS also provides explanation about the structure of the network. The user or the expert is allowed to ask for (1) the antecedents and (2) the consequents of any node. This information is displayed graphically, and gives the user or expert an explanation of the structure of the network at the node level. This explanation facility can be valuable when the expert wants to verify part of the network. It is also valuable to the user as it gives an explanation in terms of general propositions as opposed to specific instances.

This set of explanation facilities allows the system to be easily understood and followed, thus increasing the user's confidence in the system's conclusions. A more detailed description of the query types and their importance will be presented in a forthcoming paper.

VII. CONCLUSIONS

The AGNESS system has been tested on domains ranging from an expert system on wine to the real world problem of analyzing treadmill exercise ECG test results. In both domains, the system proved to be elegant and simple to use. The expert system written to analyze ECG test results has achieved a level of performance higher than that of the human doctors that were being used to analyze the data [4].

AGNESS represents a significant step forward in generalized expert system shells. AGNESS can reason both forward and backward, can use any combination of numeric and non-numeric data, and can use any well defined inference method required by the user. The system provides an excellent range of explanation queries far and above other expert system shells. The explanation query types give a full and rich explanation of the relationships that exist in the knowledge base. By including these query types as a basic feature, expert system shells can patiently wait for the technology of tomorrow while remaining useful today. The AGNESS architecture provides efficient implementations of expert systems by sharing constructs such as nodes, edges, and datum functions whenever possible. The computation network used in AGNESS allows only relevant rules to be considered during propagation, thus reducing the work needed in finding the rules that can be fired. Also, AGNESS uses the Merit scheme to handle the questioning of the user to insure that the most important questions are asked first in case the questioning period must be prematurely terminated.

VIII. PLANS

Even though AGNESS has shown to be extremely useful as an expert system shell, we are still working on more features and improvements to make the system even better. Some of the things we are investigating include new explanation facilities that contain more knowledge than the current system, new network configurations to help make the propagation process even faster, and improvements to the Merit scheme used during questioning. The interface is also being revised to include more graphical representations, and better help facilities.

ACKNOWLEDGMENTS

We would like to express our deep gratitude to the members of the Monday Night Expert Systems Workshop. Without their guidance and encouragement, the AGNESS system might never have been implemented.

REFERENCES

[1] Shortliffe, E.H., *Computer Based Medical Consultations: MYCIN.* New York: Elsevier, 1976.

[2] Duda, R.O., Hart, P.E., Konolige, K., and Reboh, R., "A Computer-Based Consultant for Mineral Exploration," *Technical Report; Final Report, SRI Project 6415, SRI International,* September, 1979.

[3] Harmon, P., and King, D., "Expert Systems: Artificial Intelligence in Business," *John Wiley & Sons, Inc.,* 1985.

[4] Slagle, J.R., Long, J.M., Wick,M.R., Matts, J.P., and Leon, A.S., "Expert Systems in Medical Studies - A New Twist," *Proceedings of the Conference on Applications of Artificial Intelligence, SPIE,* 1986.

[5] Slagle, J.R., and Hamburger, H., "An Expert System for a Resource Allocation Problem," *Comm. of the ACM,* September, 1985.

[6] Duda, R.O., Hart, P.E., and Nilsson, N.J., "Subjective Bayesian Methods for Rule-based Inference Systems," *National Computer Conference,* 1976.

[7] Swartout, W.R., "XPLAIN: a System for Creating and Explaining Expert Consulting Programs," *Artificial Intelligence* ***21,*** 1983.