PARAMENIDES:

Web-Based Expert System for Problem Solving and Diagnosis

by

Karl Cassar

Supervisor: Dr. John Abela

Co-Supervisor: Dr. Kevin Vella

A project submitted to the
Faculty of Information and Communications Technology
in partial fulfilment of the requirements for the degree of
Bachelor of Science in Information Technology (Hons.)

Department of Computer Science and Artificial Intelligence
University of Malta

June 2009

Contents

Abstract

Imagine a world where human expertise can be stored in a database and can easily be accessed by anyone through the Internet. Knowledge has always fascinated humans. Humans that possess vast knowledge in a domain are extremely valuable and experts in certain domains are quite scarce. However, human experts have various limitations:- they are not permanent (they die, retire), they can only service a limited number of users, they suffer from moods, and cannot always explain their reasoning.

An expert system is a computer program whose main aim is to serve as the role of an expert. Similar to an expert, it asks questions related to a goal defined by the user, in order to reach a conclusion. It has various benefits:- It can be reproduced easily, can service a very large number of users, can clearly explain its reasoning and is permanent. The knowledge contained by an expert is stored in a knowledge base as a set of rules, based on an ontology. The expert system's job is to reason about the stored knowledge and come up with conclusions as an expert would. It also takes care of uncertainty elements in the reasoning, and be able to produce results based on their expected belief.

The aim of *Paramenides* is to bring expert systems to the public and create a globally-accessible online resource for expert knowledge and problem solving. The main objective is to create a web-based expert system (www.paramenides.com) that is accessible publicly via the Internet where both users and experts can meet to share knowledge. Experts can define knowledge bases which are mapped to their domain of expertise, while the expert system can help users in solving/diagnosing problems. This is all done through an easy-to-use interface that does not require the users to be particularly computer-savvy to use.

P*aramenides* works by using an inference engine that implements both backward chaining and forward chaining modules. It works on knowledge bases that are created by users themselves. A knowledge base consists of a series of IF ... THEN rules, which are then used for problem solving and diagnosis. Backward chaining involves setting a *goal* and working backwards in order to reach a conclusion. Forward chaining involves processing facts inputted by the user to find out what can be the problem. It also uses *certainty factors* to allow elements of uncertainty in its reasoning and thus be able to provide a list of reasons based on the sorted by their most probable outcome. An explanation module allows the system to provide a reasonable explanation on how it came to such a conclusion, in a visual manner. This allows the user to easily follow its reasoning, thus increasing its credibility.

The knowledge bases defined in the expert system can be created by the users. Thus it is difficult to achieve test results on all possible knowledge bases. However, if correct rules are defined in the knowledge base, the system mimics an expert's reasoning with 100% similarity. The results are quite subjective since even in real-life, reasoning and conclusion can vary greatly from one expert to another.

To my family and friends,

for their help, love and support throughout my life

Acknowledgements

I would like to thank all the people around me who have given me their support during this project.

First of all, I would like to thank my supervisor, Dr. John Abela for his supervision, valuable advice and support during all the development stages of the project, from the initial concept when it was just an idea, up to the final stage.

I would also like to thank my family for their support during the stressful times and late-night periods, especially during the final stages of the dissertation. I would like to thank my friends for their support and help in the evaluation stage, through which I would not have been able to come up with such conclusions and allowed me to get a different view from a non-developer's perspective.

I would also like to express my appreciation and gratitude towards all teachers, lecturers and professors who contributed to my education and have helped me reach this stage as without them it clearly would not be possible. I would also like to thank my family for always stressing out the importance of education and helped me make it a very important aspect of my life.

Most of all, I would like to thank my father, Raymond Cassar for giving me the initial push into computer programming by handing me a Visual Basic book at the age of 10 which enabled me to develop the passion that I now have for computer programming and computing in general. For this I would like to thank him dearly.

Finally, I would also like to thank my twin brother, Mark Cassar, who is also a fellow student in the same course for all the help and support he has given me from as long as I can remember.

List of Tables

List of Figures

List of Equations

List of Algorithms

1 Introduction

1.1 Overview

The Internet can be considered one of the most important inventions of the 20th century. It has brought with it a tremendous change in how humans communicate with each other, and today we can find a great deal of information online.

However, most of this information is not structured in any manner and one needs to sift through all relevant and irrelevant information to find what he needs. This is especially true when a user uses the internet to find solutions to a certain problem. The main process involves searching on broad keywords, and constantly redefining keywords to try and close in on your subject.

The idea of this project is to create a *wiki*[1] of expertise where human experts can enter knowledge in the form of rules that can be understood by a machine. Users can use its search engine to find conditions (goals, or problems) and the system can guide them to find solutions to the problem, similar to what a real-life expert would do.

This builds closely on the very successful model introduced by *Wikipedia*[2]. Although the model has its disadvantages[3], in general it has proven to be very effective. However, even though it contains information on almost every imaginable topic, the articles are unstructured and cannot easily be used by a computer program to interpret

[1] A *wiki* is a "*a type of website that allows the visitors themselves to easily add, remove and otherwise edit and change some available content"* (79)

[2] http://www.wikipedia.org

[3] For example, some people are sceptical to use information from a publicly editable resource,

information from them. *Paramenides* (www.paramenides.com) aims at creating a *wiki* of human-expertise through a web-based infrastructure where human knowledge can be stored in a machine-readable format and can be used by the general public for problem-solving and troubleshooting.

1.2 Knowledge Systems

From prehistoric times, human beings were always fascinated with knowledge. Knowledge was always considered a symbol of power and as time went by, man has always tried to find ways and means to transfer knowledge from one generation to another. It is one of the main factors that distinguish us from animals, and was always an integral part of human civilization. In olden times, the only means to pass knowledge to other generations was through word of mouth. However, knowledge in human beings can easily be lost[4]. Several methods have been developed to try and capture knowledge in a more permanent form. Some such systems include:

- One of the earliest inventions to store knowledge was **writing**. The earliest discovered form of writing was found in Mesopotamia, circa 3100 B.C. Farmers used to record the amounts of different crops on soft clay tablets. This served as the initial stepping stone for recording knowledge, and was the first step to try and separate knowledge from humans to a time-less form. (1)
- The different pieces of information created by writing needed to be grouped in some way, and the concept of a **book** was created, where related pieces of information were stuck together to make it easier to store and keep track of.

[4] Especially in olden times, when nations were constantly at war with each other, knowledge could easily be lost due to death, the person being captured as prisoner, or some form of mental illness like *dementia*. Apart from these *physical* notions, knowledge in human beings is also dependent on various other factors like mood and willingness to cooperate.

- **Traditional Libraries**[5] were created as a means to organise books together according to their title, category, or the information they held. Libraries contained some form of indexing[6] to easily locate the required information. These places also served as place of learning and power, and nations of olden times strived to have great libraries. One classical example of this is the *Library of Alexandria.*[7]
- With the invention of the computer, came **Computerised Libraries**. Also known as databases, they serve the same core purpose as a traditional library but in a much faster way.
- The **Internet** is the current state-of-the-art in information sharing. Through the use of search engines[8], people all over the world have a vast repository of information at their fingertips, and it has totally redefined the notion of sharing information.

These different inventions have all been extremely important in transferring knowledge & information from one generation to another. As the amount of information increased, and the Internet has contributed greatly to this knowledge explosion, another problem emerged. Sometimes, having too much information, like having nothing can also be a problem. Most of the information on the Internet is stored in an

[5] In fact, the word *library* is derived from Latin '*libri'* for book and the word itself means 'a collection of books'

[6] Like labeling book shelves, book tags, etc

[7] The library of Alexandria was one of the largest libraries of the ancient world. There is no known index of its contents, but some ancient scripts mention a gift of 200,000 scrolls, giving an indication that the total is a much larger number. There are various exaggerated stories about it, including some of which told of having all visitors surrendering any books, scrolls or written media to the library. Unfortunately, it did not survive the test of time, and is thought to have been destroyed by fire. A new library, *Bibliotheca Alexandrina* has been created as a commemoration to the mentioned ancient library. (78)

[8] According to 2008 statistics, there are approximately 8 billion searches every month on internet search engines like Google, Yahoo, MSN and AOL.

unstructured manner. It is left up to the user to search for the correct information. While search engines do a quite good job at retrieving information, the user is still left with an enormous amount of information that he must sift through to determine any relevant information. The user still needs to extract *knowledge* from the information. A more efficient means to store structured knowledge was felt, and this led to **Expert Systems.**

1.3 Expert Systems

There are various different definitions of how a human can be defined an **expert** in a field. One of the most generally accepted definition would be (2):

"*A person with a high degree of skill in or knowledge of a certain subject.*"

As Peter Jackson states, an expert must be made up of these characteristics: (3)

- Possesses knowledge
- Knowledge must be focused on a *specific domain*[9]; having a random collection of information does not define expertise.
- Problem solving capabilities – The expert must be able to directly solve problems, and/or can act as technical support to problems that may arise in his domain of expertise

As computers developed and continued to progress, various scientists began experimenting in building intelligent machines to try and mimic the human brain. **Expert Systems** are computer programs whose purpose is to imitate the real-life traits of a human expert and problem-solving skills. It may completely or partially

[9] By specific domain, one implies one or more areas of knowledge. It is difficult (impossible?) to be an expert in everything, and as the proverb states *"Jack of all trades, master of none"*.

fulfil such a requirement, and it can also serve as a medium to aid in the decision-making process. Although an expert system is still a computer program, it approaches the problem from a different aspect than a traditional *procedural*[10] computer program.

1. 3. 1 Characteristics of an expert system

- Be able to reason about specific domain(s)
- Must be made up of a *knowledge base*[11], and be able to *declaratively*[12] reason about it. It must not revolve around a set of step-by-step algorithms, but must be able to alter its reasoning according to its programmed *rules*[13] in the knowledge base.
- It must be able to solve problems using an approximation method, and cannot be guaranteed to succeed, as opposed to traditional computer programming models which are programmed against an algorithm and have verifiable outputs (4)

[10] Procedural programming (also known as *imperative programming*) is a programming paradigm, which involves specifying each step of execution that the program is to take. It is the most *traditional* form of computer programming.

[11] A knowledge base is a database of rules (see footnote 13) related to one specific domain

[12] Declarative programming is a programming paradigm in which the users specifies what he wishes to accomplish, rather than how.

[13] A rule is one logical statement when reasoning about a problem. It contains antecedents, and consequents; Antecedents are its inputs, while consequents are its outputs. Thus, a rule can be defined as "If [Antecedents] Then [Consequents]", and is one of the basic elements that make up an Expert System

1. 3. 2 Components of an expert system

An expert system must be made up of distinct components, in order to serve its function. These components are:

- A *Knowledge Base* which contains knowledge about a specific domain, as a set of rules that define the reasoning of an expert as a series of logical steps
- An *Inference Engine* which acts like the mind of the expert system. It performs reasoning on the rules defined in the knowledge bases, and tries to derive answers to ultimately provide solutions for a given goal. It retrieves facts from users by asking questions that revolve around the rules in the knowledge base and the current goal.
- *A Working Memory,* which acts as the memory of the expert system. It stores any facts inputted by the user, as well as any derived facts during the reasoning process.
- An *Explanation System* that is capable of showing the reasoning process to the user by keeping track of which rules have fired and why. This can greatly aid in increasing the user's credibility of the system.

Figure 1-1 shows a conceptual view of all the components that make up an expert system, and how they interact with each other.

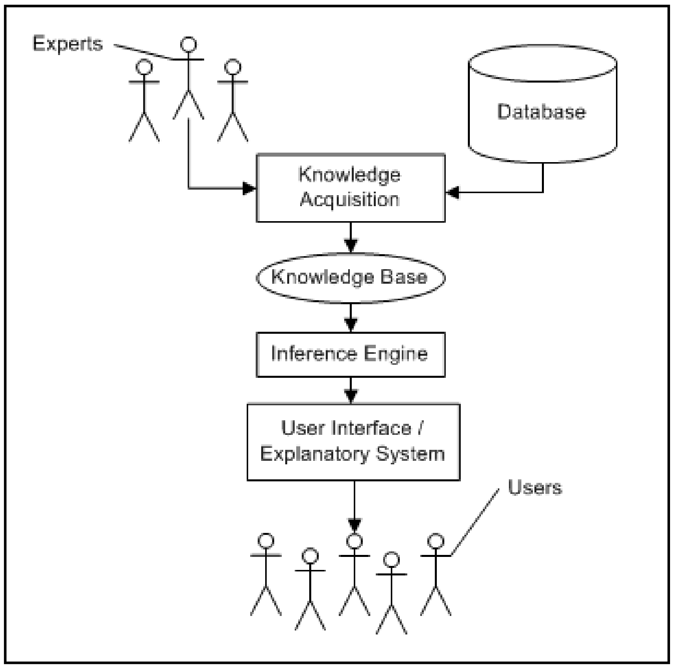


Figure 1-1: Main components of an expert system

1. 3. 3 Pros and Cons

As in almost everything, expert systems contain both positive and negative aspects. Outlined below are some of these aspects:

Pros

- More consistent in its results, and never forgets to ask a question; mood is not a factor in a computer program / expert system
- An expert system is always available. It never retires, dies, or quits
- Can scale to a multitude of users; A real-life expert is limited to the amount of 'users' it can concurrently cater for.
- Can contain knowledge of several experts in one repository
- Can provide a consistent explanation of its reasoning; Knowledge tends to be a very objectionable topic, and most experts are not able to logically provide a structured flow of their reasoning. This can also depend on their mood.

Cons

- An expert system is not creative; It cannot solve problems or provide assumptions for new problems, or for problems that it has no related rules in its knowledge base
- A human expert can adapt faster and more easily to change

1.4 Aims and Objectives

Expert systems have been a constant AI research topic from mid-20th century. Expert knowledge has always been considered extremely valuable and useful, and an expert system tries to perform the function of an expert, without the expert. However, although various expert systems have been created, most of the expert systems are off-limits to the general public. Most expert systems in production today are used by large corporations, and even though there are various open-source[14] expert systems and inference engines, they have their own limitations. Most of them have a limited set of documentations, are relatively difficult to integrate/use, or may not serve exactly the requirements. In short, although people might have heard of expert system, most users have never used an expert system or dreamed of ever using one.

The main aim of this project is to design and implement an expert system that:

- Is accessible easily to a large user-base through the use of the Internet.
- Has an easy-to-use user interface that does not require users to be particularly computer-savvy to use it.

[14] To name a few: CLIPS (http://clipsrules.sourceforge.net), JBoss Drools (a.k.a Drools, http://www.jboss.org/drools), NxBRE (http://www.agilepartner.net/oss/nxbre), Drools.Net 3.0 (http://droolsdotnet.codehaus.org)

- Is not limited to one particular domain, but serves as a generic expert system that can include various distinct knowledge bases to cater for large amount of domains.
- Able to perform both backward and forward chaining
- Can easily explain its reasoning by a graphical representation to make it easier for users to understand the conclusions

The expert system will be able to reason about different types of problems. It should be able to solve problems of the type:

- **Troubleshooting**: A user has a known problem, and wants to find the cause of the problem. The system will ask a series of questions to try and find out the possible cause of the problem. E.g. His car is not starting, and the user wants to diagnose what might be the problem.
- **Diagnosis**: A user has some certain symptoms occurring, and wants to find out what might be the consequences. The user will *feed* the symptoms to the expert system, and it will output a list of possible conditions that he may have. The user can even select a certain condition, and have the system focus on that specific problem, by asking him more questions related to the problem (as in troubleshooting). E.g. He is feeling lower-back pain, sometimes feels dizzy, and has stomach problems. The system outputs the possibility of conditions he might have.
- **Selection**: A user needs to find the most suitable object for his needs. The user will input his ‘needs’, and the system will output a ranking of objects that might suit him best. For example, a user wants to choose a restaurant. He will input his ‘wants’, like that it must be by the sea, and specialises in Mexican food. The system will then output the list of restaurants it thinks would best suit him.

1.5 Overview

In this chapter, we have given a brief overview of the need for knowledge that is felt by the human community. We have given a brief summary about the history of knowledge and how it started being represented, and described the roles of a human expert. Direct parallels were made between a human and a computer expert and given a high-level overview of what the project is about. The rest of the dissertation is organised as follows:

Chapter 2: This chapter is a short chapter that briefly goes through topics that can help the reader understand the main problem of the project, but are not directly related to the inner workings of an expert system. It can serve as a brief introduction to the topics further explained in the following chapter.

Chapter 3: In this chapter we go over the theoretical aspect of an expert system and topics of direct concert to the project. We analyse research material gathered throughout the years and provide various insights on methodologies used and arguments both in favour and against. These should allow the reader to understand in more detail the intricate workings of an expert system and its components.

Chapter 4: This chapter is aimed to give a brief outline of how we intend to develop the system, and how it is intended to function. It lays out the plan and which methodologies we will be embracing throughout its development. It will compare various approaches and provide information on why we chose such an approach.

Chapter 5: In this chapter we will explain in great detail how the system was developed. We delve into detail about its inner workings for each of its components, and provide step-by-step representation of its main algorithms.

Chapter 6: This chapter provides a thorough evaluation of the system by perform both system testing and human evaluation, and builds arguments on the results obtained. From these results we could them analyse the various strengths and weaknesses of the system and any possible improvements that could be implemented in the future.

Chapter 7: In this chapter we will provide a brief summary of the outcome of the project, and also provide a series of possible improvements that could come in the future and any current known limitations of the system.

2 Fundamentals

This chapter shall cover material that is directly or indirectly related to Expert Systems. Most of this knowledge will be assumed in later chapters. Mostly this chapter shall cover information related to logic systems, expert systems, ontologies, and the various stages of knowledge management.

2.1 Logic

The basic fundamental aspect of every expert system is *logic.* Logic is a broad term that encapsulates various methods and structures, whose aim is to represent human thoughts and reasoning[15]. It serves as the basis for inference and arguments, and there are many different forms of logic systems.

2. 1. 1 Informal Logic

This is the most basic form of logic and involves dealing with logic on a non-standard/non-formal basis. It is the study of natural language arguments, and can be considered a form of logic that is very subjective. Informal Logic also involves the study of *fallacies*[16], and was the logic system used by ancient Greek Philosophers like Plato, in their dialogues.

[15] The word *logic* in itself is derived from the Greek *logike,* which means "possessed of reason, intellectual, dialectical, argumentative" and *logos¸* which is the Greek word for "thought, idea, argument, reason", among other representations. (http://en.wikipedia.org/wiki/Logic)

[16] Fallacy: "A misconception resulting from incorrect reasoning" (60)

Informal logic, as its name implies does not follow any form of formal or rigorous method of proofing statements. There is no means to prove the invalidity of an argument, and can pose very difficult problems to work with in an algorithmic manner.

2. 1. 2 Propositional Logic

Propositional logic is a *formal* logic system. It is the study of the conjunction of various axioms[17], propositions, statements or sentences in order to derive more complicated sentences, as well as define logical relationships and properties.

A statement in propositional logic can be defined as a fact that can have a truth value:- either *true* or *false*. For example, the following are all valid statements:

"Plato was a Greek Philosopher"
"Everyone who can walk is not dead"

Statements portray information on the world in the current context, and are the premise for more complicated logical calculations. A statement can be made up of more than just one statement, by using the conjunction operators **AND** ($\wedge$) and **OR** ($\vee$). These two operators are the most basic logical operator, and are used in almost every logical system. A 'proposition' is most of the time used interchangeably with the term 'statement', and is used to express two different statements that portray the same meaning, for example: *"The moon orbits the Earth"* and *"The Earth is orbited by the moon"* are considered to be the same proposition. Apart from the basic operators **AND** and **OR**, propositional logic also uses various other operators like negation (**NOT,** $\neg$), implication ($\Rightarrow$) and bi-implication ($\Leftrightarrow$). The **not** is different from the

[17] Many of such words in logic systems derive from Greek words. This is because Greek philosophers were the first known documented records of formal logic. The word *axiom* comes from the Greek word *axioma,* which means, "to require". Considering

other operators because it works on one single argument, and is thus considered a *unary* operator. (5)

These operators are used to reason about the statements, and one can use them to obtain information from statements, based on their truth values:

> *"If it is raining or there are black clouds in the sky, then take an umbrella"*

One can use propositional logic to deduce that if one of the two statements "*It is raining*" or *"there are black clouds in the sky*" is true, then the consequent is true, i.e.: *"Take an umbrella"*. Truth tables are used to define the inputs and outputs of each of the mentioned operators. The figure below shows the truth tables for AND and OR operators.

A	B	A ∧ B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A ∨ B
T	T	T
T	F	F
F	T	F
F	F	F

Table 2-1: Truth tables for AND and OR operators

Propositional logic deals with statements as a whole, and considers a simple statement to be indivisible. It does not involve the study of logical relationships between parts of a sentence that are smaller than a 'simple statement'. An example:

> *"John is the son of Mary"*
> *"Mary is the president of America"*
> *"Therefore, John is the son of the president of America"*

To deduce the third statement in the above arguments, one needs to be able to perform some identification on the subjects and actions as defined in the premises.
Propositional logic considers statements to be indivisible, and such the statement ‘John is the son of Mary’ is not divided into smaller parts, and no further information can be concluded about it.

However, formal proofing of statements is not the aim of this chapter, and for further reading on propositional calculus one is referred to (5) and (6).

2. 1. 3 Predicate Calculus

Predicate logic is an extension to propositional logic. Propositional logic deals with simple statements/propositions, while predicate logic (also known as *first-order logic*) also deals with *predicates* and *quantification*.

Unlike natural languages, predicate calculus is a non-ambiguous language used to define relationships and logical properties of statements, propositions and predicates. One of the main limitations of propositional logic was that one had to enumerate all elements in a set, and in many cases this can be very inconvenient and tedious. It provided no means to group elements based on their characteristics.

A **predicate,** denoted by *P(x)*[18], is a statement, which works upon an element (or a group of elements) and is used to group elements that contain certain properties. A predicate is similar to a function in a computer program that returns a true/false value, and is also called a **propositional function**.

[18] A predicate can act on more than just one element, and a predicate acting on say 3 elements would be written as P(x,y,z)

Some examples of predicates would be:

Red(x)	–	*x is red*
Son(x,y)	–	*x is the son of y*

The set of elements that is defined when acted upon by a predicate is defined as:

$$\{x \mid P(x)\}$$

Equation 2-1: Set defined by P(x)

Another important notion introduced in predicate calculus is that of **quantifiers**. There are two types of quantifiers – Existential quantifiers ($\exists$) and Universal quantifiers ($\forall$). Each quantifier acts on a *universe of discourse*[19], which serves as the set of all elements on which predicates work upon. (7)

Universal Quantifier

The universal quantifier is the operator that denotes all elements in a given set. In the English language, this quantifier could be expressed as *"For all x that contain a certain property, P(x)"*. An example of a universal quantifier could be:

$\forall x : \boldsymbol{wheels(x)}$ –*All object x that have wheels*

[19] The *universe of discourse* can be a set of real numbers, all human beings, all living creatures on earth, the set of students in a university, etc. Although the elements in the universe are normally not included with logic statements, it should be obvious from the context, or else defined.

Existential Quantifier

Whereas the universal quantifier acts on all elements in the universe of discourse, the existential quantifier acts on select group of elements based on a certain criteria. In English, an existential quantifier could be expressed as *"There exists at least one element such that P(x)".* One example of existential quantifiers in a statement could be:

$\exists x\ \exists y\ Son(x, y)$ - *There exists persons x and y, such that x is the son of y*

The use of quantifiers enabled predicate calculus to address some limitations of propositional logic. Consider the classical example:

"All men are mortal"
"Socrates is a man"
"Therefore, Socrates is mortal"

While this is entirely valid, as already explained previously, propositional logic provides no means to arrive to the conclusion that *"Socrates is mortal".* There is no means to represent this form of logic using just the operators defined in propositional logic.

On the other hand, this can be easily translated into predicate calculus statements. The same argument could be defined as:

$\forall x(Man(x) \rightarrow Mortal(x))$ - *For all x, an x of type Man is also of type Mortal*

There is no need to go in further detail on the formality of such logic systems. For further reading, one can refer to (8) which contains further information on predicate calculus, and also its usage in the *Jess* expert system.

Predicate Logic in Expert Systems

This type of logic is very important in implementing an expert system. It would be very difficult to create an expert system which has no notion of predicates and types, and also would make it very inefficient. Predicates enable the expert system to 'know' facts about the objects in the universe of discourse, and can allow rules to be defined more specifically to only apply to certain types of objects.

Predicates can allow the expert system to operate in an *object-oriented* manner, by defining classes[20], and sub-classes. It could allow for example defining relationships like: *"A mammal is an animal"* and *"A Human is a mammal".* From these facts, one can easily deduce that a human is also an animal. Thus, properties that apply to an animal also apply to a human. This can prove very useful especially when defining rules because it greatly reduces the amount of rules needed to express the same concepts or relationships. A rule that applies to an animal need only be defined for an animal, and it will automatically be taken to apply for any mammals, or humans in question. There is no need to define 3 separate rules, one for each object-type (animal, mammal, human).

2. 1. 4 Boolean Algebra

Boolean algebra is logical system invented by George Boole (9) in mid-19th century. It expands on propositional logic, and provides an algebraic-like system for dealing with logical statements. It was the first such system, and has proven to be extremely popular in digital electronics. It allowed electronic computers to count using only two possible values, *true* or *false*, or 0s and 1s.

[20] A *class* is considered to be a specific type of objects with certain similar traits

All operators defined in propositional logic were defined in Boolean logic, and various properties applicable to Boolean logic were defined. Some of these include:

- Associativity: $a \vee (b \vee c) = (a \vee b) \vee c$
- Commutativity: $a \wedge b = b \wedge a$
- Distributivity: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
- De Morgan's Laws: $\neg(\mathrm{a} \vee b) = \neg\mathrm{a} \wedge \neg\mathrm{b}$

For further reading on the topic and other properties, the reader is referred to (10).

2. 1. 5 Uncertainty in Logic

All the previously mentioned logic systems deal with discrete values for logic variables. However, as one can easily see from real-life example, this is often not the case. Most of real-life problems that experts need to solve deal with a certain element of uncertainty. There are various extensions to the mentioned logic systems to enable them include probabilistic factors in the reasoning. Each different methodology has its own pros and cons, but these will be explained in more detail in page 82. (Chapter 3, Uncertainty Management)

2.2 Ontologies

Now that we have described the different logical systems that exist, we can move on to explain what an ontology is[21]. There are various different definitions of what is an ontology, and it can have different meanings, based on the context.

[21] *Ontology*, unlike most of the other terms is not directly derived from Greek. The oldest known similar word is the Latin *ontologia.* However, the context of an ontology was used by ancient Greek philosophers, and we have records of philosophers like Parmenides, Plato and Aristotle defining ontology as a conceptual schema describing objects and relationships between them

Nicola Guarino, (11), defines an ontology as:

- A philosophical discipline
- An informal conceptual system
- A formal semantic account
- A specification of a “conceptualization”
- A representation of a conceptual system via a logical theory
- Vocabulary used by a logical theory
- Specification of a logical theory

As one can see, although most of the interpretations are very similar, interpretation 1 is very different from all others. So one may ask, what exactly is an ontology with regards to knowledge bases and expert systems?

An ontology can be seen as a formal specification of all the objects and properties related to a domain, and relationships between them. Ontologies are used in various computer-related fields, which include:

- Artificial Intelligence
- The Semantic Web
- Knowledge Representation

To make up all the properties of a domain, an ontology is made up of various different components.

2. 2. 1 Components of an ontology

- **Individuals** are the most basic element in an ontology
- **Classes** encapsulate a related set of objects, and define what an object is made up of
- **Attributes** define characteristics about objects (or classes)
- **Relations** are ways in which objects are related to one another

By no means should the above list[22] be considered to be an exhaustive one. However, it mentions the elements that are of interest when considering expert systems.

To define such ontologies, one normally uses several standard ontology languages like *OWL*[23], *DAML+OIL*[24] and *RDF*[25]. The reader is referred to (12) and (13) for further reading on these ontology languages.

2. 2. 2 Why is an ontology needed?

An ontology makes it much easier for systems to know exactly what they are reasoning about, and to avoid ambiguity. There are many real-life examples that can lead to ambiguity, and words can have totally different meanings based on the context. For example, the word *flip-flop* might mean entirely different if you are talking to a person on the beach or whether in an electronics laboratory. If an ontology is about beach or

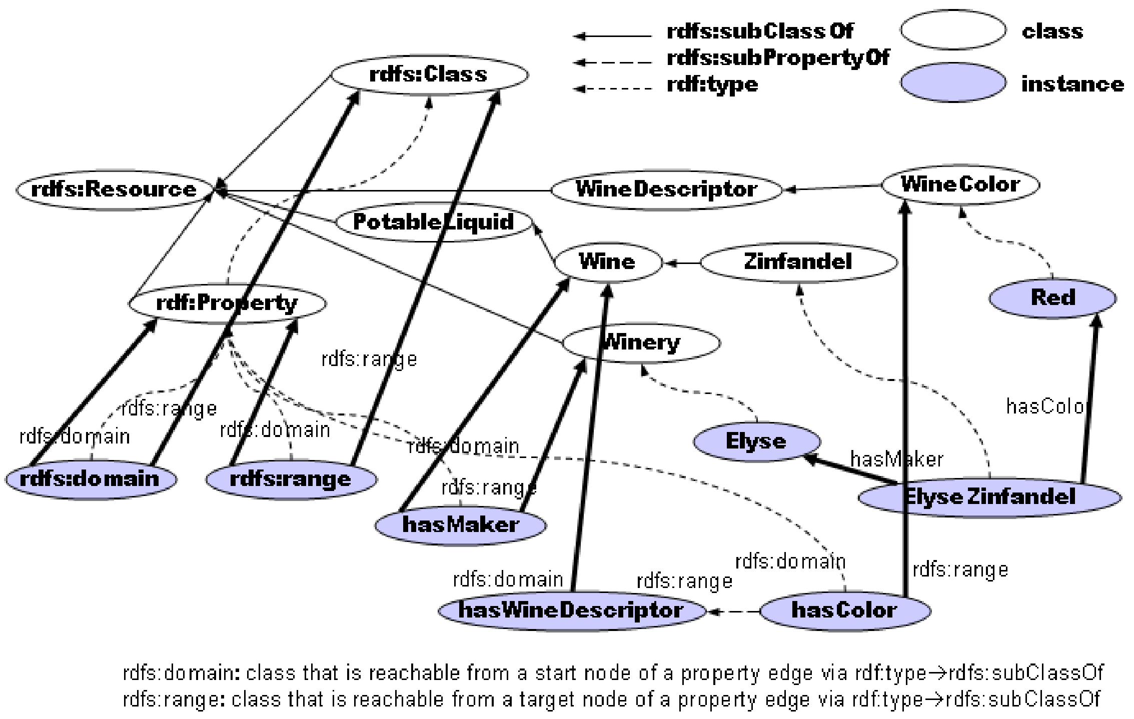
[22] Adapted from (61)

[23] Web Ontology Language: Although the acronym should essentially be WOL and not OWL, the acronym OWL was proposed as a more easily pronounceable acronym and would be better suited for designing logos. An owl is normally associated with wisdom and honor, and quoting Guus Schreiber, *"Why not be inconsistent in at least one aspect of a language which is all about consistency?"* (62)

[24] A language combines features from DAML, and OIL, hence the name. The acronyms stand for DARPA Agent Markup Language and Ontology Inference Layer/Ontology Interchange Language respectively

[25] Resource Description Framework

leisure, *flip-flop*[26] would imply a type of footwear, whereas in an electronics laboratory it would be referring to a type of electronic circuit. Figure 2-1 shows an ontology describing wine, as defined in RDF.



rdfs:domain: class that is reachable from a start node of a property edge via rdf:type→rdfs:subClassOf
rdfs:range: class that is reachable from a target node of a property edge via rdf:type→rdfs:subClassOf

Figure 2-1: A diagram showing a wine ontology (Taken from (14))

[26] The mentioned electrical circuit was not always called a *flip-flop*. Initially it was called the *Eccles-Jordan trigger circuit*. The name was later derived from the sound produced by a speaker connected to such a circuit. (Adapted from (63))

2.3 Knowledge Management

As time goes by, knowledge management is gaining more and more importance. Companies are realizing how important it is to know as much as possible about the information that they already have. Knowledge management allows enterprises to make the maximum use of such knowledge.

In this age of information, knowledge is an extremely important asset. However, having enormous amounts of unstructured knowledge can pose several problems. Sometimes, work is repeated simply because it is impossible to make 'sense' out of the available knowledge, even though such knowledge exists and could have been used in the first place to reduce costs & time.

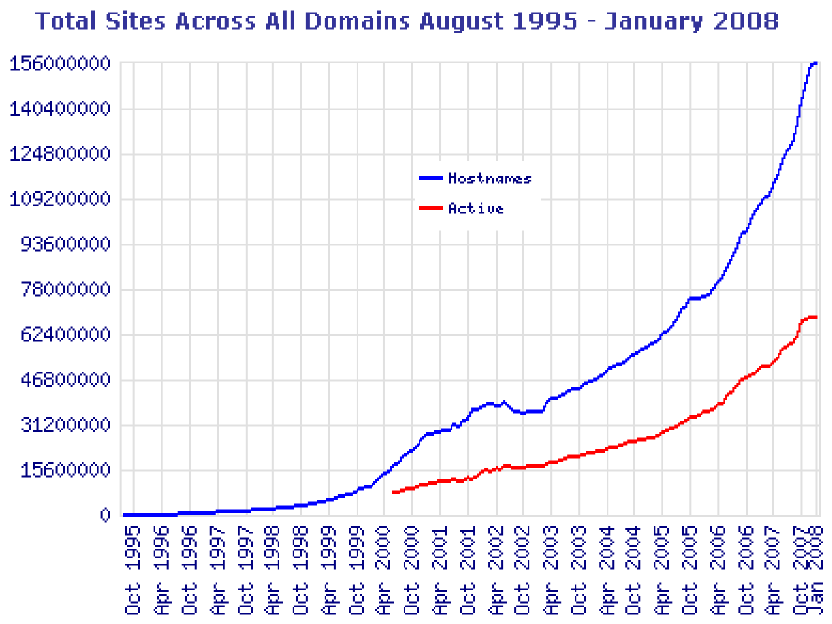


Figure 2-2: Internet growth during 1995-2008[27]

[27] Taken from http://www.labnol.org/wp/images/2008/02/total-number-websites.gif

This problem is greatly affecting the Internet – As more and more users turned on the Internet as a global means of communication, sharing knowledge, and researching for information, the amount of information available has exploded. Figure 2-2 shows the exponential growth of the internet. It is estimated that the size of the internet doubles every 5 years[28], and will continue to expand. One cannot imagine the vast amount of information available. Although search engines facilitate the work of parsing through the enormous amount of data, it can still be considerably difficult to find what you need. A major drawback for search engines is that web pages are created in a mark-up language aimed for human consumption. This makes it difficult for search engine to parse the content of the web pages. The **Semantic Web** is an extension to the World Wide Web whose aim is to make the internet more computer-friendly. Tim Berners-Lee[29] describes in (15) the need for having such semantics about the services and information exposed on the internet. The Semantic Web it is still in its infancy, and there is a long way to go before we will start using it on a daily basis.

Knowledge management is an integral part of any expert system. The process can be divided into various sections, as explained in the following.

[28] Research from a group of Chinese students has shown that the Internet can also be considered to follow Moore's Law, but doubling every 5.32 years over the 18 months stated for CPU transistors by Moore's Law. (http://www.physorg.com/news151162452.html)

[29] Tim Berners-Lee is also considered the founder of the *World Wide Web*

2. 3. 1 Knowledge Acquisition

Knowledge acquisition is an integral part of the knowledge management process. Its main aim is to retrieve information from different data sources, and convert it into a form that is more useable by the system. This is normally done directly via a user interface used directly by the expert, or by a *knowledge engineer*, who converts the knowledge of the expert into a more suitable format for the system.

Huge amount of information is very easily accessible around us and due to this same reason, it is difficult to use this information. This problem is called **infosmog**. Infosmog is when the actual information that you are looking for is hidden underneath a wealth of information that you don't actually need.

Expert systems rely greatly on **tacit knowledge**[30]**.** Tacit knowledge is that knowledge which is available only to an individual and only that individual holds that particular knowledge. Tacit knowledge is highly personal and is built upon experiences, ideas, values and emotions. This could be stored in one's own mind, available subconsciously through cultural knowledge, habits, learnt knowledge etc. Knowledge acquisition aims at exposing this type of tacit knowledge and converts it into **explicit knowledge**. Explicit knowledge is knowledge that is available to everyone. The aim of an expert system is to convert tacit knowledge from experts into explicit knowledge in the forms of rules, which can then be used by a much larger user base.

[30] Tacit knowledge can be considered knowledge that people are not aware of, or knowledge that cannot be easily explained (transferred) to other people. It is knowledge that cannot be easily explained in writing. A classical example of such knowledge is that of knowing how to ride a bike. Although for most people that can ride a bike, it comes naturally, it is not easy to explain to a person how to balance yourself on a bike but can be learnt more through personal experimentation.

There are various methods to obtain information from experts, such as:

- Interviews
- Questionnaires
- Learning by observation
- Learning by being told / explained to

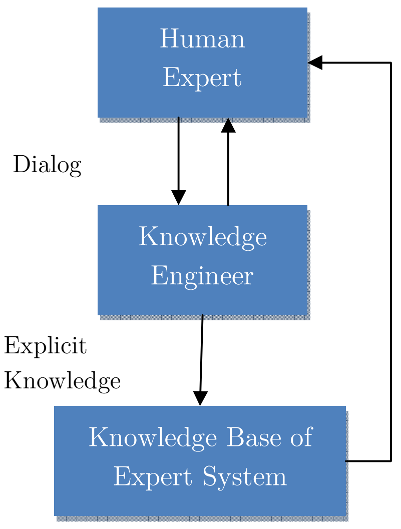


Figure 2-3: Knowledge Acquisition Process[31]

[31] Adapted from (65)

2. 3. 2 Knowledge Representation

It is important that the knowledge acquired is stored in a form that enables the expert system to work efficiently on. R. Davis et al. (16) define knowledge representation as having five different roles, one of them being “a medium of human expression”. Knowledge representation is a crucial aspect in artificial intelligence. In expert systems, knowledge is stored in a *knowledge base*, which in turn is stored as a set of rules. Each rule implies one logical statement, and all the rules together can be considered the mind-map of one or more experts.

One main advantage of an expert system is that it can store the combined knowledge of several experts. These rules, together with an ontology, define the *long-term memory* of the expert system.

What needs to be represented?

There are several notions that need to be expressed in a Knowledge Base. The main ones are *objects, facts,* and *relationships.* The objects in a knowledge base define the types in the ontology, and the domain of the expert system. Facts are knowledge about the objects, like *‘The car is red’*. Relationships on the other hand define the links between objects, such as *‘A car is a machine’*, which defines a relationship between *car* and *machine*, such that the car inherits any properties of a machine.

2. 3. 3 Knowledge Inference

Acquiring and representing knowledge is a very important part of any expert system. However, the expert system needs to reason about this knowledge. Knowledge alone does not constitute an expert system. A classical expression defined by Nicholas Wirth[32] with regards to computer programming is:

Algorithms + Data Structures = Programs

Joseph C. Giarratano et al. apply this same expression for expert systems, converting it to: (17)

Knowledge + Inference = Expert Systems

Inference is the process of deducing information from known facts to arrive at other premises. Reasoning in an expert system is entirely dependent on the rules found in the knowledge base. The purpose of an *inference engine* is to mimic the reasoning process of an expert, and thus can automatically expand the knowledge base through known facts inputted by the user and production rules stored in the KB. An inference engine can produce both valid and invalid inferences. An expert system cannot easily distinguish between what is a valid inference or not (apart from a contradiction).

An example of an invalid inference would be:

A bird has two legs

A cat is a bird

∴ A cat has two legs

[32] Swiss computer scientist, best known for designing several programming languages including *Pascal.* He holds a Turing Award for developing a sequence of innovative computer languages.

We can easily know that this is not true, because we have a good idea of what the notion of a *cat* is, and that a cat is surely not a bird. However, it is a perfectly legal argument for an expert system[33].

There are mainly two types of inference in an expert system – **Backward chaining** and F**orward chaining**. Backward chaining is a method of reasoning that starts with a goal (hypothesis) and works from it backwards (hence the name) until a solution is found. It will only fire the least amount of rules possible, and is an ideal method of reasoning for troubleshooting problems when one knows the problem, and wants to find out what is the cause.

On the other hand, Forward Chaining is a method of reasoning which considers any known facts and tries to obtain as much information as possible by firing rules. Forward chaining is a form of *greedy* solution, which will use all rules available. This method is useful when one wants to diagnose a problem, for example in medical diagnosis. One would not know the condition that he has before hand, but knows only of the symptoms that he is experiencing.

Forward chaining can give an indication of what might be the condition based on the information the user inputs the system. Explaining these types of inference is not the scope of this chapter, and they will be dealt in more detail in page 54 (Chapter 3, Inference), including methods of reasoning employed under uncertain circumstances.

[33] An invalid/incorrect inference is also known as a *fallacy,* and can be defined as "a misconception resulting from incorrect reasoning" (http://wordnetweb.princeton.edu/perl/webwn?s=fallacy)

2. 3. 4 Explanation

One very important aspect of any expert system is the ability to clearly explain its reasoning process. The user must be able to easily follow up why certain questions were asked, and how the expert system came to that conclusion. This greatly increases the credibility of the system, as it will not be considered a system that produces a magical answer without any reason.

This same explanation capability is the same way that human experts can explain their reasoning[34]. Various factors make this feature extremely important:

- The system can be critical (as in a medical-diagnosis expert system) and human life or property can depend on it. Thus, the system has a great deal of responsibility to come up with reasonable answers just like an expert would. Having an explanation system enables the user to justify its answers, and serves as a sanity check of the reasoning for humans.
- An explanation system can aid a lot the *knowledge engineer* in debugging the system, and can be extremely useful in the development stage. Misunderstandings or human mistakes can be detected more easily if a good explanation clearly explains its reasoning of why it came to such conclusions. It allows the knowledge engineer to verify its accuracy. It is next to impossible to understand the reasoning of such a system just by looking through the rule base, especially as rule bases tend to get quite large and contain many chained[35] rules. Also, rules are not fired sequentially as they were entered, and an expert system

[34] Not every human expert can clearly explain their reasoning for coming to such a conclusion, and that is one of the most important aspects to consider in *knowledge acquisition*

[35] By *chained* rules, one implies rules whose consequents refer to other rules.

acts like a *parallel program* by firing rules according to the circumstance, and in some cases even simultaneously.

Therefore, an explanation system is very important both for the end-user, and for the developer (knowledge engineer) (18). An explanation system must be able to answer two types of questions for its users. These are:

- Why did the expert system ask that particular question?
- How did the expert system reach that conclusion

An inference engine must keep track of any rules fired and the sequence in which they were fired. In the case of backward-chaining inference, it must also keep track of any goals and sub-goals generated during the progress. Answering the above two questions can be considered a *tree-search* on the rules fired and goals.

The explanation facility of an expert system depends on the intended function of the expert system. Some different methods of explanation could include:

- Listing all rules in chronological order of execution, so that the user can follow up the reasoning process step-by-step of the expert system
- A tree-like structure of the goals and generated sub-goals, so that the user can easily follow up how did the expert system reach a certain conclusion, and why certain questions were asked

2.4 State of the art in Expert Systems

Considering the vast amount of information available at this day and age, artificial intelligence and expert systems are a continuous research process in the hope that they can help us make more sense of the enormous amount of knowledge around us.

The thought of being able to build intelligent machines came about around the middle of the 20th century. Initially, progress was moving at a very fast rate and computers could be able to solve "astonishing" problems like solving algebra and speaking English. This even led some notable scientists like H. A. Simon to say that "Machines will be capable, within twenty years, of doing any work a man can do" (19).

As one can clearly see, these predictions did not come true, as researches failed to recognise the difficulty in solving certain set of problems. After the boost of AI in around the 1960s, came the first *AI Winter*[36]

The potential of Expert Systems was first seen in around the 1980s and it gave another great boost to AI. Artificial Intelligence has yet to be able to create a computer that is really 'intelligent'. One can easily debate on how you can define a machine as *intelligent*, but that clearly is not the scope of this dissertation. Expert Systems are one of the most successful areas in the field of Artificial Intelligence.

Enterprises and companies are investing money in creating such expert systems, to be able to store tacit knowledge contained by their experts in a more useful form. This also enables the experts to focus on new research, and on more productive work than problem solving. Lately, more interest is being taken once again in this topic, and we

[36] AI Winter is considered to be a period of slowness in research, and lack of interest in Artificial Intelligence. This came mainly due to various disappointments and lack of funds.

have seen the emergence of ontology definition languages like OWL, DAML+OIL and RDF, research into the Semantic Web, data mining, clustering, behavioural analysis and much more. A drawback to existing expert systems is that most such systems encapsulate business logic, and many companies generally do not publish such information because competitors can take advantage from it. The aim of this project is to try and bring expert systems more into the usage of the general public, and also aim to remove the scepticism around them. (20)

2.5 Summary

In this chapter, we have introduced some fundamental concepts behind the theory of expert systems that are the basis for topics that are to be covered in the next chapter. We have identified the various forms of logic that are used for representing knowledge and conditions, giving slightly more prominence to Boolean Algebra which is the basis for any expert system. We have also given a brief outline of uncertainty in logic; defined the concept of an ontology, what it consists of and its importance in an expert system. Also a brief outline of the steps involved in the knowledge management process was given, splitting the process into three main steps: Knowledge Acquisition, Knowledge Representation, and Explanation. Finally, we have also given some background information on knowledge inference, which is one of the main aspects of an expert system that is expanded in much more detail in the following chapters.

3 Background

This chapter shall provide an overview of how expert systems evolved, the theory behind such systems, how they work in practice, their limitations and their strength.

3.1 Internet

Before we start to explain in more detail how an expert system, we will explain why the Internet was the medium of choice for such a project.

3. 1. 1 Brief History of the Internet

Computers have been around now for around a century, however when computers first started appearing they were nowhere similar to what we expect them to be/do today. Computers were lonely machines that worked solely on their own without any form of communication. However, communication is an essential process for computing in general, and this sparked lots of research in networking.

There are various networks that led to the global network we nowadays know as the **Internet**. These include ARPANET, NSFNet, X.25 and UUCP. With the invention of TCP/IP, much of these networks were merged and the network became worldwide. The internet is nowadays an integral part of almost everyone's life, and internet usage has grown exponentially in the past few years, and will continue to do so in the near future.

Initially, the internet served more as a large repository of information, where users used search engines to search for information, and find what they need. The 21st century saw the emergence of so called *dynamic web pages* and *web-applications,* and this has continued to give a great boost to internet usage in general. (21)

3. 1. 2 Why the Internet?

Applications developed over the internet have many advantages over traditional applications. A major advantage of web applications is that they require **no installation.** The user simply has to log on to the website, and he can start using the application instantly. Also, there are no hardware incompatibilities, OS settings, etc.[37]. Having no installation also makes the application available anywhere, and you users can use the same application anywhere they like (assuming they have an internet connection)

Traditional desktop applications also have a problem of synchronizing versions when deployed over a network. If an upgrade is performed on the software, the patch needs to be done on each and every computer where the application is installed. This is not the case for web applications. The update is done to the web server and the application is automatically updated when the user logs on.

The application is also able to cater for a **vast population** of internet users. This is one of the main points of why this project is being deployed over the internet. Having access to all these human resources allows users to contribute much more easily to the system, and the ultimate aim would be to create a global repository of knowledge bases

[37] Although a classical problem when developing applications on the internet is having browser-incompatibility instead of operating system incompatibility. This is especially a problem with users who are using old, non-standards compliant browsers like Internet Explorer 6 and Netscape 4. Still one can argue that it is much easier to change a browser, than to change your operating system.

that are freely accessible by anyone who has access to the Internet. Figure 3-1 shows the latest information available on internet users worldwide. Current population stands at approximately 1.6 billion users. (22)

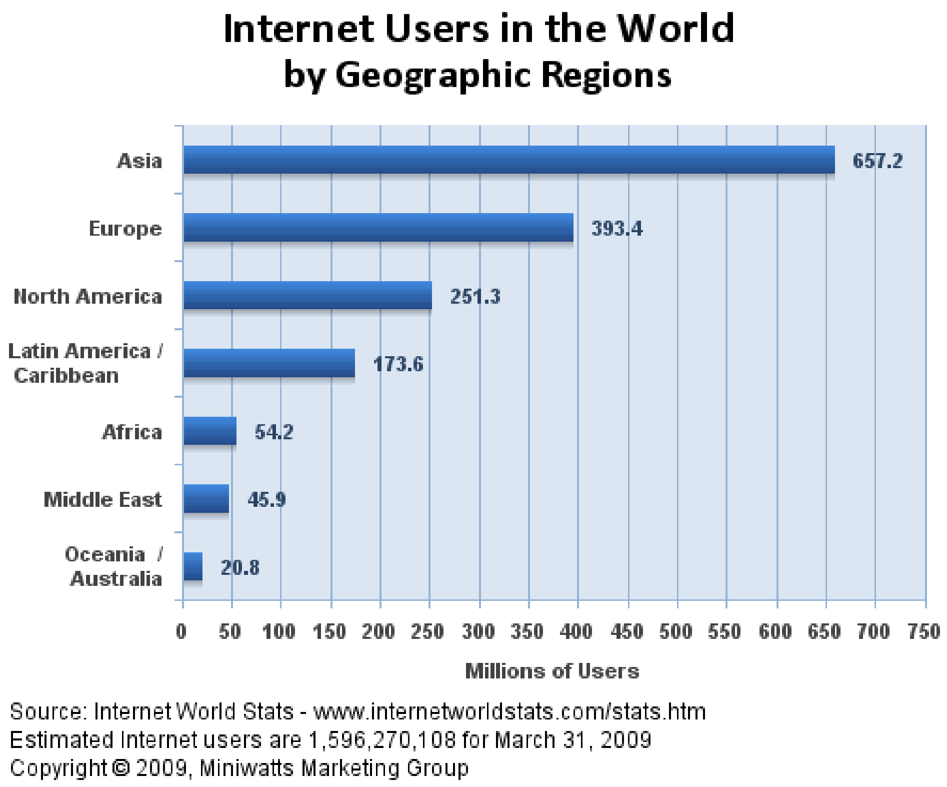


Figure 3-1: Internet users as of March 31, 2009[38]

[38] Taken from http://www.internetworldstats.com/stats.htm, which contains various statistics on internet population, broadband penetration, and world internet usage

3.2 Philosophical Basis

3. 2. 1 What is knowledge?

There are various definitions of what is knowledge. However, it is not that easy to define exactly what knowledge is, and it may mean differently from one person to another.[39] Oxford English Dictionary (23) defines knowledge as:

i. Expertise, and skills acquired by a person through experience or education; the theoretical or practical understanding of a subject
ii. What is known in a particular field or in total; facts and information
iii. Awareness or familiarity gained by experience of a fact or situation

Knowledge has always fascinated human beings. *Epistemology*[40] is the study of knowledge, and is mainly concerned with the nature, structure and origins of knowledge. It can be classified into mainly two types of knowledge:- **a priori** and **a posterioiri**. Figure 3-2 shows a brief outline of how epistemology can be classified[41]

A priori knowledge is knowledge that is known independently of any experience. It is the considered to be universally true, and is not subjective (assuming there is no contradiction). Mathematical laws are an example of such knowledge. **A posteriori** on the other hand is knowledge that requires a certain kind of experience to be known, and can also be verified. For example, if one sees a black sheep he can conclude that sheep are black. However, if later he sees a white sheep, he has to revise his knowledge. Most 'a posteriori' knowledge can be verified using the sensory experience of human beings.

[39] Joseph C. Giarattano in (66) points out that "Knowledge, like love, is one of those words that everyone knows the meaning of, yet finds hard to define or even feel the same way about"
[40] Epistemology like most other words is derived from Greek *episteme* and means 'theory of knowledge'
[41] Adapted from (67)

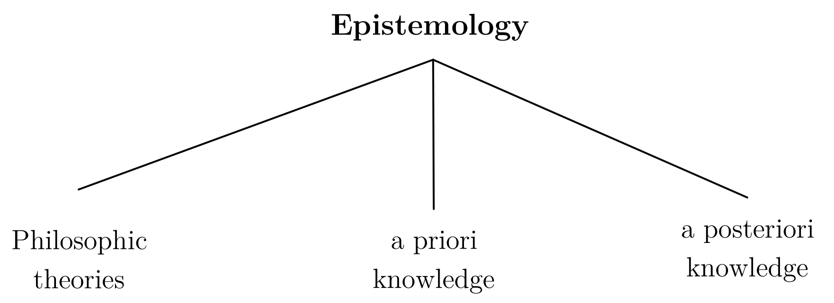


Figure 3-2: Some different types of knowledge in epistemology

Most knowledge that is used in problem-solving uses a heuristic approach, and the expert cannot say with certainty what the problem is. However, based on experience, an expert can give an indication of the likelihood of the cause of the problem, and how accurate he is in his calculations is what makes the difference between how 'expert' an expert is considered. Most problem-solving can be converted into a form of *check-list*, where the expert mentally cross-checks this list to come up with a conclusion. This type of reasoning is called **case-based reasoning**, and is a very important type of reasoning used in expert-system. A. Aamodt et al. (24) define case-based reasoning as "the ability to solve new problems by remembering a previous similar situation and by reusing information and knowledge of that situation". This type of reasoning involves a cyclical process. Experts solve new cases from which they learn knowledge and information. These solved cases are used and can be 'retrieved' from memory when need arises.

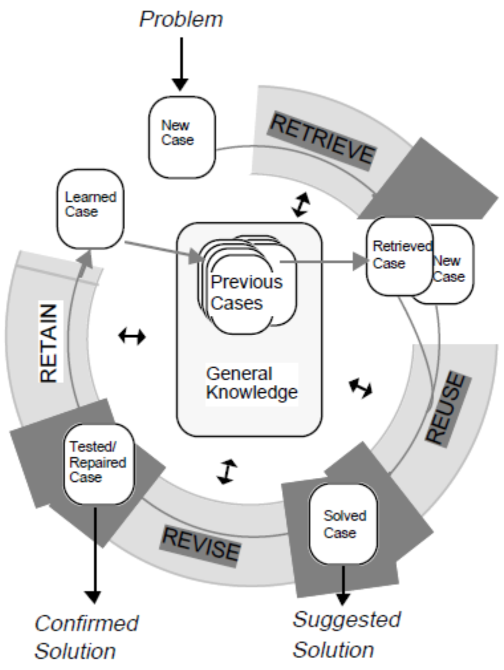


Figure 3-3: The case-based reasoning cycle[42]

[42] Reproduced from (68)

Information around us can be split in various hierarchies. Figure 3-4 depicts the so-called 'Pyramid of Knowledge'.

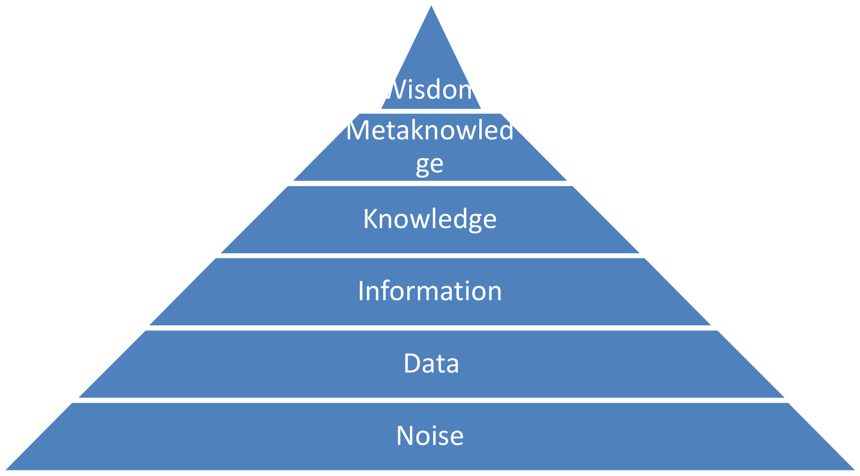


Figure 3-4: Pyramid of Knowledge[43]

- **Noise:** Data that has no apparent meaning
- **Data:** Information that could potentially be useful
- **Information:** Could potentially be useful for knowledge
- **Knowledge:** Rules about using information
- **Metaknowledge:** Knowledge about Knowledge, contains information about the rules
- **Wisdom:** Using knowledge in a beneficial way

[43] Adapted from (69)

3. 2. 2 Common Sense

Common sense is a very debatable subject in epistemology. One cannot easily define what common sense is, and it is very difficult for computers to work with such knowledge. Common sense is defined as a method of reasoning that uses the beliefs of a person as opposed to rigorous scientific methods. For example, if a person near you drops a glass of water from a table, and you are 6 meters away, you will not jump away to try and avoid it. This is because you know that the glass shards and water will not travel that distance. You do not use any formulas to find out exactly how much the glass shards will travel, or how much will the water spread - Most people don't even know them, and if they do, it is not possible to calculate the results instantly as required before you can take action.

Users of expert systems need to understand the limitations of such systems. Although most expert systems have not been able to yet employ common sense, they still are able to perform quite well in their domains. The problem with common sense is that it has proved very difficult to express it in formal terms. One of the best known expert systems, MYCIN worked by assuming that the user was aware of its limitations, and that the user himself used his common sense to reason about the conclusions that MYCIN reached. This enabled such a system to perform reasonably well without common sense. (25)

3. 2. 3 Expertise

What makes an expert? There are various arguments on what are the factors that make a person an expert. Being intelligent or having a degree does not imply that you are an expert, and vice-versa. Even if you have a PhD or a degree in a topic does not essentially make you an expert in that field.

Experts normally have very broad knowledge about one (or more) particular domains, and have managed to acquire troubleshooting experience through training and/or practical experience in the field. Typical examples of every day experts are:

- Mechanic: An expert on cars
- Doctor; An expert on human anatomy and diseases
- Computer Technician: An expert on computer hardware and software troubleshooting

An expert is able to adapt to the situation, and can use his expertise to try and solve new problems. Case-based reasoning is used to recall past problems and come up with solutions. (26)

How can one become an expert?

One cannot become an expert by reading a degree, or simply by some form of training. Experts are considered 'experts' because of their experience in the field, their achievements, and their trouble shooting skills. Some key factors in becoming an expert in a domain could be:

- **Practice**: An expert needs to have practical experience in the field
- **Diagnosing & fixing problems**: Ability to solve problems
- **Recall Memory**: Be able to store solved cases, and use them to expand his knowledge on the subject and solve new cases (Case-Based reasoning)

3.3 Artificial Intelligence

Artificial Intelligence (AI) is the branch of computer science that aims to make machines perform intelligibly.

Barr and Feigenbaum define artificial intelligence as (27):

> *"Artificial Intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on".*

In more layman terms, AI can be considered as programming computers to do tasks that humans are currently more suited for, like picture recognition, translation, etc. Programming a computer to perform long and complicated mathematical formulas is not artificial intelligence, even though a human might find it difficult to perform. This is because computers are more suited for such type of problems, while *abstract reasoning* is not performed 'naturally' by a computer. The mentioned mathematical task would involve following an algorithm in a step-by-step manner, which can easily be done by a computer.

3. 3. 1 Weak AI

Also known as narrow intelligence, this type of artificial intelligence is when a machine tries to act as if it was intelligent; it is intelligence that does not try to surpass human intelligence, but performs a task with a limited set of abilities (normally limited to one human cognitive ability). Most of the current research in AI revolves around this area. Expert systems are considered to be a relatively successful example of weak artificial intelligence systems. A classical example of a weak AI system is the Deep Blue[44] machine. It used its enormous computational capabilities (over 200 million chess positions per second), to analyse possible moves to a great depth. It performed a tree-search on the possible states, and one argues whether that can really be considered intelligent. However, there is no clear definition of what is intelligence, and quoting Drew McDermott "*Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings".* (28)

However, the type of intelligence possessed by the machine, opposed to what was employed by the human player is clearly different. We can assume that it is next to impossible for a human player to perform anything near 200 million comparisons per second. Considering that the machine still lost some games, there is more and more research to be done into this topic before we can see machines acting intelligently, like what we nowadays only see in fiction.

[44] Deep Blue was a computer developed by IBM specifically for playing chess. It was the first machine to win a chess game against a world champion, Garry Kasparov.

3. 3. 2 Strong AI

Strong artificial intelligence, also known as general artificial intelligence is considered intelligence that closely mimics human intelligence. Its main aim is to replicate human intelligence and is an extremely complex task. Progress in strong AI has been quite limited. A machine is considered to have general intelligence if it manages to pass the **Turing test**[45]**.** This test involves having a human judge engage in a natural language conversation with two entities, a human and a machine (chatbot). If the judge cannot reliably tell which one of them is the machine, then the machine is said to be intelligent. No machine has yet passed the Turing test for any given length of time. (29)

3.4 Knowledge Representation

Before an expert system can perform any type of reasoning/inference, it must store rules in a formal manner. As explained previously, expert systems are a form of weak AI, and depend entirely on the rules in their knowledge base to be able to reach conclusions. There are various different methods for knowledge representation, and we will discuss several of them, including their advantages and disadvantages.

[45] A competition is held yearly, called the *Loebner Prize* that involves contestants submitting implementations of chatbots, and the chatbot that manages to convince the most judges that it is not a machine wins the competition. The latest competition was won by Elbot, which can be accessed online on www.elbot.com

3. 4. 1 Rules

A collection of facts in the knowledge base is not able to provide any further information. A rule is a formal representation of one logical statement, and creates a relationship between facts in the knowledge base.

A rule is made up of two parts; **antecedents** and **consequents,** and defines an English-like statement based on a minimum of 1 antecedent and 1 consequent. An antecedent is a requirement for the rule to *fire.* When a rule fires, it is implied that all its conditions have been met and thus one can assume that its consequents are true. From this we can derive that a consequent is the *output* of a rule. A rule is defined in this format:

```
Rule = IF {Antecedents} THEN {Consequents}
```

The inference engine of an expert system would then use these rules to perform either *backward chaining* or *forward chaining* inference (30). These two types of inference are explained in more detail in page 54. (Chapter 3, Inference).

3. 4. 2 Semantic Nets and Associative Networks

A semantic network is a labelled, directed graph that represents information about facts and the relationships between facts. It is also called a *propositional network*[46]. Initially, semantic nets were invented for use in machine translation of natural language. These semantic nets were used to represent words in the language, and the relationship between them. Later on, they found various other users for such networks. A semantic network can easily be visualised.

[46] As previously discussed, a proposition is a fact / statement.

The nodes represent the *objects* and *facts* of a knowledge base, while the edges represent relationships between them. The semantic web is a project that aims to create a semantic network out of the Internet, so that machines can make more sense out of it much more easily and efficiently. (31)

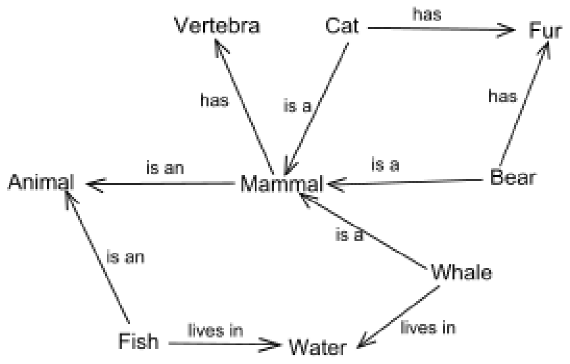


Figure 3-5: An example of a semantic network[47]

Associative networks are similar structures to a semantic network, but can contain further information about objects. Nodes in an associative network can be *connected* to other nodes to share information between them. For example, if we know that an animal can move, and we also know that a human is an animal, we can easily deduce that a human can also move. This feature is known as **cognitive economy,** and can greatly reduce the storage requirements while still getting the same amount of information. There is no need to attach the information (that it can move), to both *animal* and *human*, if we can consider human to be a sub-class of animal. Today, this is now known more conventionally as *inheritance*[48]. (32)

[47] Reproduced from http://en.wikipedia.org/wiki/Semantic_net

[48] Inheritance is a very important feature of the OOP (Object-Oriented Programming) paradigm, and is based on the same concept, that of defining a hierarchy of classes and sub-classes.

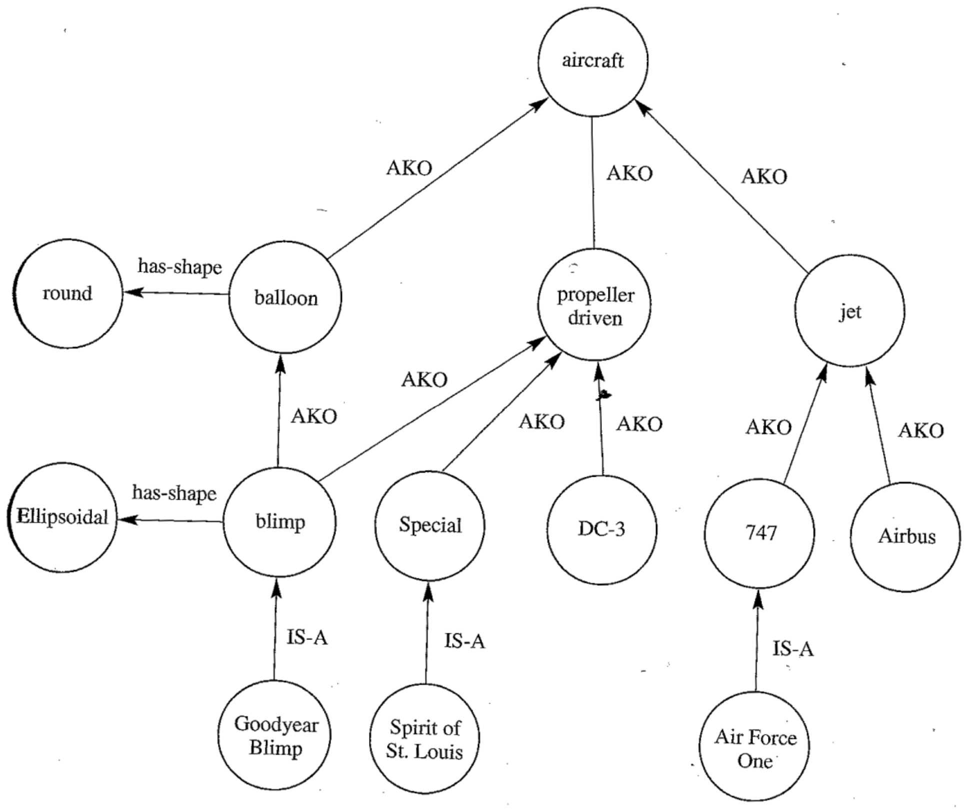


Figure 3-6: An associative network showing inheritance and relationships. AKO (A-Kind Of) relates one class to another, with the node being pointed to by the arrow being the generic class

Problems of semantic nets

- Semantic nets cannot define certain subsets of a class. For example if a node defines a 'table', is it referring to all tables or just some tables? It has no concept of an *ontology*
- A search on a semantic network can be very inefficient, especially if the result of the query is negative. If there is no result, all nodes and links of the net need to be searched and this as proven in the traditional Travelling Salesman Problem[49] is of factorial time complexity.

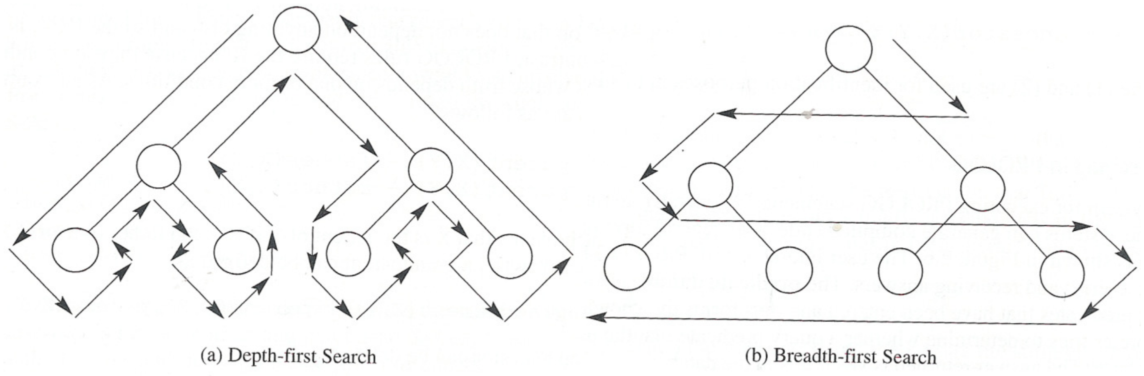


Figure 3-7: Two types of searches on a semantic net[50]

For further reading on the topic, the reader is referred to (33) which contain more detailed information on their usage, their concepts and a brief history.

[49] Travelling Salesman Problem is a classical NP-problem, which involves finding a solution for a sales man who wants to travel to all N-cities exactly once, and ending at the same place where he started.
[50] Adapted from (75)

3. 4. 3 Conceptual Graphs

Conceptual Graphs are graphs representing logic, based on the *existential graphs* developed by Charles Sanders Peirce (34), and *semantic networks*.

These types of graphs contain a formal specification of objects (nodes) and relationships, and build upon the limitations of semantic networks. Objects (also called *concepts)* can be either *generic* or *individual.* As their name implies, generic objects are objects that refer to any object of a certain type, for example a concept could define a *chair.* An individual concept would specify in more detail what the object is, for example *a wooden chair*. (35)

Conceptual graphs can also be represented using predicate logic. However, it is much better to understand the problem using a visual diagram.

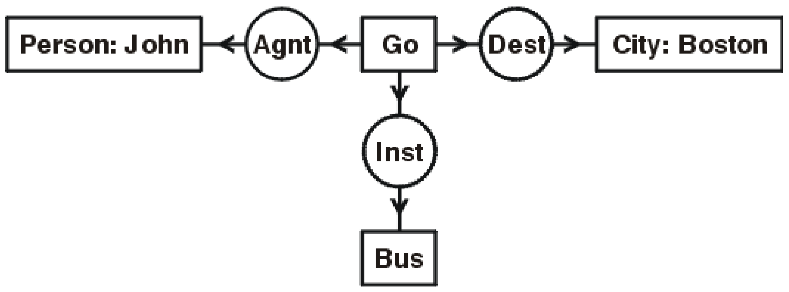


Figure 3-8: A conceptual graph

The conceptual graph in Figure 3-8 shows the logic involved in defining the statement *"John is going to Boston by bus"*. In these types of graphs, concepts are defined as rectangular boxes, and the diagram includes both *generic* and *individual* concepts. The individual concepts are the concepts that have *names* (John, and Boston). The circular nodes define relations between the concepts.

The graph also depicts relations (*Agnt*-Agent, *Dest*-Destination, *Inst*-Instrument) [51] between the concepts. The conceptual graph indicates that the person named John is an agent of some instance of going, the city Boston is the destination, and a bus is the instrument. (35)

3. 4. 4 Object-Attribute-Value Triples

Object-Attribute-Values[52] (OAVs) are a form of general-purpose data structure used for knowledge representation. It is more suitable for defining properties (attributes) about objects, and their possible values. Such properties are similar to the ones defined by a *HAS_A* relationship in a semantic net.

Sometimes, most of the objects in a knowledge base have many of such relationships, that it is more efficient to store them directly as a triplet of Objects, Values and Attributes. OAV information is similar to an extension of a traditional *relational*[53] table in a database.

Object	**Attribute**	**Value**
dog	colour	brown
dog	species	fox terrier
dog	size	medium
cat	colour	white
cat	species	persian
cat	size	small

Table 3-1: An example of an OAV-triplet

This type of data structure is very useful for representing facts corresponding to antecedents of a rule. They organise facts as a collection of objects and attributes. Defining attributes for objects allows objects to *inherit* from other objects (inherit the

[51] Agent and Instrument are different kinds of effectors in Natural Language semantics
[52] Also known as Entity-Attribute-Values (EAVs)
[53] A relational table is a table stored in a database that is made up of fields (columns), and data (rows).

properties of the other object). *Frames* are another data structure used for knowledge representation and inheritance.

Sometimes, simpler objects can be difficult to classify them into an OAV triple, as it would be difficult to separate attributes from an object. In that case, sometimes the object is omitted and the data is stored as an Attribute-Value tuple. This is normally used when one is only representing a small number of objects, and inheritance is not required. (36)

3. 4. 5 Frames

Another form of knowledge representation are **Frames**. Frames are used to define typical stereotypes, and contain a *skeleton*[54] to describe an object. They are an extension to rules and semantic nets. Semantic networks can be considered a 2-dimensional structure, which is more adept at defining broad knowledge. Frames add a third dimension by allowing objects to also have structure.

These are used to represent a narrow subject in detail, and are especially useful for describing real-world objects that contain various properties, like mechanical devices. A frame is made up of both *member links* and *subclass links.* Member links define relationships between the properties of an object, while subclass links define inheritance between objects. Member links can be thought of representing a '*has-a'* relationship, while subclass links represent an '*is-a'* relationship. Frames also incorporate **slots,** which can be further subdivided into *own slots* and *member slots.* 'Own slots' are information about the class the frame is representing as a whole, while *member slots* are information about the specific instance of the class. (37)

[54] By skeleton one is referring to the properties that make up an object, without any values

Considering a frame representing the **ELEPHANT** class. An elephant can have 'own slots' describing it as 'Heavy' and 'Large'. Member slots for such a class could be 'colour', and 'weight'. This means that an object of type ELEPHANT is considered to be large and heavy, while colour and weight are attributes that vary based on the instance. Each member can have their colour and weight specified.

Inheritance

A very important feature of frames is inheritance. A concept we are used to nowadays that is very similar to frame-based inheritance is inheritance in an object-oriented programming language[55].

A class frame can have subclass links to other class frames, and this implies that the members defined in the *superclass*[56] re 'copied' also to the inheriting frame. For example, the ANIMAL frame might have the member slots 'Weight' and 'Number of Legs'. Consider the HUMAN frame which has a subclass link to the ANIMAL frame. The human is considered to also have the 'weight' and 'number of legs' members, as well as say an 'IQ' member.

A frame-based representation structure has various benefits to an expert system and its inference engine. The frame-structure allows a more powerful language to describe objects, and can perform useful inferences based on the member links and subclass links. (38)

[55] In fact, and object-oriented language is considered to be a frame-based language, and implements most of the features found in frame-based reasoning

[56] By *superclass,* we are implying the more 'generic' class in a relationship, for example: A 'human' can be considered to inherit from the 'mammal' class, and thus the mammal class is the superclass while 'human' is a subclass of 'mammal'.

3.5 Inference

Inference is the process of taking the representation of knowledge inside the knowledge base, and performing 'reasoning' to reach conclusions and/or other facts and information. The inference engine performs a series of operations commonly known as the **recognise-act cycle** or **inference cycle**. There are mainly two forms of inference: *backward chaining* and *forward chaining*. Expert systems can also include a mixture of these two, in order to get as much information as possible out of the available facts.

3. 5. 1 Backward Chaining

Backward chaining on the other hand works by setting a consequent as a goal, and asks question to the user based on the antecedents. If a user does not know the answer to a question, the system will automatically generate a *sub-goal* based on the antecedent that was not fulfilled.

Consider an example of a very simple knowledge base about cars:

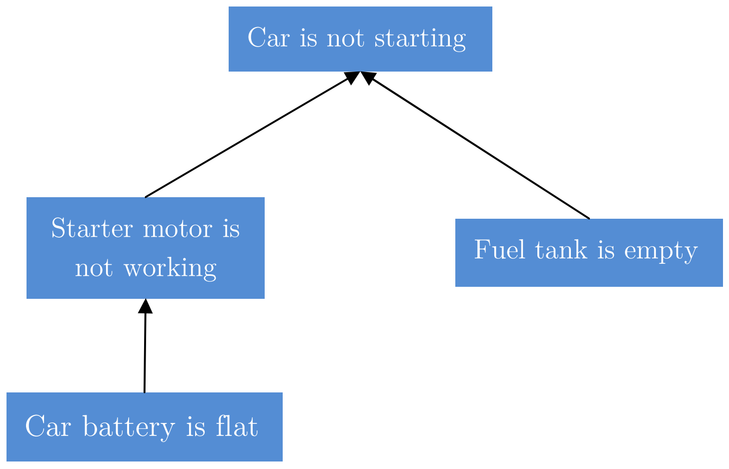


Figure 3-9: A very simple knowledge base on cars

If the user wants to know why his car is not starting, the expert system will look for any rules which have the condition as their antecedent. Lets imagine it picks "*Starter motor is not working*" first. It will ask the user whether it is true. If the user answers that he doesn't know, then that is set as a sub-goal, and it will ask whether the *car battery is flat*. The system will continue chaining rules, until a solution is found, or no possible solution is valid with the existing rules. (39)

Automatic Goal Generation

An important feature of backward chaining is the ability to automatically generate *subgoals*, based on the inputs of the user. Initially, a backward chaining inference starts with one goal condition:- trying to find an answer to the asked problem. However, most of the time it is not possible to find the solution by looking only at rules that have that goal directly in the rule's RHS as the user might not know the values for the LHS variables.

If the answer to a question posed by the user interface is answered as 'I don't know' by the user, the inference engine can still consider that value as unknown. The inference engine will keep a stack of goals and each time such an event arises, a new subgoal is created and added automatically on top of the stack (the initial goal should always remain the goal at the bottom of the stack). These subgoals can then be used to infer variables, through which it would be possible to find a solution to the initial goal.

3. 5. 2 Forward Chaining

Forward chaining inference is when the inference engine acts upon the known facts, and from the known facts reaches conclusions. This is done by firing rules, based on which rules have their antecedents fulfilled. Consider the following rule base:

```
If it barks then it is a dog
If it sings then it is a bird
If it is a bird then it has 2 legs
If it is a dog then it has 4 legs
```

Imagine one wants to find out how many legs a certain animal has, and he knows that *it barks.* The user will tell the system that *it barks.* The forward chaining module will check which rules have the known fact as its antecedent, and reach the conclusion in its consequent. Thus, it can deduce it is a dog. From the rule base, it also knows that if it is a dog, it has 4 legs. Thus, it will output to the user that the animal is a dog and has 4 legs.

The inference engine can choose to iterate through all the rules each time a new fact is either inputted by the user or deduced from rules. This brute-force approach can be extremely slow in larger knowledge bases, and faster methods needs to be defined in order to make it much more efficient. A classic algorithm for forward-chaining inference is the **Rete algorithm.**

Another method is to perform some form of indexing on the rules, so that the required rules can be found much faster without the need to iterate through the entire knowledge base every time. With the emergence of various DBMSs[57], one can create a database schema that can easily accommodate a rule-base. Indexing is a key feature of any database, and carefully selected indexes can render the searching very efficient, and removes the need of having the entire knowledge base in memory. (40)

3. 5. 3 Recognise-Act Cycle

The inference process is as a continuous cycle and stops execution to wait for input from the user or when it has come up with a conclusion. The steps involved in the cycle are: (41)

- **Matching:** This will check which rules can be activated. For forward chaining, the LHS is evaluated and keeps track which rules are satisfied. For backward chaining, the inverse is done and it keeps track of rules whose RHS satisfies the current goal. This is most intensive process of the cycle, and is the step where the Rete algorithm can be used.
- **Conflict Resolution:** If more than one rule is satisfied during the matching cycle, the inference engine must decide on which order will the rules be processed. The conflicting rules are stored in a **conflict** set. If on the other hand, no rule is satisfied, the inference will either ask for some form of input from the user, or halt execution and output the contents of the *Working Memory.*
- **Rule Firing:** A rule is fired based on the choice in the **conflict resolution** stage, and its consequents are added to the working memory.
- The process repeats itself

[57] DBMS stands for Database Management Systems, like MySQL, MS SQL, Oracle, amongst various others.

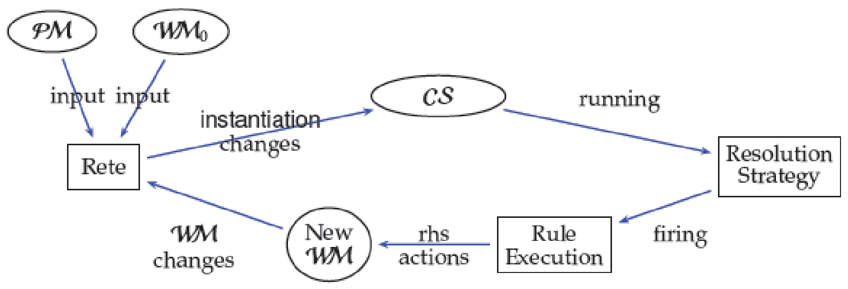


Figure 3-10: recognise-act-cycle[58]

3. 5. 4 Rete Algorithm

For even a medium-sized knowledge base with a moderate amount of rules, sequentially going through every rule is very inefficient. This renders large knowledge bases impractical and the Rete[59] [(42), (43), (44)] algorithm is an algorithm that greatly enhances the speed and efficiency of forward-chaining inference systems.

A typical working memory of a knowledge base changes comparatively slowly to the number of pattern-match cycles and rules in the KB. The Rete algorithm creates a network based on the rules that will greatly help improve the matching performance of the system. It performs a trade-off by increasing memory usage for performance gains. The Rete can be quite memory intensive with large knowledge bases, and this was especially a problem when it was first invented mainly due to the low memory available at those times. Nowadays even normal desktop computers are equipped with a relatively large amount of RAM, and it is not such an issue as it was before. However, various other implementations (both new and based on Rete) have been

[58] PM represents Production Memory, and is an alias for Knowledge Base. CS is the conflict set. Reproduced from (72)

[59] The word *Rete* is taken from the Latin word for 'net' and the same word is also used in modern Italian to mean 'network'. (Wikipedia: http://en.wikipedia.org/wiki/Rete_algorithm)

designed since the Rete algorithm was invented that require less memory. It is not the scope of this dissertation to discuss more memory-efficient implementations of Rete, and the reader can refer to (45), (46) for some of these implementations

A Rete is a directed acyclic graph whose nodes correspond to the current state of the objects in the working memory. The network is made up of two distinct parts:

- **Alpha Network (left side):** Performs constant tests on working memory elements (such as equality tests). The output of this network is stored in *alpha memories,* each of which holds information on the current set of WMEs passing the tests for a single condition.
- **Beta Network (right side):** Performs join conditions between different nodes (Working Memory) of the system. The memory of this network stores *partial rules* also known as *tokens* which match some but not all of the conditions in a rule.

The Rete network has one single point of entry, and all Working Memory (facts) are fed from this point. The updates are represented either by a *positive* token, or by a *negative* token. The positive token implies facts that are to be inserted in the memory, while a negative token implies facts that are to be removed from the memory.

Each node has a memory that stores related facts. Nodes with one-input, also called **Anodes**, contain working memory elements (WMEs) that contain one condition (pattern). If the condition succeeds (it is fulfilled), the token is passed on to all its successors.

Nodes with two-inputs, also called **Bnodes**, are used to test working memory elements that contain more than one condition. If a positive token arrives at this node, it will be stored in its local memory, while if a negative token arrives, any similar fact is removed from its memory. Once again, similar to ‘anodes’, if the condition is fulfilled, the generated token is also passed to its successor. (47)

By passing tokens to it successors, it reduces the total iterations required to check the effected rules, because the rules for which the tokens do not apply are not used. This can render the search much more efficient, to the detriment of memory space. For a more detailed explanation about the Rete algorithm and its inner workings, one can refer to the works cited in (42), (43), (44) and (47).

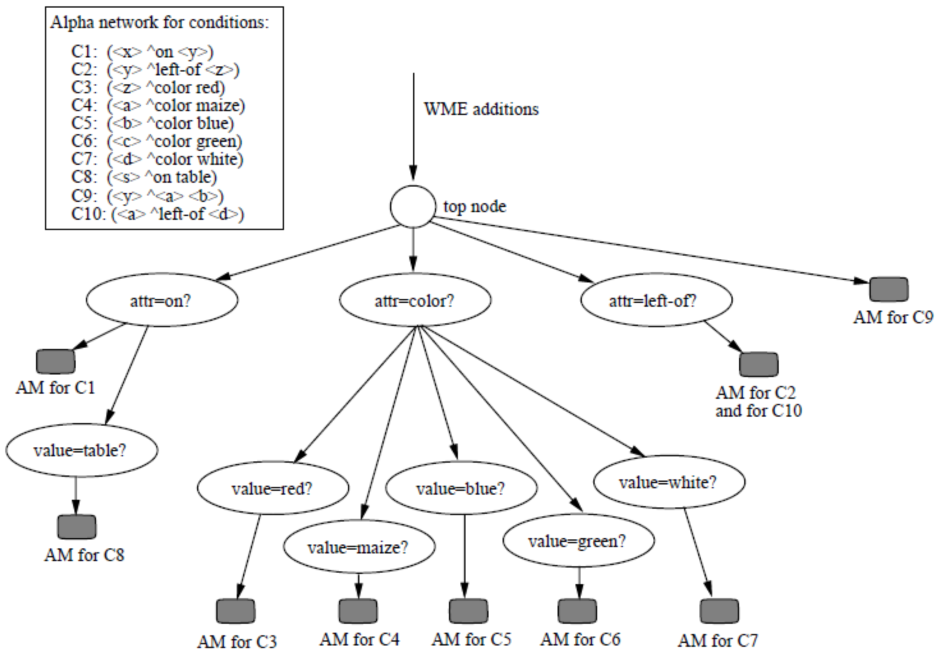


Figure 3-11: A sample Rete network, showing Alpha Memories[60]

[60] Reproduced from (73)

3.6 Uncertainty Management

Any form of intelligence system based on artificial intelligence has to operate with some form of uncertain information. The previously mentioned methods of inference all worked on one assumption: Discrete information. Conditions of rules had only two possible values[61] - True or False (or Yes & No). However, this is very different from real-world problems, and most of the time these two answers are not enough to represent practical problems. Most of the sources of uncertainty in problem solving come due to imperfect knowledge about the domain. (48)

Sometimes uncertainty allows a system to be more feasible. Uncertainty allows an expert system to come up with conclusions, without the need for all the possible evidence. This is sometimes extremely useful. For example, consider an expert system for medical diagnosis: Medical tests cost money and time. An expert system might come up with a diagnosis which is highly likely, but not certain. Although it would have been possible to perform more tests in order to increase the likelihood, it would cost money and time, and the patient might even die until the results are available.

Thus, answers should be able to reflect levels of uncertainty, and we will discuss various different methods of uncertainty methods, and their pros and cons.

[61] Three values, if you count the 'I don't know' value as supplied by the user but that is not used for inference but just for backward chaining for automatic goal generation.

3. 6. 1 Bayesian Probability

Bayesian probability extends on classical probability[62]. Classical probability deals with **a priori probability.** As already discussed in Chapter 3.2, ‘a priori’ knowledge is knowledge that one knows beforehand, without any need for experience. This type of knowledge includes game odds, playing dice, flipping a coin[63] etc. The opposite of ‘a priori’ is **a posteriori**, and is knowledge about an event based on certain experience.

Outcomes of an event can be either **deterministic** or **nondeterministic.** Deterministic are results that given the same inputs, give the same always the same results. Non-deterministic are the vice-versa, although non-deterministic does not essentially mean the same as random.

Let us consider the example of a dice. A dice has 6 possible outcomes, one of either 1, 2, 3, 4, 5 or 6. Assuming a fair dice, one can say that:

```
P(x) = 1/6, where x is a number from 1 to 6
```

This means that the probability that the outcome is a number X is one in every six times. This is a generalization, and it does not mean that if you throw the dice twelve times, it will strictly appear exactly two times. (49)

Conditional Probability

Conditional probability is the probability of an event A happening, given that another event B has already happened. It is written as P(A | B) and is read as “Probability of A given B”.

[62] Classical probability is the oldest tool for problem solving in uncertainty, and has been around for a very long time. It can be considered a quantitative way of dealing with uncertainty
[63] Considering the dice has not been worn out, the coin is fair, so on and so forth

Bayes Theorem

Bayes Theorem[64] is used to find the *posterior probabilities* for an outcome. This is defined as:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

Equation 3-1: Bayes Theorem

- P(A) is the prior probability (a priori)
- P(A|B) is the probability of A given B
- P(B|A) is the probability of B given A (posterior probability)
- P(B) is the prior probability of B

PROSPECTOR is an expert system that used Bayesian Probability for its decision making process[65]. As an example, let us take into consideration the prospect of finding oil by a mineral-exploration expert system which uses Bayesian decision making.

Initially, the probability of finding oil is split 50-50[66]. The decisions the system takes and the final outcome affects the profit or loss of the venture, thus one must try to maximise the profits based on the uncertain information at the current stage.

Let us assume that initially, one can choose to make a seismic test or not, which costs money to do. The results of the test can affect the probability that oil exists or not, and can give a better indication (albeit never with certainty). The outcome of the test affects the *posterior probability* of whether there is oil or not. However to obtain the

[64] Bayes theorem is named after the 18th century British clergyman and mathematician, Thomas Bayes.
[65] It achieved a great deal of fame as the first expert system to discover a valuable molybdenum deposit worth $100,000,000l.
[66] The prospector might believe that there is a better chance of finding oil, and splits the probability by say 60/40. However, in our case we will stick to 50-50.

results of the test, it first needs to be done. This finally all boils down to risk and probability, and one needs to decide given the current circumstances. (50)

To be able to use Bayesian probability in an expert system, one needs to define the prior probability for rules. However, most of the time this cannot be determined. Another problem is that in some domains, one cannot go for a decision simply because it has the highest probability or best posterior probability. For example, consider an expert system on medical diagnosis. If a person goes to a doctor because he is sick, the doctor does not assume that the person has a cold, because that is one of the most common diseases.

3. 6. 2 Dempster-Shafer Theory

Dempster-Shafer theory is another uncertainty management method that builds upon the limitations of Bayesian theory. While Bayesian theory is made up of probabilities for each condition or outcome, Dempster-Shafer Theory requires degrees of beliefs. These beliefs allow the expert system (or person) to show, as their name implies, how much he believes that the outcome is true. (51)

In Dempster-Shafer theory, one assumes that there is a constant set of events that are mutually-exclusive[67]. All the events are called the **environment**, and are expressed by the Greek letter Θ. An environment is called a **frame of discernment** when its elements contain possible answers, and only one of them can be considered the correct answer.

Although belief values can be considered similar to probability theory, they contain important difference. In probability theory, if one has a 0.7 probability that a certain

[67] Mutually exclusive means that if one event happens, the other cannot happen. Cold and hot are said to be mutually exclusive because it cannot be both at the same time.

horse will win the race, it also means that you have a 0.3 probability that it will not win the race. In Dempster-Shafer theory, having a 0.7 degree of belief does not mean that you have a 0.3 degree of unbelief in the outcome. Also, in probability theory, if you have no knowledge at all on the subject, it means that there is an outcome of:

$$P = \frac{1}{N} \text{ , N being the total number of possible outcomes}$$

In most real-life problems, this is not true and cannot be assumed. This is one of the major limitations of probability theory. DST[68] uses **mass functions**[69] that represent the total *mass of evidence* that support the belief, and is written as:

$$m(X) = B \text{ , where X is the outcome and B is the degree of belief } \{0 <= B <= 1\}$$

Dempster's Rule of Combination

A very important feature of DST is that of combining evidence together to come up with a better estimate of the belief in the evidence. The new evidence should be *independent* of the other outcomes, and this issue of independence is a very important aspect when combining evidence. The combined evidence is calculated using:

$$m(A) = \frac{\sum_{B \cap C} m_1(B) m_2(C)}{1 - K} \text{ ,} where\ A \neq \emptyset$$

$$\mathrm{K} = \sum_{B \cap C} m_1(B) m_2(C)$$

Equation 3-2: Dempster's Rule of Combination

m(a) is considered to be the combined mass function, and is a *consensus* between the old evidence and the new evidence. (52)

[68] Dempster-Shafer Theory

[69] Sometimes also called *basic probability assignment (bpa)*

Comparing Dempster-Shafer theory with probability theory

While DST can be considered similar to probability theory, there are various important differences between them. Table 3-2 below compares the two theories together:

Dempster-Shafer Theory	**Probability Theory**
m(Θ) does not have to be 1	$\sum_i Pi = 1$
If $X \subseteq Y$, it does not imply that $m(x) \leq m(y)$	$P(X) \leq P(Y)$
No required relationship between m(x) and m(X\`)	$P(X) + P(X`) = 1$

Table 3-2: Comparison of probability theory and DST[70]

3. 6. 3 Certainty Factors

Another method for dealing with uncertainty uses so called **certainty factors** which were developed by Shortliffe and Buchanan for the MYCIN[71] expert system.

This type of uncertainty management also addresses the same problem addressed by Dempster-Shafer theory – Allowing users to input their belief in the outcome of a condition. Initially, MYCIN was developed to treat uncertainty using probability theory; however it was found out that many experts refused to commit to a probabilistic outcome.

Certainty factor is a method for combining both belief and disbelief in a single value. This is very useful because it can rank hypotheses (outcomes) in their order of certainty.

[70] Adapted from (74)

[71] MYCIN was an early expert system developed in the 1970s that was used to diagnose blood infections and meningitis. Although it was very successful, it was never used in practice. This was mainly due to legal and ethical issues as well as computation power, as this was developed in the era preceding the modern personal computer. However, it had a great influence in the expert systems that were later to be developed.

A certainty factor is defined as:

$$CF(H,E) = MB(H,E) - MD(H,E)$$

, **CF** is the certainty factor of outcome H due to evidence E
MB is the measure of increased belief in H due to E
MD is the measure of increased disbelief in H due to E

Equation 3-3: Certainty Factor[72]

A CF of 1 means certainty of outcome, while a CF of -1 represents total disbelief in the outcome. Therefore, a positive CF means that the expert system is in favour of the hypothesis, while a negative CF represents the opposite. A value of 0 implies neutrality in the outcome.

When new evidence emerges in the form of belief/disbelief, certainty factors can be combined to come up with one single value. (53)

Combining certainty factors

The output of several rules can contribute the belief in a certain outcome. Thus, there is the need to be able to combine certainty factors together. Below are three equations used, based on the value of the *old* certainty factor and the *new* certainty factor:

$$new\ CF(CF_1, CF_2) = \begin{cases} CF_1 + \left(\left(CF_2 \,.\, (1 - CF_1)\right)\right), & CF_1, CF_2 \geq 0 \\ \dfrac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)}, & CF_1 < 0 \ or\ CF_2 < 0 \\ CF_1 + \left(CF_2 . (1 + CF_1)\right), & CF_1, CF_2 < 0 \end{cases}$$

Equation 3-4: Combining certainty factors

A positive aspect of certainty factors is that they require simple computations, while still clearly separating belief from disbelief. (54)

[72] Adapted from (76)

Apart from being used to determine the certainty of the outcome, *certainty factors* are also used to conclude outcome with an amount of certainty, based on the certainty of the antecedents of a rule. This is done by the equation:

$$antecedent\ CF = \min\left(|antecedent_1 CF|, |antecedent_2 CF|, |\ldots|\right)$$

Equation 3-5: Merged antecedent certainty factor outcome for antecedents

This certainty factor is then multiplied with the certainty factor for the consequent, so for example if the antecedent CF is 0.8, and the consequent CF is 0.7, then the final certainty factor for the consequent would be 0.8 x 0.7 = 0.56.

3. 6. 4 Fuzzy Logic

The last method of uncertainty that we will discuss is called **fuzzy logic**. This theory deals with uncertainty based on natural language, where many words can have ambiguous meanings. We use fuzzy logic all the time in our conversations. For example, "If temperature is very hot then switch on AC", uses fuzzy logic. Although we have an idea of what is *"very hot",* it is not defined precisely, and can be ambiguous. What is very hot for a person might be normal for another. We call the term *very* a **fuzzy quantifier.**

This type of logic allows a concept to be contained in a set by a certain degree. For example, a glass that is half-full of water can be said to have a 0.5 degree of fullness, and a 0.5 degree of not fullness. This is the opposite of Boolean logic, where only discrete values of 1 and 0 are allowed.

The degree of membership in a set is calculated using a **membership function.** This is normally a function based on an equation, which translates units into a fuzzy value.

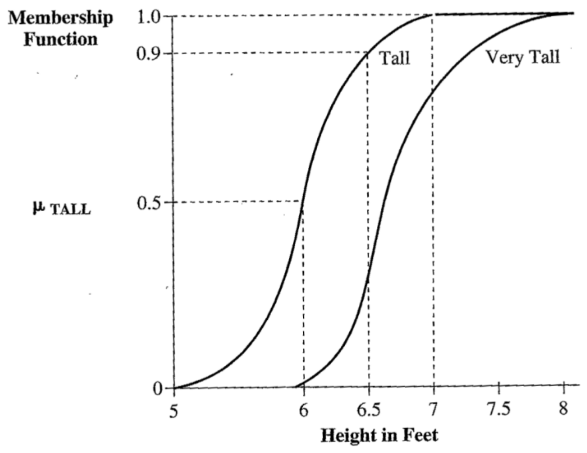


Figure 3-12: Membership function for TALL[73]

Based on Figure 3-12, one can say that a person who is 6.5 feet tall is considered 0.9 tall. This is different from discrete systems based on Boolean logic. Consider a crisp rule saying that a tall person is a person who is taller than 7 feet. A person of 6.9 feet in height is considered medium, and there is a sharp distinction for a very small change in the height value. By allowing membership in the two sets 'medium' and 'tall', one can vastly reduce this problem. (55)

In an expert system, such membership functions will be constructed based on human the expert's opinion of the fuzzy logic value that is being modelled. This can vary from expert to expert.

[73] Adapted from (77)

3.7 Summary

The way knowledge is represented in an expert system is very important. One must take into considerations the various implications of each knowledge representation structure, and consider which ones would be the most suitable for the type of expert system that is to be developed.

Figure 3-13 shows the various different types of information and knowledge available to expert system.

The data structures used to represent knowledge are the main input of the inference engine, and play a very important role in the efficiency of the reasoning process.

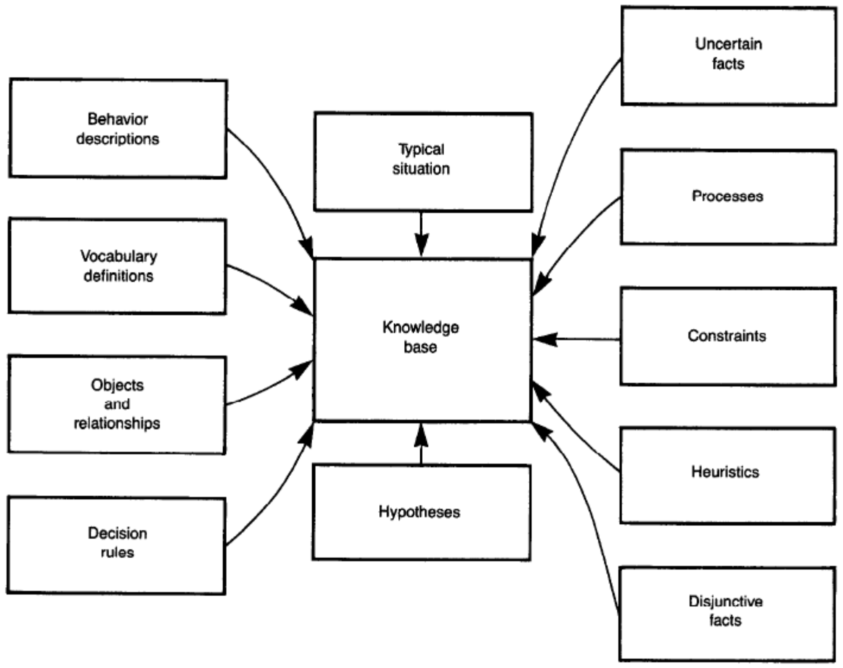


Figure 3-13: Typical knowledge available to an expert system/knowledge base[74]

[74] Adapted from (71)

Our implementation of the expert system will mainly implement three different types of knowledge representation techniques:

- **Rule-based**
- **Object-Attribute-Value triples**
- **Frames**

The main decision for such KR-techniques is due to the general nature of the knowledge bases that are to be implemented. The aim of this project is to create a general-purpose expert system that is not confined to just one specific domain. Knowledge will still be segmented in related knowledge bases, but one has to make certain assumptions because the type of information available can be quite vast and distinct.

We have also discussed various uncertainty management methods and theories which all have their pros and cons. In our implementation, we will be using certainty factors to model uncertainty, as they are computationally simple, users can easily to understand them, and allow easy combination of evidence.

However, the design of the expert system is not the scope of this chapter, and will be explained in further detail in the next chapter - Chapter 4, Design.

4 Design

4.1 Approach

Building an expert system requires various decisions to be made with regards to its various components. An expert system is mainly made up of four components:

- Knowledge Base
- Inference Engine
- User Interface
- Explanation System

Figure 4-1 shows a block diagram of the proposed implementation of the expert system, showing all the main components and how they interact with each other.

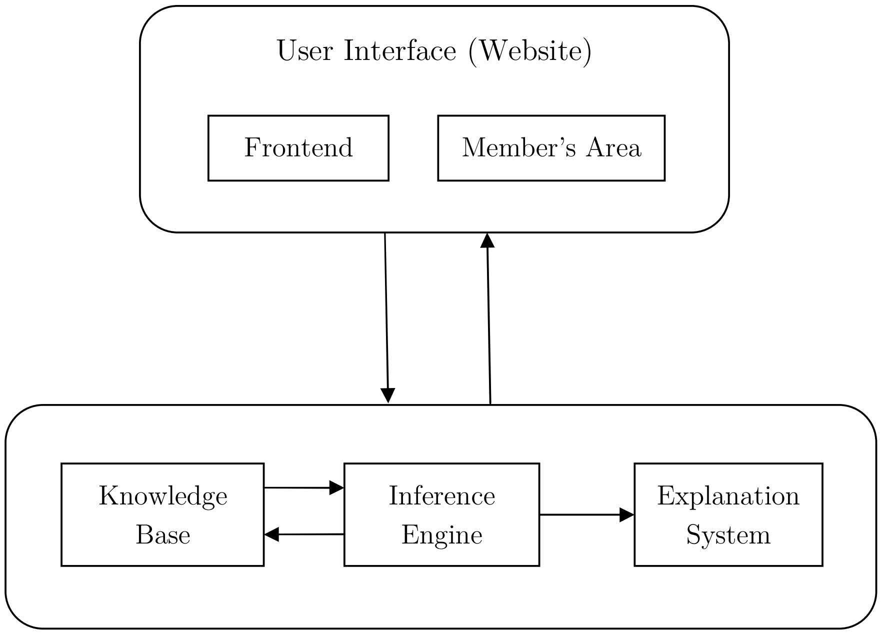


Figure 4-1: Main components of an expert system

4. 1. 1 Knowledge Base

A knowledge base serves as a repository of rules for the inference engine, and expert system in general. One must take into consideration the amount of rules intended to be stored in a knowledge base. Rules can be stored in various possible formats, some of which are:

- Flat Files
- XML Files
- DBMS

We have chosen to use a DBMS to store our rules. This decision was based on the facts that:

- A database can help abstract the details of storing information onto the hard disk
- Indexing can make searching very fast
- SQL queries can enable efficient retrieval of rules

The database that we chose to use was MySQL. The main reasons why we chose MySQL over various others mainly because it is open-source, relatively light-weight, and very fast.

4. 1. 2 Inference Engine

The reasoning process is extremely important in any expert system. Implementing an inference engine involves a considerable amount of work to ensure that it is efficient, useable and can integrate easily with the knowledge base. One can choose to either implement an inference engine from scratch, or choose an existing system.

Choosing an existing system can save you a considerable amount of time, but there are various limitations. It must be well documented and explained, ideally by having a manual on its usage clearly explaining its modules, their inputs/outputs, and any required information to integrate with the inference engine. Code comments would also be extremely useful, as would community support to be able to help you should you find difficulties in integrating with the system. Also, its feature set must complement your needs for the expert system, and one must make sure that rules can be easily consumed by the inference engine, and if needed, feasible uncertainty management methods are implemented that can easily represent the type of problems you intend to solve with the expert system. The programming language (C#, Java, LISP etc.) used to develop the inference engine is also an issue. Some existing open-source inference engines are:

- CLIPS: A software tool for building expert systems developed in C, using a forward-chaining inference engine
- Jess: Started as a Java implementation of CLIPS, later on evolved to include various other features. Includes mainly a forward-chaining inference engine, but also has a backward-chaining module.
- NxBRE: A forward-chaining inference engine developed in C#

- Drools: Another Java implementation of a forward-chaining inference engine

On the other hand, developing your own inference engine would be a much more time consuming approach. However, you would have direct control on its feature set. The inference engine can be tailor made for your knowledge base, and uncertainty management methods required. It can be developed in your language of choice, and you can make it as efficient as possible for your needs.

In our case, we have chosen to implement our own inference engine. The main reason being that most of the existing inference engines available are programmed in Java. Most of the available implementations are mainly focused on forward-chaining inference, and are not very straightforward to use. Also most of them do not cater for uncertainty management, or there are separate offshoots of the project to cater for uncertainty. (Like FuzzyCLIPS, which includes a fuzzy-logic uncertainty management module extension to the CLIPS expert system building tool)

The objective of Paramenides requires an inference engine that is able to do both backward and forward chaining, including mixed inference that allows it to use the best of both worlds when trying to solve a problem. It must also cater for uncertainty, and after analyzing the various different uncertainty methods already explained in Chapter 3, Uncertainty Management, we have opted for *Certainty Factors*. The reason we went for such a decision is because the aim of this project is to cater for as many non-computer savvy users as possible. Certainty Factors are relatively simple to understand, and were thus considered the best solution.

4. 1. 3 User Interface

The user interface of an expert system is extremely important because it is the only way that users can interact with the expert system. Paramenides is aimed at the general public, and thus the UI needs to be extremely user friendly, minimalistic in design, and easy to use. It must also be able to reach a large number of users.

For this reason, we have opted to expose Paramenides as a website, based on ASP.Net v3.5 and AJAX. A website is accessible on any computer that has access to the Internet, and thus makes it very accessible and can potentially include a very large user-base. Expert Systems are always considered to include a tedious installation / configuration process. There is no installation to be done by its users to user Paramenides, and all that one needs is an internet browser like Mozilla Firefox, Internet Explorer, Apple Safari, Opera, etc.

Through the website, the users can perform two main functions:

- Create / Manage Knowledge Bases
- Search for knowledge bases and problem solving, which can be split in:
    - Backward Chaining: Users select a goal (problem), and the system will try and find its possible causes
    - Forward Chaining: Users select a knowledge base, and they input their current symptoms and the system will output the most likely problem. The user can choose to select hypothesis from the list of possible outcomes and the system will ask more specific questions.

The UI will try to make the process as fast as possible. For this reason, we have opted to use AJAX[75]. This makes the user experience much more smooth, fast and responsive. Traditional web pages can be considered relatively slow, as every action requires a round-trip to the server. This is not the case with AJAX, and the performance of such a website can be closely related to that of a desktop application, with all the benefits of an online web application.

AJAX is programmed using JavaScript, which is the language used to program client-side code in modern web-browsers. JavaScript is a weakly-typed[76], dynamic language and these features have their own advantages and disadvantages. It has no built-in implementation of inheritance and interfaces. However, there are various existing frameworks to make mimic object-oriented like functionality, due to the fact that JavaScript is a dynamic language and allows you to add methods and properties on the fly, at runtime. The fact that it is a weakly-typed language can make it very problematic to maintain when the codebase starts getting quite large. Considering that the website's main user interface will be accessed through JavaScript, we have opted to use a different way to program in such a language, so as to make it more maintainable.

We will be using **ScriptSharp**[77] to greatly aid in JavaScript development. ScriptSharp is a tool developed for C# that will translate C# code into human-readable JavaScript code. It will only let you use a small subset of the C# framework, and several HTML

[75] AJAX (Asynchronous JavaScript and XML), is technique used in web development for retrieving data asynchronously from the server without the need to refresh the current page. This technique was made popular through Google's Gmail, which was launched in April 2004. The UI closely mimics that of a desktop application, and has opened up the web to various possibilities previously not thought as feasible.

[76] A weakly typed language means a language where variable data types are not defined and a variable can be used interchangeably.

[77] Available online at http://projects.nikhilk.net/ScriptSharp/

DOM are implemented into a C#-equivalent. It uses the C# compiler to validate the code, and once it has verified that the code passes the C# checks, it will translate it into valid JavaScript, including inheritance.

While ScriptSharp can aid a lot in development, another issue when programming JavaScript is browser incompatibility. Most of the browsers in use today are not fully-standards compliant. Sometimes, doing a simple thing like getting the width of an element in a document can be quite time consuming because what works in one browser does not work in others. We have chosen to use the **Dojo**[78] framework to help in this aspect. The Dojo framework is a JavaScript framework that contains various utility functions and classes to help standardise development and cross-browser compatibility.

Apart from these functions, it also extends the default components for user interface found in standard browsers like text boxes, check boxes, etc. to include custom components built entirely through JavaScript and HTML. These include slider controls, splitter controls, scrollable panels, accordion controls, and tabbed panels, amongst others.

In order to be able to provide a clear explanation of how the expert system came up with that certain conclusion, we will be implementing a component in **Adobe Flash**. Adobe Flash is a web-technology that allows rich user interfaces that are directly deployable in a web-browser. This will allow us to create components that are very user-interactive and easy to use, and allow features like zooming in/out (so as to be able to view the entire tree), animate nodes, and mouse effects like tooltips on rollover, amongst various others.

[78] Dojo toolkit is available online at http://www.dojotoolkit.org/

4. 1. 4 Explanation System

The main role of the explanation system is to explain to the user how the expert system came up with such a conclusion. *Paramenides* keeps track of all the rules used during the inference process. Also during a backward-chaining search, it will keep track of the goals generated in chronological order. Any questions asked by the system are also stored.

These are then used to create a tree-like chart which shows a hierarchical-graph based on the rules used. Each condition is also linked with its *certainty factor,* and the user can visually see how the system came up with such a conclusion. At each node of the tree, the user can see any possible solutions to overcome the condition. The user can also see the list of asked questions and the system can point out due to which conditions it asked the questions.

This explanation-chart will be created as an Adobe Flash component, and the main reason that this was chosen was because it can be easily embedded in a webpage and can make much easier for the system to use. The chart will enable the user to visually see something similar to a *mind-map* of how the system came up with such a conclusion. To aid the user in getting a better idea of the explanation, it will support zooming in/out and panning, as well as visual cues like highlighting the path taken from a certain condition to the goal, tooltips, and others.

4.2 Similar Projects

The concept of having an expert system that is very easy to use and available to the general public through a website is a new concept. However, the fundamental aspect of Paramenides is not new. Expert systems have been around from the mid-20th century, and we have lent various approaches and designs from existing implementations of expert systems. Two notable expert systems described hereunder are MYCIN and PROSPECTOR.

4. 2. 1 MYCIN

MYCIN was an expert system developed in the start of the 1970s at Stanford University. It was written by Edward Shortliffe for his PhD dissertation under the direction of various others including Bruce G. Buchanan. It had most of its roots based in the Dendral expert system, which was developed at the same university. An important feature first introduced in this expert system was *certainty factors.* These allowed the system to closely mimic a doctor's reasoning by catering for elements of uncertainty. Its main role was to identify bacteria causing severe infections as well as for the diagnosis of meningitis. (56)

It used around 500 – 600 production rules, and achieved a great level of success and accuracy compared to human experts and general practitioners. However, it was never used in practice even though its performance was a success. Some of the main reasons were due to ethical and legal issues, since no one could be held responsible if the system gave a wrong diagnosis. It was still a very great influence on future expert systems to come. (57)

4. 2. 2 PROSPECTOR

PROSPECTOR is an expert system developed in the 1980s to aid geologists while looking for minerals. Some of its key features are the various uncertainty management methods that it incorporates, including Bayesian Theory and Fuzzy logic amongst others. It achieved a great deal of fame when it managed to find a molybdenum deposit worth $100,000,000. (58)

4.3 Class and Function Design

The following will briefly discuss the classes / namespaces to be implemented in the expert system. It should give a brief outline of the inner workings of the system, which will then be expanded in more detail in the next chapter, Chapter 5: Implementation.

There will be three aspects to the system − Server-Side code implemented as a C# class library, classes for the ASP.Net user interface code, and Client-Side code implemented in JavaScript, through ScriptSharp.

4. 3. 1 Server-Side Code

***DB* namespace**

This namespace will contain classes that perform database storage and retrieval. It will contain classes that are mapped to the database schema. Since all database access code will be done through the classes in this namespace, this will make the code more maintainable since it will serve as a layer before accessing the database. Search facilities will also be implemented to search for conditions / knowledge bases by keywords.

***WorkingMemory* namespace**

This will serve as the working memory of the inference engine. It will contain a collection of facts currently known by the reasoning process, and several properties on each fact like whether it was inferred by rules or obtained directly from the user. Each fact will also be linked with a list of certainty factors, which can be combined to result in one single certainty factor;

***KnowledgeBase* namespace**

As its name implies, this will contain the implementation of the knowledge base. This will allow retrieval of knowledge bases by keywords, and contains the list of rules that are linked with the knowledge base. A KB is also made up of lists of objects, attributes and values, which are then used to create rules.

ExplanationSystem namespace

This class will keep track of information that is to be used for explaining the system's reasoning to the user. It will keep a list of all the subgoals as generated, the rules used, and the questions asked. It will also keep values as entered by the user for every question asked. Once the backward/forward chaining search is ready, it will create an inference tree which is based on the stored information, which will contain a hierarchical structure displaying the order of rules used and why they were used.

BackwardChaining namespace

This will perform a backward-chaining search on a knowledge base, starting with an initial goal. It will trigger an event whenever a question needs to be asked to the user, and will also automatically generate goals based on the user's response. Any new facts inferenced are also fed to the forward chaining module to see if any new rules have been satisfied based on the current information. It will also keep track of rules used and questions asked through the *ExplanationSystem* class.

ForwardChaining namespace

Performs a forward chaining search on the knowledge base. It will take a given fact and perform a search on the rule base to see if there are any specific rules that have now been satisfied due to the new fact, and fire them.

ReasoningChain namespace

Creates a reasoning chain that keeps track of goals generated in chronological order. It also keeps track of the rules used when processing each goal/sub-goal. The values at that point in time are kept for each processed rule, and any values entered by the user to questions posed by the system are also stored. This is later used for the explanation system.

InferenceTree namespace

The InferenceTree is used for explanation purposes. It creates a hierarchical structure based on the rules. The top level will contain the initial goal, and it will progress downwards based on the sub-goals generated. For each node in the tree, various information will be stored like the condition values, their certainty factors, and any questions asked. This should enable the user to visually follow how the expert system came up with such conclusions.

ExpertSystem namespace

This class will contain the implementation of the expert system. It will contain instances of the *BackwardChaining*, *ForwardChaining*, *WorkingMemory* and *ExplanationSystem* classes, which will be integrated together in order to achieve the functionality of an expert system. An instance of this class is also linked to a knowledge base. The backward chaining search, forward chaining etc will work upon the information contained in that knowledge base.

***Member* namespace**

This will be used to keep track of the logged in user for this session, store and retrieve member information, and perform authentication. Logged in members can have access to the member's area, where they can manage knowledge bases and enter new rules in the KB.

4. 3. 2 ASP.Net User-Interface Code

The ASP.Net user-interface will be split into two main parts:

- Frontend
- Member's Area

The frontend will be available to all users. From the frontend, the users can choose from two distinct types of problem-solving methods:

- Solve a problem (Backward Chaining)
- Find what is the problem (Forward Chaining)

Backward Chaining

When one will choose the first option, he can then search for conditions that match the problem that he currently has. Conditions could be like '*Car is not starting', 'Computer is not working'*, etc. The user can then select a condition from the given list, and the system will then start asking questions to try and find the cause to the problem.

For each question, the user can choose from either a YES or a NO, or a value based on how certain he is of the outcome (from *almost uncertain* to *almost certain*). The user can also choose to reply as *I don't know,* in which case the system will mark it as unknown and generate a sub-goal from the rule. Not all questions can be answered with certainty, and some attributes can only be answered by a YES or NO, due to the intrinsic value of the property. Once the system has exhausted all possible rules, or the conclusion has been reached with certainty, the backward chaining search will stop and the user is shown the outcome together with an explanation (through an inference tree).

Forward Chaining

The user will first enter keywords to search for a knowledge base. A knowledge base contains information related to one specific domain, and needs to be specified because the entered *symptoms* must be related to one specific domain and not scattered from various knowledge bases. After the user has selected a knowledge base, he can then search for symptoms and enter their values. Symptoms can include: *I am feeling dizzy; car battery is flat,* etc. Similar to backward chaining, each fact selected can be answered by either a YES, NO, and a value to represent his certainty. The *I don't Know* option is not available because this time, the user has opted to enter the value and not the system has asked the question.

Based on the given symptoms, the expert system will show the possible outcomes. The user can also choose to select an outcome, and the system will perform a *backward chaining* search with the selected outcome as the goal. This will enable the user to check for a certain hypothesis, and try to increase its likelihood by having the expert system ask related questions.

Explanation sub-system

Once the expert system has come up with a conclusion, it is very important that the user is explained how the expert system came up with such an answer.

This is to be done via a tree-like structure that will display the rules used in chronological order, any questions asked at which nodes, and the certainty of the facts/conditions. The user can click on a node and more information on the node will be displayed, like the goals generated through that node, any questions asked and their values, as well as zoom in/out to be able to view the tree more comfortably if it is quite large.

4.4 Summary

In this chapter we have described the approach that we will be taking in creating the proposed system. We have outlined what each component of the expert system is expected to do, and provided a detailed comparison on the different methodologies available for each component and reasons for our decision in designing it this way. We have also given a brief summary of other expert systems that are based on the same fundamental aspect as Paramenides. Finally, we have given a brief outline of how we intend to split the project into various modules and namespaces, in order to make it easier to maintain and test.

5 Implementation

In this chapter, we will go through the implementation details of Paramenides. The project was built mainly using C# 3.5, ASP.Net, JavaScript (using ScriptSharp) and AJAX. The explanation system was developed using Adobe Flash.

We shall give primary importance to the logic and functionality of the inference engine and expert system in general. The backward chaining / forward chaining inference modules will be explained in detail, and we will also go through the explanation subsystem and the user interface used for inputting rules in a knowledge base. We will also describe the database schema, and the database access code and classes used for user interface (both in ASP.Net and JavaScript).

5.1 Database Schema

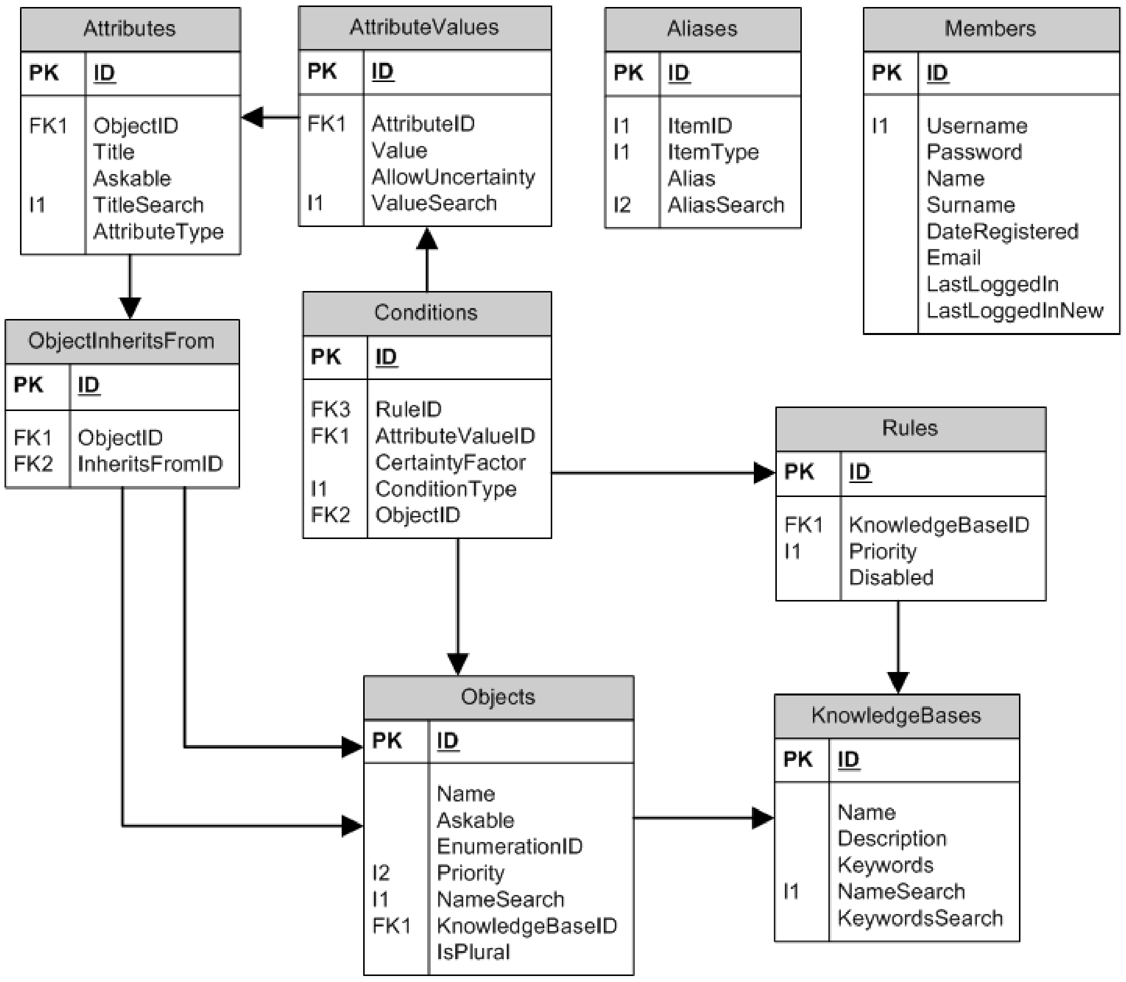


Figure 5-1: Database schema as used by Paramenides

The diagram above shows the tables in the relational database that are used by the expert system. The arrows show the relationships between the tables, with the table being pointed to being the *parent* table while the other being the *child* table that contains the *foreign key*.

These tables are mapped to C# classes for database access, in order to create a data-access layer and make code more maintainable. These classes are explained in the next section.

5.2 *ExpertSystemFYP* Class Library

In this section we will describe the class library that contains the implementations of the expert system and all its various sub-systems, including the backward-chaining module, the forward chaining module and the explanation module.

5. 2. 1 *DB* namespace

This namespace contains classes that map one-to-one with the tables in the database. The base code for these classes is generated using a tool that I have created myself. The tool takes a database schema, and it will generate code for each table, and for a collection of rows from the table. It will also create links between the classes for foreign keys. We will describe the typical generated classes by the tool. There is no need to explain every class in this namespace because most of them contain the exact code but replicated for the different column names. For each table, 7 classes are created, named:

- Base{TableName}:
- Base{TableName}Factory
- {TableName}
- {TableName}List
- {TableName}Factory
- I{TableName}
- I{TableName}Factory

We will use the **Conditions** table to explain these generated classes.

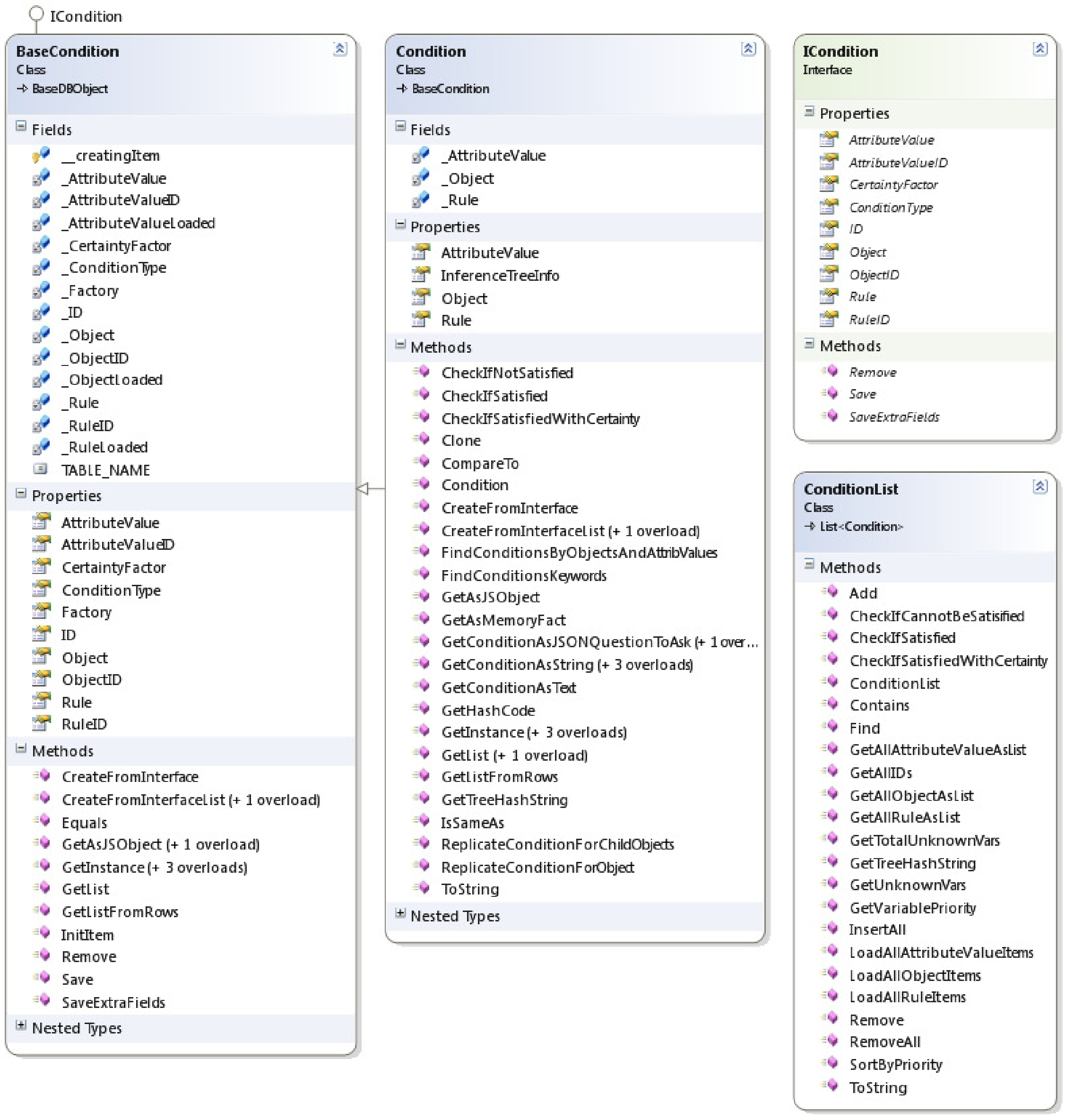


Figure 5-2: Class diagram for generated classes (continued)

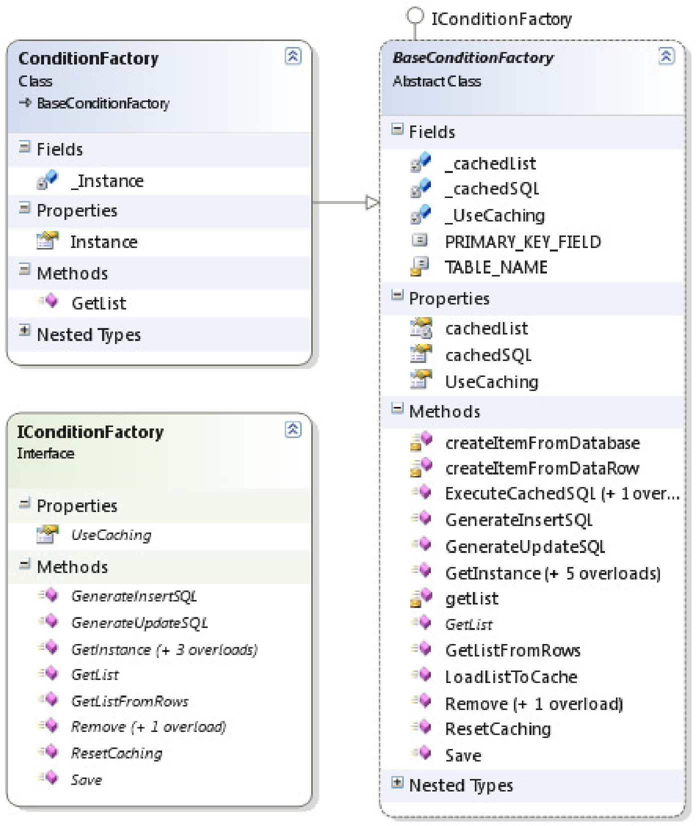


Figure 5-3: Class diagram for generated classes

The **ICondition** interface defines all the properties that map to the column names in the table. It also contains Save() and Remove() methods to perform inserts, update and deletes on the database.

BaseCondition implements ICondition and maps to a row in the Conditions table. Apart from defining properties based on the interface that it implements, it also serves as a singleton factory for retrieving rows from the database, based on ID. The class does not include a public constructor, and the only way to get a record from the database is to use BaseCondition.GetInstance(id). This is done because it can enable *caching* to be used, and if a row has already been retrieved from database, it can be loaded from memory instead of from the database to save on access time. This is controlled from the factory class. It also includes a static method GetList() which retrieves a list of BaseConditions, based on a **criteria** variable. The *criteria* class contains properties used for searching on the database. By filling these properties, they are then converted into an SQL query and it is used to retrieve a list of rows from the database.

Since the code in ICondition and BaseCondition is automatically generated, it is not suggested to amend any portions of it because it would be overwritten when the code is re-generated. While this can be easily overcome by using *partial classes*[79], it does not allow you to attach to existing methods like Save() and Remove(). **Condition** extends from BaseCondition, and can allow you override all the methods in the **BaseCondition** since all of them are marked as *virtual.* Partial classes are used so as not to be overwritten if code is re-generated.

[79] Partial classes are a feature of C# 2.0 onwards that allows classes to have multiple definitions in separate files. This is aimed specifically for code generation. C# 3.0 also introduced *partial methods* and allows you to define 'method placeholders' for partial classes that can be optionally defined in another partial class.

`ConditionList` extends from `List<Condition>`[80], and defines methods to act on the entire collection. Some of these include loading in bulk any related classes linked by a foreign key, saving in bulk, amongst others.

`IConditionFactory` contains properties and methods defined in the Factory. The Factory is used as the entry point to the database, and contains methods to convert a list of database rows into BaseConditions, retrieving conditions by their primary key, amongst others.

`BaseConditionFactory` implements `IConditionFactory`, It also can be used to enable *caching*. Caching allows any rows loaded by primary key to be loaded from memory instead of from the database. This can result in much faster data loading, but if a row is updated without going through the factory, the row will not reflect the latest database changes.

`ConditionFactory` extends on `BaseConditionFactory`, for the same purpose that `Condition` extends from `BaseCondition` – To allow overridden virtual methods defined in `BaseConditionFactory` by the auto-generated code.

These classes explained above are defined also for the other tables. These are:

- Alias
- Attribute
- AttributeValue
- Condition
- KnowledgeBase
- Member
- ObjectInheritsFrom
- Object
- Rule

Also, please note that not all methods/properties shown in Figure 5-2 and Figure 5-3 are auto-generated.

[80] List<TYPE> is called a generic class, and represents a list of TYPE.

5. 2. 2 *Rules* namespace

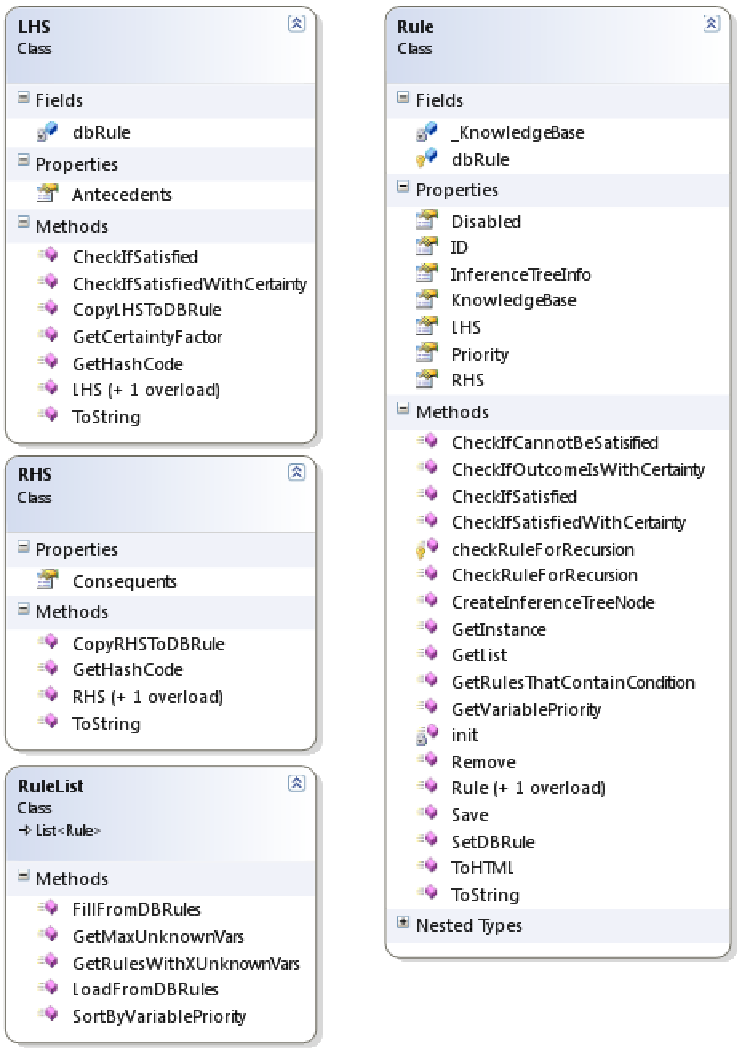


Figure 5-4: Rules namespace

There are two classes named **`Rule`** in the project. One is found in the `DB` namespace, while the other in the `Rules` namespace. The difference is that the one found in the `DB` namespace maps one-to-one with the table in the database, while the one found in the `Rules` namespace contains 'higher-level' properties and methods related to the functionality of a rule within a knowledge base. The `Rule` class contains the **`LHS`** and **`RHS`** of the rule. It also stores information related to the node it is found on in the *inference tree.* Apart from these mentioned properties, it contains methods to check whether it is satisfied based on a given `WorkingMemory`, whether it is satisfied with certainty, whether it can NOT be satisfied, and others. These methods are used during backward/forward chain searches. A rule also contains a *priority* value. The priority value is used to sort the rules by, so that when the expert system needs to find a certain set of rules pertaining to some conditions and more than one match, they are sorted by this value and the one with the lowest priority value is used first.

A **satisfied** rule implies that all of its antecedents are contained in the current working memory. Apart from facts, the working memory also stores the *certainty factor* associated with each condition. A condition's value must either be TRUE or FALSE. A TRUE condition is represented by a *certainty factor* that is greater than 0, while a FALSE condition is represented by a *certainty factor* that is smaller than 0.

5. 2. 3 *Conditions* namespace

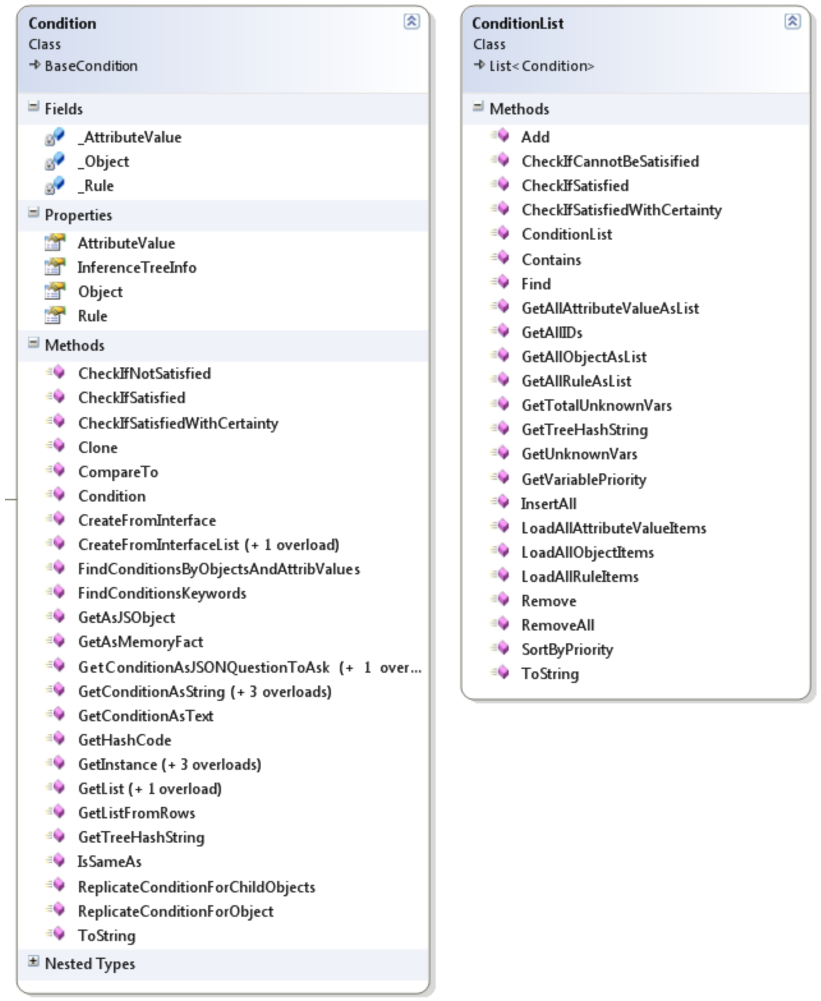


Figure 5-5: Condition namespace

Apart from the auto-generated code already explained in the `DB` namespace, the **`Condition`** class also contains various other methods. Some of these methods are:

- Check if the condition is satisfied with certainty, from a given `WorkingMemory`
- Check if the condition cannot be satisfied
- Find conditions by keywords
- Replicate condition to create a condition for each object inheriting from its object.

5. 2. 4 *KnowledgeBase* class

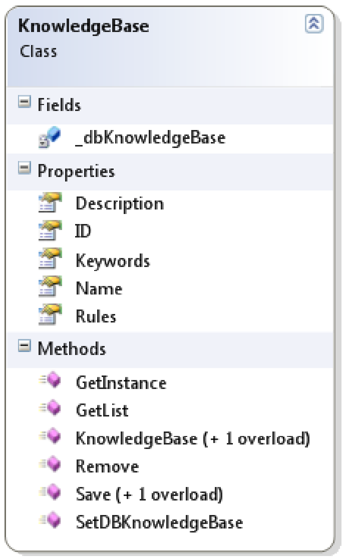

KnowledgeBase
Class
Fields
_dbKnowledgeBase
Properties
Description
ID
Keywords
Name
Rules
Methods
GetInstance
GetList
KnowledgeBase (+ 1 overload)
Remove
Save (+ 1 overload)
SetDBKnowledgeBase

Figure 5-6: KnowledgeBase class

The **`KnowledgeBase`** class is a wrapper on the `KnowledgeBase` class that is mapped to the database. It contains a list of rules that are based on the `Rule` class that maps to the higher-level Rule structure as used by the inference engine.

5. 2. 5 *ExpertSystem* class

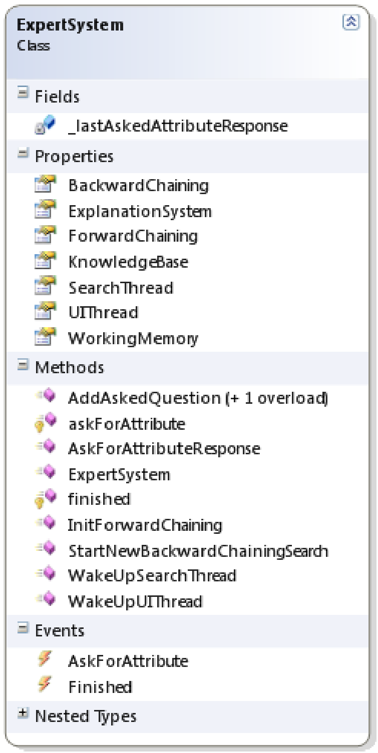
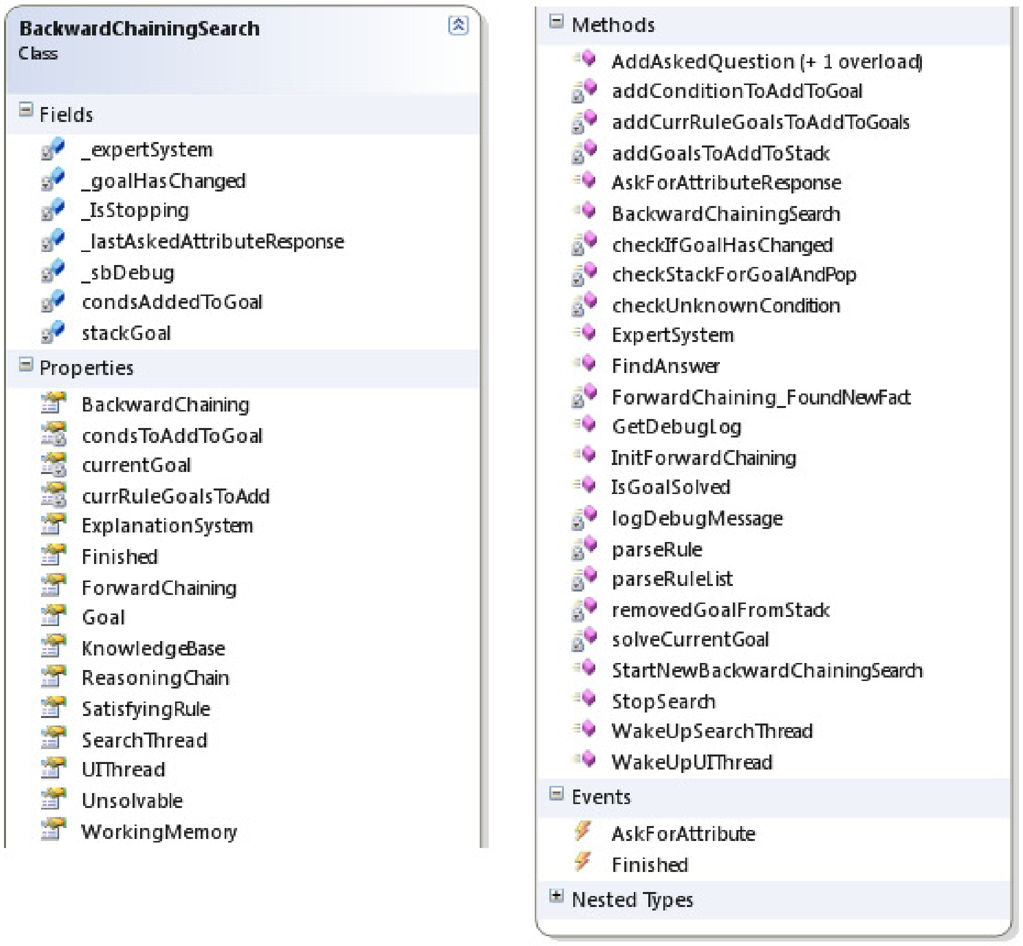


Figure 5-7: *ExpertSystem* class

The **`ExpertSystem`** class contains the main functionality of the project. It contains instances of the `BackwardChaining`, `ForwardChaining`, `ExplanationSystem` and `KnowledgeBase` classes. It also exposes two events `AskForAttribute` and *Finished*. `AskForAttribute` is triggered whenever a new attribute is to be asked by the user interface. The user interface will then attach to this event and ask for the value from the user. This is mainly used during a backward-chaining search. Once the UI is ready, it will call `AskForAttributeResponse()`. The `ExpertSystem` class also keeps track of the thread for the user interface, in `UIThread`. Since the UI thread will be waiting for an *event* to occur (ask for a variable), this reference is used to notify the thread when such an event occurs.

5. 2. 6 *BackwardChainingSearch* class



BackwardChainingSearch
Class

Fields
_expertSystem
_goalHasChanged
_IsStopping
_lastAskedAttributeResponse
_sbDebug
condsAddedToGoal
stackGoal

Properties
BackwardChaining
condsToAddToGoal
currentGoal
currRuleGoalsToAdd
ExplanationSystem
Finished
ForwardChaining
Goal
KnowledgeBase
ReasoningChain
SatisfyingRule
SearchThread
UIThread
Unsolvable
WorkingMemory

Methods
AddAskedQuestion (+ 1 overload)
addConditionToAddToGoal
addCurrRuleGoalsToAddToGoals
addGoalsToAddToStack
AskForAttributeResponse
BackwardChainingSearch
checkIfGoalHasChanged
checkStackForGoalAndPop
checkUnknownCondition
ExpertSystem
FindAnswer
ForwardChaining_FoundNewFact
GetDebugLog
InitForwardChaining
IsGoalSolved
logDebugMessage
parseRule
parseRuleList
removedGoalFromStack
solveCurrentGoal
StartNewBackwardChainingSearch
StopSearch
WakeUpSearchThread
WakeUpUIThread

Events
AskForAttribute
Finished

Nested Types

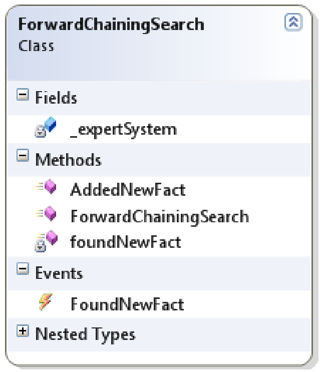
Figure 5-8: BackwardChainingSearch class

The **`BackwardChainingSearch`** class does the backward-chaining searches of the expert system. The class takes the *goal* as its input parameters. It will check any rules in the knowledge base that have the *goal* as one of their consequents, and see if they can be *satisfied* so that the rule can *fire*. During the process, it will ask questions to the users (by triggering event `AskForAttribute` that is captured by the UI), and generating sub-goals for questions that the user does not know the answer for. Algorithm 5-1 describes the algorithm in more detail. During the backward search, information is also stored in the `ExplanationSystem` class that is then used for providing explanations to the user. The information stored will be explained in more detail when explaining the related class.

```
Backward Chaining Algorithm
Set mainGoal as {Input_Goal}
Initialise GoalStack
Add mainGoal to  top of GoalStack
Set LastGoalSolved = false
Initialise WorkingMemory
While GoalStack is not empty
.... Pop GoalStack in currentGoal
.... Set lastGoalSolved = false
.... If currentGoal is NOT already in WorkingMemory with certainty then
.... .....Set RuleList = Get Rules that have the currentGoal as one of their consequents
.... .....Sort RuleList by the rules that have the least unknown variables and priority
.... .....Foreach rule in RuleList
.... ..... ....If Rule can be satisfied then
.... ..... .... ....ForEach unknown condition in Rule
.... ..... .... .... ....If condition is not already asked then
.... ..... .... .... .... .... Ask Question to user
.... ..... .... .... .... ....If response is NOT 'I Don't Know' then
.... ..... .... .... .... .... .... Add response to working memory
.... ..... .... .... .... ....Else
.... ..... .... .... .... .... .... Add condition as new sub-goal
.... ..... .... .... .... .... .... Mark condition as unknown and already asked
.... ..... .... ....If rule is satisfied then
.... ..... .... .... ....Add rule's consequents to working memory
.... ..... ....If rule is satisfied
.... ..... .... ....Set lastGoalSolved = true;
.... ..... .... ....Break;
If lastGoalSolved then
.... 'Goal has been found'
Else
.... 'There is no solution'
```

Algorithm 5-1: Backward chaining search algorithm

5. 2. 7 *ForwardChainingSearch* class



The *ForwardChainingSearch* class performs forward chaining inference on the knowledge base. Whenever a new fact is entered into the working memory, the forward chaining module is triggered and it will to fire any rules whose antecedents have now been satisfied. This is done either when new information is *discovered* either through facts inserted by the user, or through the consequents of rules fired during a backward-chaining search. Algorithm 5-2 describes the process involved, which is fairly simple.

Forward-Chaining Search algorithm
Inputs: Fact

```
Get all rules that contain fact as one of their antecedents
ForEach rule in Rules
.... If rule is satisfied then
.... .....ForEach consequent in rule
.... ..... ....Add consequent to working memory
.... ..... ....Do another forward chaining search on consequent [Recursion]
```

Algorithm 5-2: Forward chaining search algorithm

5. 2. 8 *WorkingMemory* namespace

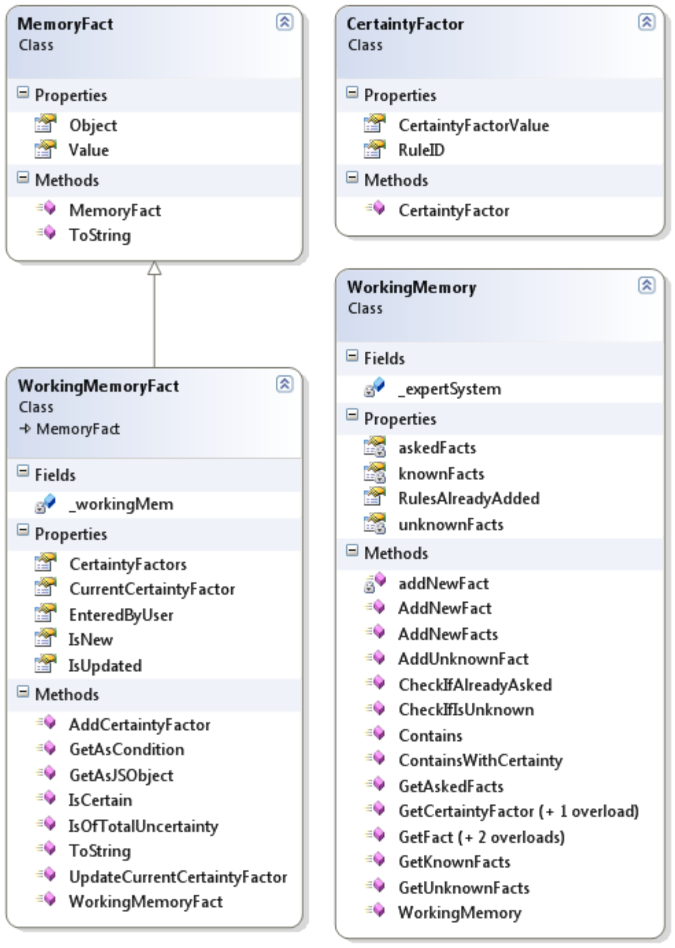


Figure 5-9: WorkingMemory namespace

The **WorkingMemory** namespace contains classes that are related to the working memory of the knowledge base. It allows adding and removing facts to the working memory. When a new fact is added to the working memory, the forward chaining module is invoked to try and extract as much information from the new fact as possible. It is made up of a list of WorkingMemoryFact*s.*

WorkingMemoryFact contains the current certainty factor for the fact and the list of certainty factors in order to re-calculate the certainty factor if a fact is added/removed.

5. 2. 9 *InferenceTree* namespace

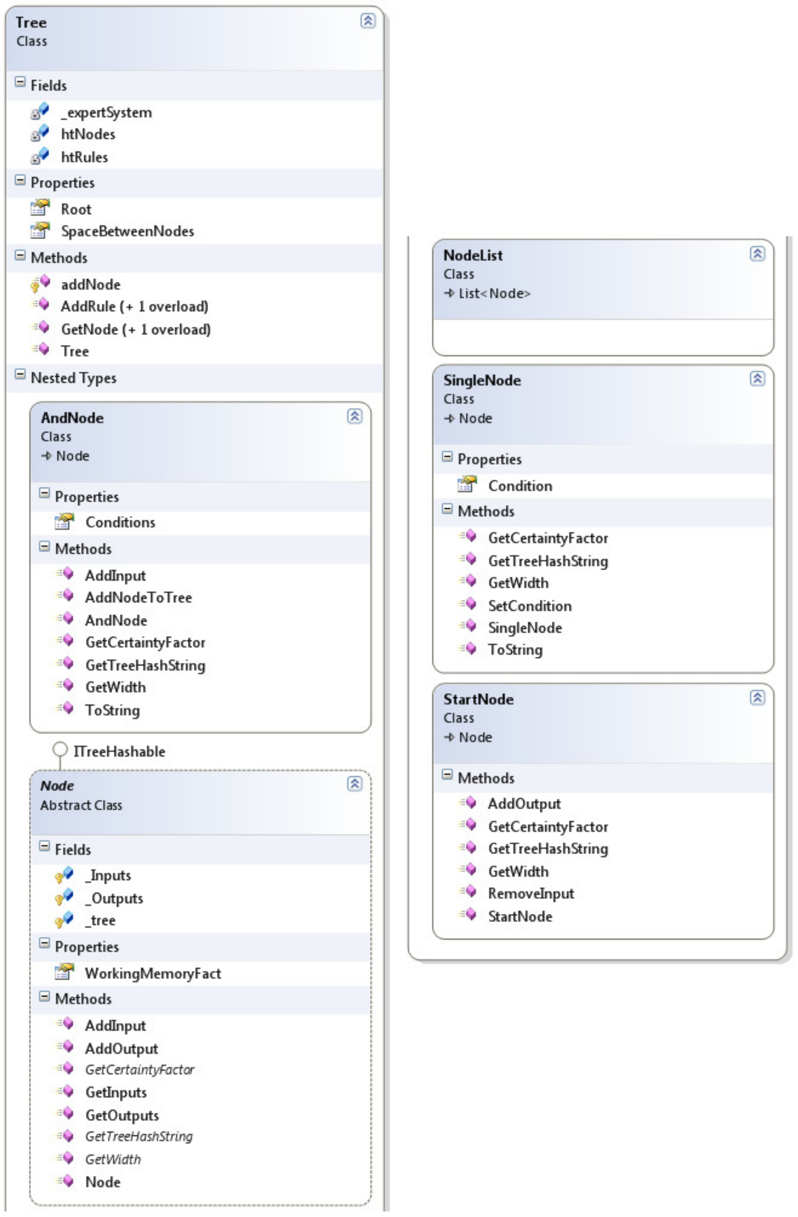


Figure 5-10: InferenceTree namespace

The **`Tree`** class is built from the list of rules used by the backward chaining / forward chaining modules in the expert systems. Each rule used by one of these modules is stored in a list. These rules are then used to create an inference tree, which is made up of a hierarchy of nodes. The nodes contain information about the rules and conditions. The top most nodes represent the initial goals to be solved, while the children show the various sub-goals generated during the backward chaining process. The `AndNode` is used to AND antecedents together, and serves as an intermediary node between the AND-ed antecedents and the consequents.

The tree keeps track of all its nodes and the facts represented by each node in a hash table. This is used when generating the tree so as not to generate duplicate nodes. To generate the tree, all the rules used are iterated and their antecedents and consequents are extracted and used to create the nodes, starting from the root. Algorithm 5-3 describes the process involved in creating and adding nodes to a tree, given an input rule.

Creating/Appending to an inference tree from a rule
Input: Rule

```
Set node = null;
If total antecedents in rule = 1 then
.... Set node = get node from tree based on antecedent
.... If node is null then
.... .....set node = create new node based on antecedent
.... .....add node to tree
else
.... set node = new AND-node
.... ForEach antecedent in rule
.... .....Set currNode = get node from tree based on antecedent
.... .....If currNode is null then
.... ..... ....Set currNode = create new node based on antecedent
.... .....Add currNode to children of node
ForEach consequent in rule
.... Set currNode = get node from tree based on consequent
.... If currNode is null then
.... .....Set currNode = create new node based on consequent
.... Link currNode as parent of node
```

Algorithm 5-3: Creating/Appending to an inference tree from a rule

While the algorithm may not seem as if it is building a correct tree, one needs to give special attention to the loop going over the rule's consequents. Each node in the tree represents a fact, and each fact is unique. The `tree` stores a *hash table* of each node added to the tree, and if it already exists it is returned when requested. Any nodes that have no *parent* nodes are attached to the root nodes. These would become the initial goals. The tree can contain multiple initial-goals. This could especially be the case if the inference tree is built after a forward-chaining search, as opposed to a backward-chaining one.

5. 2. 10 *ExplanationSystem* namespace

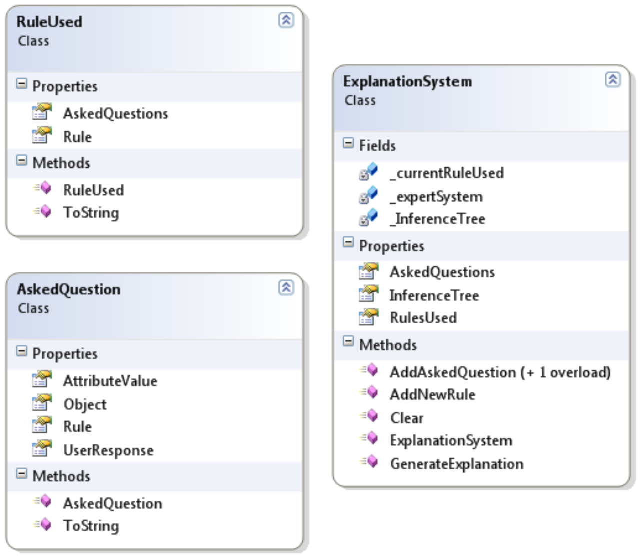


Figure 5-11: ExplanationSystem namespace

The **`ExplanationSystem`** class is the module that performs the explanation of the system to the end user. It is constantly updated during the backward-chaining/forward-chaining process with the rules used and questions asked to the user. The responses of the user are also logged. This information is then used to generate the `InferenceTree`.

5. 2. 11 Utility classes

Apart from the mentioned namespaces and classes, there are various utility classes whose aim is to perform repetitive functionality that is not directly related to the expert systems. These include:

- Database access
- Sorting
- Searching on Lists
- Conversions from data type to another

They are relatively simple in nature, and there is no scope in detailing each of them in this dissertation, and explaining their functionality.

5.3 User Interface

As already explained, the user interface for the expert system will consist of a website that is publicly accessible at www.paramenides.com. The website will consist of two main sections:

- Searching/Solving for problems (Backward/Forward chaining)
- Member's Area for managing knowledge bases

Once the user enters the website, he can either choose to search for knowledge base or solutions to his problem, or log in to the member's area. If the user does not already have an account, he can choose to create one.

To make the website much faster and responsive, AJAX is used. This does not require every action on the website to require a round-trip to the server. This can be seen for example, by typing some keywords in the search box and clicking on GO. Using

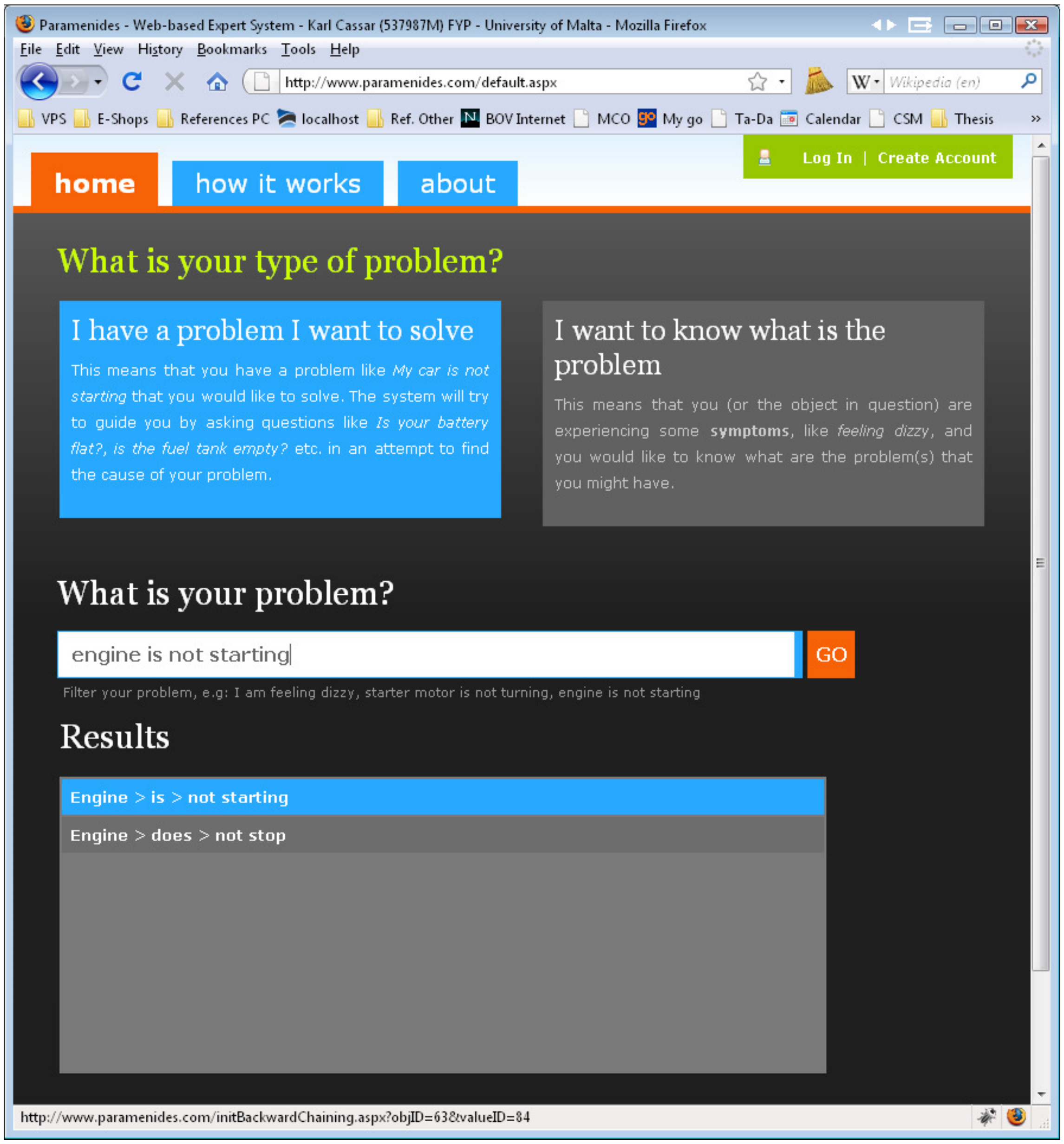
JavaScript, an asynchronous call is made using the JavaScript object `XMLHttpRequest`, which will retrieve data from the server. As soon as the data returns, this is updated in the listing. This type of asynchronous programming is used for:

- Managing knowledge bases (objects, attributes & values)
- Managing rules in a knowledge base
- Searching
- Asking questions to the user / submitting response
- Entering facts (for forward chaining)

Script# is used to translate C#-equivalent code into human-readable JavaScript code, as this makes it much more maintainable. Also, the JavaScript Dojo toolkit is used to extend JavaScript functionality and browser-compatibility.

5. 3. 1 Website Frontend

The frontend of the website serves as a search engine to let the user find the problems that he wishes to solve, or knowledge bases to work upon. The user has the choice to either perform a backward chaining search, or a forward chaining search.


Paramenides - Web-based Expert System - Karl Cassar (537987M) FYP - University of Malta - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.paramenides.com/default.aspx

VPS E-Shops References PC localhost Ref. Other BOV Internet MCO My go Ta-Da Calendar CSM Thesis

Log In | Create Account

home how it works about

What is your type of problem?

I have a problem I want to solve

This means that you have a problem like *My car is not starting* that you would like to solve. The system will try to guide you by asking questions like *Is your battery flat?*, *is the fuel tank empty?* etc. in an attempt to find the cause of your problem.

I want to know what is the problem

This means that you (or the object in question) are experiencing some **symptoms**, like *feeling dizzy*, and you would like to know what are the problem(s) that you might have.

What is your problem?

engine is not starting

GO

Filter your problem, e.g: I am feeling dizzy, starter motor is not turning, engine is not starting

Results

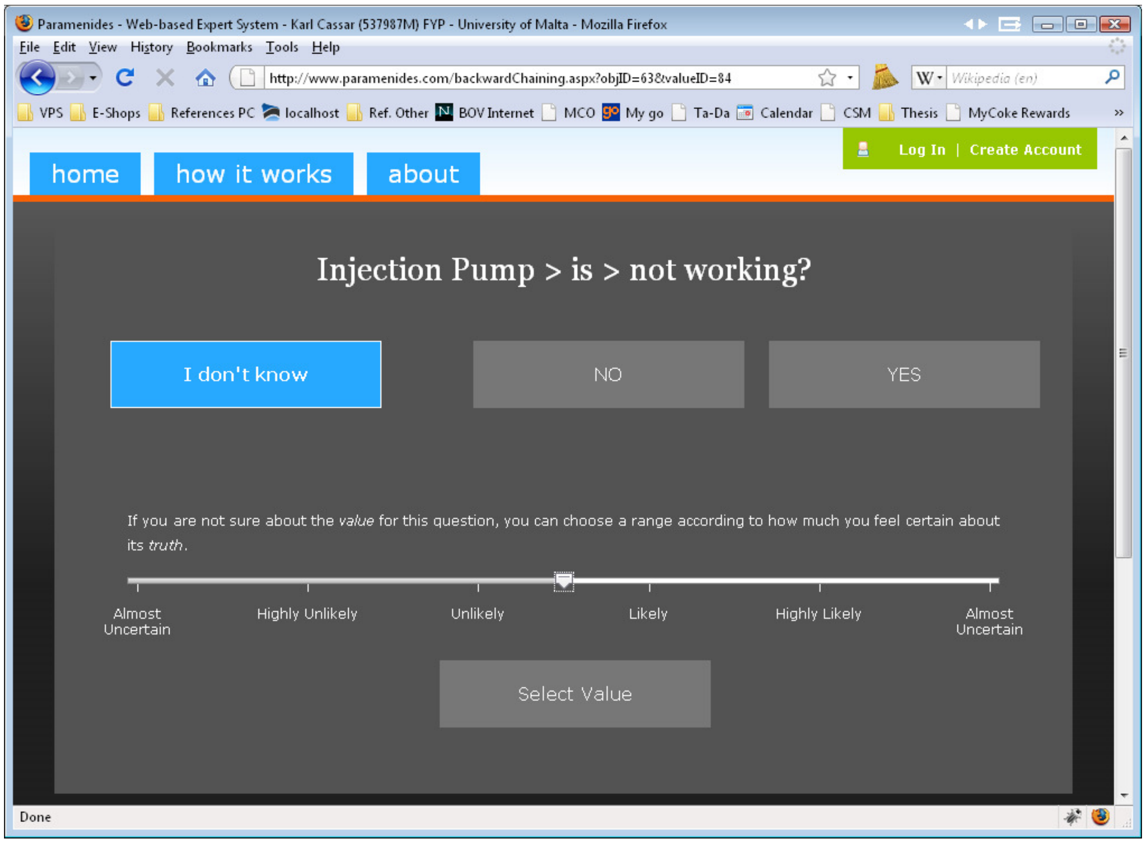
Engine > **is** > **not starting**

Engine > **does** > **not stop**

http://www.paramenides.com/initBackwardChaining.aspx?objID=63&valueID=84

Figure 5-12: The homepage of the website

5. 3. 2 Backward Chaining Search


Paramenides - Web-based Expert System - Karl Cassar (537987M) FYP - University of Malta - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.paramenides.com/backwardChaining.aspx?objID=63&valueID=84
home
how it works
about
Log In | Create Account
I don't know
NO
YES
Almost Uncertain
Highly Unlikely
Unlikely
Likely
Highly Likely
Almost Uncertain
Select Value
Done

Injection Pump > is > not working?

If you are not sure about the *value* for this question, you can choose a range according to how much you feel certain about its *truth*.

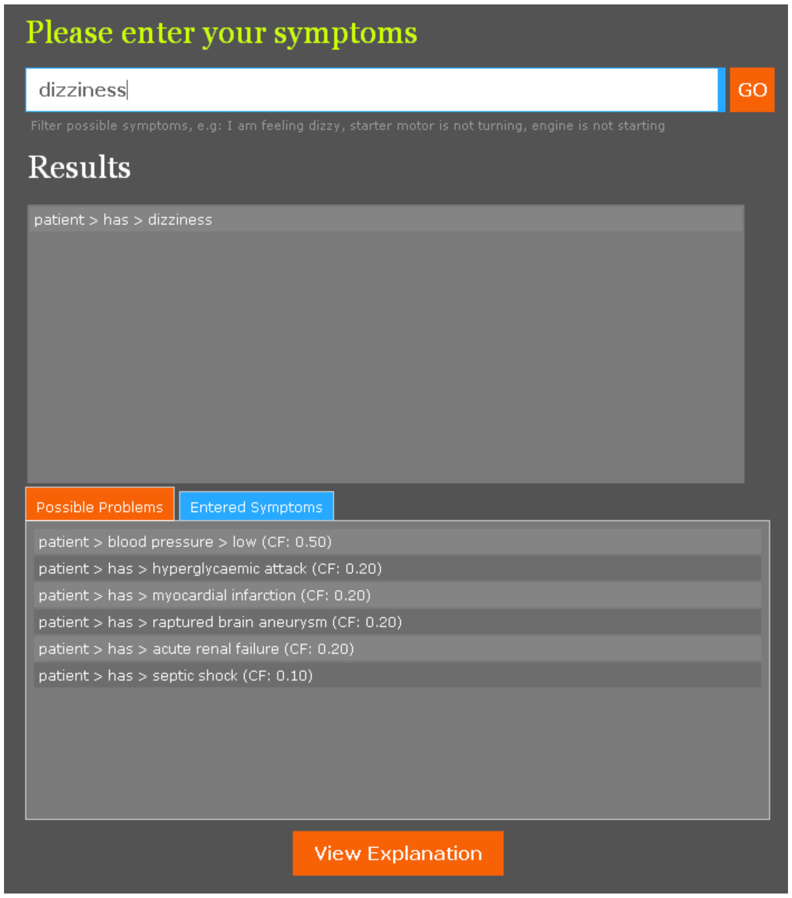
Figure 5-13: A question asked by the system, during backward search

The backward chaining module of the system will ask the user a series of question similar to Figure 5-13, through which the user can answer either as a simile YES or NO if he knows the exact answer. If he is uncertain but has a certain belief in the statement, the slider allows the user to portray his belief. *I don't know* can be used if the user has no idea of the answer. In that case, the system will automatically generate this condition as a sub-goal, when all the possible rules for the existing goal have been exhausted.

5. 3. 3 Forward Chaining Search

The forward-chaining search interface allows the user to search for symptoms, enter their values and the system will list the possible problems based on the given inputs. The user can even select a possible problem and the system will perform a backward-chaining search on the problem, and try to come up with better conclusions on the problem by asking further questions related to the chosen problem.

The user can also see the symptoms he has entered into the system, and can choose to view an explanation which display a tree like structure with the goals being all the possible problems.


Please enter your symptoms

dizziness

Filter possible symptoms, e.g: I am feeling dizzy, starter motor is not turning, engine is not starting

Results

patient > has > dizziness

patient > blood pressure > low (CF: 0.50)
patient > has > hyperglycaemic attack (CF: 0.20)
patient > has > myocardial infarction (CF: 0.20)
patient > has > raptured brain aneurysm (CF: 0.20)
patient > has > acute renal failure (CF: 0.20)
patient > has > septic shock (CF: 0.10)

Figure 5-14: A typical forward-chaining search conversation

5. 3. 4 Explanation System

The explanation system provides a visual representation of the conclusions reached by the expert system. The use of Adobe Flash has enabled us to allow the user to zoom in/out to get a better idea of the graph, tracing the line to easily follow the conclusion reached, tooltips and panning to view the other portions of the graph.

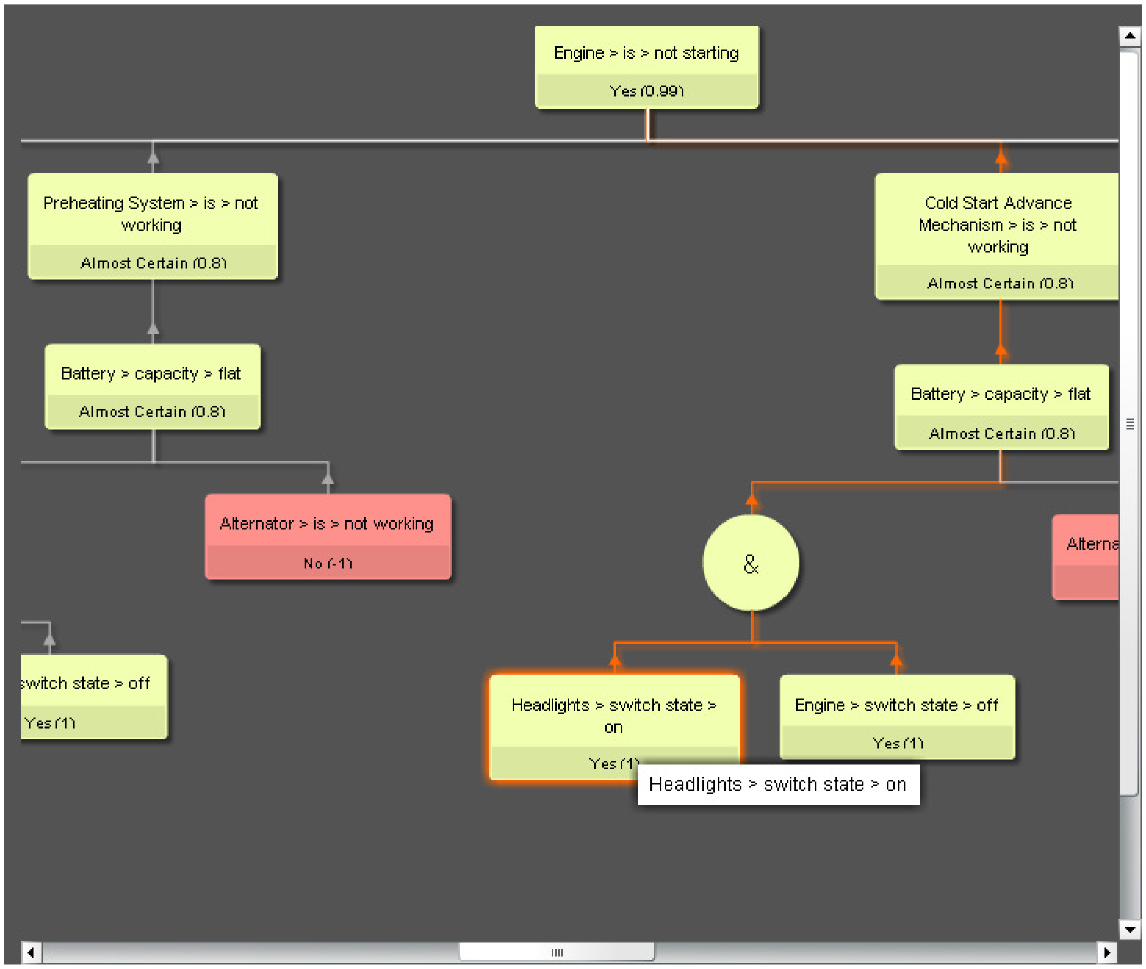

Engine > is > not starting
Yes (0.99)
Preheating System > is > not working
Almost Certain (0.8)
Cold Start Advance Mechanism > is > not working
Almost Certain (0.8)
Battery > capacity > flat
Almost Certain (0.8)
Battery > capacity > flat
Almost Certain (0.8)
Alternator > is > not working
No (-1)
&
Alterna
switch state > off
Yes (1)
Headlights > switch state > on
Yes (1)
Engine > switch state > off
Yes (1)
Headlights > switch state > on

Figure 5-15: A portion of the explanation system

5.4 Summary

This chapter has provided a detailed explanation of the inner workings of each component of the system. We have explained the schema of the database we are using, the main technologies used to implement the expert system and reasons for using the technologies. An overview of each module of the expert system is given, and also provided more detailed explanation of the algorithms for the backward chaining and forward chaining modules, through pseudo code.

6 Evaluation

In this chapter we shall provide an evaluation of the expert system developed to give us an indication of its performance

The system will be tested in its three main aspects:

- Backward Chaining
- Forward Chaining
- Rule Management

In order to be able to test better the functionality of the system, a total of three knowledge bases have been inserted. The test plan is split into two different stages, which are:

- System Testing
- Human Evaluation

We will later explain these two different types of testing, and how the test cases used.

6.1 Corpus (Knowledge Bases)

Paramenides was created with the goal in mind to cater for a large number of knowledge bases. However it is impossible to enter knowledge bases on many distinct domains and for the evaluation of the system we will be using three knowledge bases. The questionnaires provided the users will act upon two different knowledge bases, which will be explained briefly.

6. 1. 1 Diesel Car Engine Starting Diagnostics

This knowledge bases includes various rules about the problems that may arise if a car does not start. This tests conditions like whether the starter motor is working well or not, the kind of weather, the car battery state, amongst various others. It includes around 35 rules, which are listed in Table 6-1: Diesel Car Engine Starting Diagnostics rule base

Antecedents	**Consequents**
engine turns and engine does not start	cold start advance mechanism is faulty (0.3)
engine turns and engine does not start	preheating system is faulty (30%)
engine turns and engine does not start	cranking speed is low
engine turns and engine does not start	engine's compression is poor
engine turns and engine does not start	Fuel tank is empty
fuel tank empty	engine does not start
cranking speed is low and weather is cold	engine does not start (50%)
preheating system is faulty then	Engine does not start (30%)
cold start advance mechanism is faulty then	Engine does not start (30%)
fuel system contains air	engine does not start
engine turns and engine does not start	air in fuel system (30%)
engine turns and engine does not start	fuel is contaminated (20%)
engine turns and engine does not start	stop solenoid defective (30%)
engine turns and engine does not start	injection pump internal fault (30%)
low cranking speed	battery capacity is inadequate (40%)
low cranking speed	oil grade is incorrect (30%)
low cranking speed	Start motor circuitry high resistance (30%)
low cranking speed	Starter motor is faulty (30%)
starter motor is faulty	cranking speed is low
starter motor circuitry high resistance	cranking speed is low
oil grade is incorrect	cranking speed is low (80%)
injection pump internal fault	engine does not start
battery capacity is flat	electrical component is not working
starter motor is not working	engine does not turn (100%)
fuel system contains air then	Engine does not start (70%)
fuel is contaminated then	Engine does not start (50%)
stop-solenoid is defective then	Engine does not start (40%)
engine compression is poor then	Engine does not start (70%)
stop-solenoid is defective and engine is started and engine switched off then	Engine does not stop (80%)
headlights are on and engine is switched off then	Car battery is flat (80%)
alternator is not working then	Car battery is flat (60%)
Starter motor is an electrical component	
Cold Start Advance Mechanism is an electrical component	
Fuel Pump is an electrical component	
Preheating System is an electrical component	
Headlights are an electrical component	

Table 6-1: Diesel Car Engine Starting Diagnostics rule base[81]

[81] *cf* is short for 'certainty factor' and represents the amount of certainty the system has in that conclusion

6. 1. 2 Clinical Manifestation Diagnosis

This knowledge base includes rules to diagnose clinical manifestations like sepsis, asthma, septic shock, etc by comparing symptoms of the patient.

Mental status Change	Sepsis (cf 0.3)
Hyperventilation	Sepsis (cf 0.2), Asthma Attach (cf 0.5)
Hyperthermia	Sepsis (cf 0.5)
Hypothermia	Sepsis (cf 0.5)
Short of breath	Asthma Attack (cf 0.5)
Sweating	Asthma Attack (cf 0.3)
Chest Pain	Asthma Attack (cf 0.3)
Cool Pale Extremities	Septic Shock (cf 0.3)
Low Blood Pressure	Septic Shock (cf 0.3)
Increased heart Rate	Septic Shock (cf 0.3)
Low Urine Output	Septic Shock (cf 0.1)
Increased Thirst	Hyperglycaemic Attack (cf 0.4)
Increased Urination	Hyperglycaemic Attack (cf 0.2)
Fruity Smelling Urine	Hyperglycaemic Attack (cf 0.2)
Dizziness	Hyperglycaemic Attack (cf 0.3)
Loss of Consciousness	Hyperglycaemic Attack (cf 0.3)
Sweating	Hyperglycaemic Attack (cf 0.1)
Loss of coordinated movements	Stroke (cf 0.3)
Face drooping	Stroke (cf 0.35)
Fully coordinated movements	Stroke (cf -0.5)
Localised Headache	Raptured Brain Aneurysm (cf 0.4)
Nausea and Vomiting	Raptured Brain Aneurysm (cf 0,3)
Blurred Vision	Raptured Brain Aneurysm (cf 0.1)
Double Vision	Raptured Brain Aneurysm (cf 0.1)
Photophobia	Raptured Brain Aneurysm (cf 0.1)
Pain at wound	Wound Infection (cf 0.2)
Redness around wound	Wound Infection (cf 0.2)
Swelling at wound site	Wound Infection (0.2)
Puss from wound	Wound Infection (0.2)
Fever	Wound Infection (0.2)
Low Urine Output	Acute Renal Failure (cf 0.4)
Increased Urination	Acute Renal Failure (cf -1)
Nausea and Vomiting	Acute Renal Failure (cf 0.2)
Nausea and Vomiting	Myocardial Infarction (cf 0.3)
Sweating	Myocardial Infarction (cf 0.3)
Chest Pain	Myocardial Infarction (cf 0.3)
Dizziness	Low Blood Pressure (cf 0.5)
Chest Pain	Low Blood Pressure (cf 0.2)
Dizziness	High Blood Pressure (cf -1)

Table 6-2: Clinical Manifestation Diagnosis rule base

6. 1. 3 Computer Hardware Diagnostics

This knowledge base contains rules about computer diagnostics, and what might be the possible causes if the computer is not working (from the hardware aspect).

6.2 System Testing

System testing involves testing the core functionality of the backward chaining and forward chaining modules of the inference engine. Each test case is performed on a set of rules and the result is then compared to the expected result.

6. 2. 1 Test Case 1: Basic Forward Chaining

In this test case we will check the system's functionality for basic forward chaining, for rules of the form:

$$A \Rightarrow B$$

Test Rule: *If Fuel System contains air then Engine does not start (cf 0.8)*

Symptoms	Working Memory	Possible Conditions
Fuel system contains air (cf 1)	Fuel system contains air (cf 1)	Engine does not start
	Engine does not start (cf 0.8)	

Table 6-3: Test case 1 - Results

6. 2. 2 Test Case 2: Multiple Forward Chaining

This test case will check the system functionality for multiple forward chaining, for rules of these form:

$$A \Rightarrow B$$
$$B \Rightarrow C$$
$$C \Rightarrow D$$

Test Rules: *If car battery is flat then starter motor is not working*

If starter motor is not working then engine does not start
If engine does not start then car is not working

Symptoms	Working Memory	Possible Conditions
Car battery is flat (cf 1)	Car Battery is flat (cf 1)	Engine does not start
	Starter motor is not working (cf 1)	Starter motor is not working
	Engine does not start (cf 1)	Car is not working
	Car is not working (cf 1)	

Table 6-4: Test case 2 - Results

6. 2. 3 Test Case 3: Basic Backward Chaining

This test case will check the system functionality for basic backward chaining:

Test Rule: *If Fuel System contains air then Engine does not start (cf 0.8)*

Initial Goal: *Engine does not start*

Goal	Question(s) Asked / Response	Result
Engine does not start	Fuel System contains air? Yes (cf 1)	Engine does not start (cf 0.8)

Table 6-5: Test case 3 – Results

6. 2. 4 Test Case 4: Mixed Inference, Automatic sub-goal generation and Inheritance

This test case will check how the system handles situations that contain mixed inference, require sub-goal generation and object inheritance. Mixed inference is a combination of both backward chaining and forward chaining. Sub-goal generation is when the system creates new *temporary* goals in order to be able to reach the initial goal

Test Rules: *If car battery is flat then electrical component is not working*
If starter motor is not working then engine does not start
If fuel tank is empty then engine does not start
If engine does not start then car is not working
Headlights are an electrical component
Starter motor is an electrical component

Initial Goal: *Car is not working*

Goal	**Process**	**Result**
Car is not working	Fuel tank empty? No Engine does not start? Unknown	Generated subgoal: engine does not start
Engine does not start	Starter motor not working? Unknown	Generated subgoals: starter motor not working, electrical component not working
Starter motor not working	[no rules]	
Electrical component not working	Car battery flat? Yes	Starter motor not working (cf 1) Electrical component not working (cf 1) *Forward Chaining:* engine does not start (cf 1) car is not working (cf 1)
Engine does not start	Exists in working memory	
Car is not working	Exists in working memory	Initial Goal found

Table 6-6: Test case 4 – Results

6. 2. 5 Test Case 5: Rule Recursion

This test case will check the system on how it handles rule recursion. Rule recursion happens when there is a bi-implication. The system must detect a recursion and skip the rule, and it must make sure that it does not end in an infinite recursive loop.

Test Rules: *If engine is not starting then starter motor is not working (cf 0.2)*
If starter motor is not working then engine is not starting (cf 1)

Goal	Process	Result
Engine is not starting	Starter motor not working? Yes	Engine is not starting (cf 1)

Table 6-7: Test case 5 - Results

The system successfully detects the recursion, and does not try to deduce that starter motor is not working, from the new fact *engine is not working* because it would create an infinite loop.

6. 2. 6 Test Case 6: Combining certainty factors

This test case will check the system functionality when the rule's antecedents contain different certainty factors, and the consequents already exist in working memory and have a certainty factor that needs to be combined with the *output* certainty factor of the rule being processed.

Please refer to Equation 3-4 and Equation 3-5 for equations on combining certainty factors.

Rule: *If Nausea and Vomiting then Myocardial Infarction (cf 0.3)*

Working Memory	Process
Nausea – cf 0.8 Vomiting – cf 1 Myocardial Infarction – cf 0.5	Myocardial Infarction (cf 0.62)

Table 6-8: Test case 6 - Results

Expected Result:

$$output\ cf = \min(0.8, 1)\ x\ 0.3 = 0.24$$

$$result\ cf = 0.24 + 0.5 - (0.24 * 0.5) = 0.62$$

6. 2. 7 Overview

The various test cases explained above test different functionality of the system. These are the basis for any expert system. *Paramenides* passed all test cases successfully. The test cases used have been created in such a way that although they test the basic functionality of the system, the same functionality can be extended to work on much larger rule bases since all rule bases are made up of a similar format.

6.3 Human Evaluation

To better evaluate the system and its functionality, we have opted to create a questionnaire and hand it out to human testers in order to get a much better overview of the system. The aim of these questionnaires is to gather information by letting people use the system and then providing feedback on their experience. The questionnaire is split into three different cases, each case aimed at one of the three main aspects of the system: *backward chaining*, *forward chaining*, and *rule management.* The results obtained are quite interesting.

6. 3. 1 Questionnaire

Listed below is the questionnaire that has been handed to the human testers. We will be using a total of three human testers with each user giving feedback on each test case. The results are then based on the feedback received.

Test Case 1: Backward Chaining

One morning, you went to start your car but it refused to start. You need to know what are the most probable causes. You are to consult Paramenides to help you diagnose the problem.

You are to imagine that you know of the below symptoms, and you are unsure of anything else about the car:

- Headlights were left on.
- Engine compression is likely to be poor.
- There is fuel in the tank.
- Oil grade is correct.
- Weather is not cold.
- Engine is switched off.

Part 1: Use *Paramenides* to perform a *backward chaining* search on your problem, by selecting the problem *engine does not start*.

Part 2: Log the main causes of the problem, through the help of the system.

Part 3: Describe how you found the process.

Test Case 2: Forward Chaining

A patient is not feeling well and would like to consult a doctor. You are to consult Paramenides to help you give an indication what conditions the patient may have.

You are to imagine that you know of the below symptoms, and you are unsure of anything else about the car:

- Nausea
- Vomiting
- Dizziness

Part 1: Use *Paramenides* to perform a *forward chaining* search on your problem, on the *Clinical Manifestation Diagnosis* knowledge base.

Part 2: Log the conclusions (possible diseases) reached, through the help of the system

Part 3: Describe how you found the process

Test Case 3: Rule Management

You are an expert in computer hardware diagnostics and want to contribute to *Paramenides.*

You are to imagine that you use these rules in your mind when troubleshooting computer-related problems:

- If computer is switched on and computer fans not turning then power supply is most probably not working
- If power supply is not working then computer is not working
- If ram is not working then computer is not working
- If nothing visible on screen then it may be either the RAM not working or graphics card not working

- If nothing visible on screen and computer beeps continuously during post then most probably RAM is not working
- If nothing visible on screen and computer does not beep during post then most probably graphics card is not working
- If graphics card is working and nothing visible on screen then monitor is not working

Part 1: Use *Paramenides* to enter a new knowledge base, *Computer Diagnostics*

Part 2: Describe how you found the process and any difficulties encountered.

Part 3: Mention any possible improvements that you think can make the system easier to use.

General Questions

- Do you think it would be more helpful to diagnose problems, as opposed to finding information by searching through the Internet?
- Do you think it can replace an expert, if sufficient information is inputted in the system?

6. 3. 2 Results

User 1 - Test Case 1 (Backward Chaining)

Q: Log the main causes of the problem, through the help of the system

From the explanation diagram I could conclude that there were various problems with the system. I could notice that the main problem were that the *headlights were switched on* and *engine was switched off*. This resulted in the *car battery being flat*. The system also pointed out that several items were not working as a direct result of the car battery. These included the *preheating system, cold start advance mechanism, starter motor, injection pump and stop solenoid*. Another likelihood condition was that the *engine compression is poor*, although this provided a much smaller contribution to the result.

Q: Describe how you found the process

Searching for the initial goal was relatively easy. The system started asking questions related to the problem, and although one might think that some questions are useless, the system provides an explanation to its reasoning when it is finished. The colouring scheme has helped me better identify which of the conditions were met, and thus can provide a visual cue for better understanding how it came to the conclusion.

User 1 - Test Case 2 (Forward Chaining)

Q: Log the conclusions (possible diseases) reached, through the help of the system

- Patient has low blood pressure (cf 0.5)
- Patient has ruptured brain aneurysm (cf 0.49)
- Patient has hyperglycaemic attack (cf 0.2)
- Patient has acute renal failure (cf 0.2)
- Patient has myocardial infarction (cf 0.2)

- Patient has septic shock (0.1)

Q: Describe how you found the process

The process was relatively easy to use, and fast. The search for knowledge bases worked nicely, allowing me to even enter keywords not directly found in the title, like for example *nausea*. The possible problems the patient might be experiencing were outputted as soon as the symptoms were inputted in the system. The results were sorted by the belief the system has in the conditions making it easy to deduce what is most likely to be the problem. I also liked a lot the explanation, which clearly shows the reasoning the system performed to come up with such a conclusion.

User 1 - Test Case 3 (Rule Management)

Q: Describe how you found the process and any difficulties found

The process is quite straightforward to use, and I don't think that one needs more than basic computer usage skills to be able to enter rules. However, one needs to have a brief understanding on how the system works and what it is expecting as inputs before he is able to convert rules into the format required by the system. It is a very repetitive job, and can be quite tedious for a long list of rules.

Q: Mention any possible improvements to make the system easier to use

It would be much easier if the user can enter rules in plain English, and the system somehow analyses the text and converts it automatically into rules. This would eliminate the need for the user to split the expert system into a list of objects, attributes and values, before he can start working on the rules.

User 1 – General Questions

<u>Q: Do you think it would be more helpful to diagnose problems, as opposed to finding information by searching through the Internet?</u>

Yes I think it would be extremely more helpful in diagnosing problems. The internet contains huge amount of information and sometimes you can get mixed up in all the information. It would be an interesting system if the number of knowledge bases is expanded to cater for various different topics and on a larger scale. It can also help because sometimes you do not know exactly the keywords to look for. Especially in the given topic of medical diagnosis, many users do not know the *technical* word initially and are able to describe what they are feeling in layman's terms like *I am feeling dizzy.* It would greatly reduce the time needed for troubleshooting problems, and it could even have other uses that I might not anticipate right now.

<u>Q: Do you think it can replace an expert, if sufficient information is inputted in the system?</u>

I'm not sure if I would be ready to trust a computer program with my health, but it could give a very good indication before you can consult a real-life doctor. It might even allow the doctor (or expert) to diagnose the problem more easily, and could maybe lead to less expenses.

I think the system would be especially useful for less-critical topics, like car diagnostics and electronics, and can be very useful for the DIY enthusiast. It might help him troubleshoot problems without the need of having a great deal of experience, since most DIY hobbyists do not have the advantage of working on various examples to expand their knowledge

User 2 – Test Case 1 (Backward Chaining)

<u>Q: Log the main causes of the problem, through the help of the system</u>

The main cause seems to be the headlights being left on when the car was switched off, resulting in a high chance that the car battery is flat. Due to this, the system pointed out that several components are not working, which ultimately leads to the engine not starting. Apart from these components, another problems seems to be that the engine compression is poor, but one is given the idea that it is more likely that it is due to the car battery being flat than due to the compression, but it must not be eliminated. I think the system worked quite well as the conclusion reached is quite logical.

<u>Q: Describe how you found the process</u>

The process was very intuitive. I found it easy to use and could easily follow up the reasoning behind the system thanks to the explanation system.

User 2 – Test Case 2 (Forward Chaining)

<u>Q: Log the conclusions (possible diseases) reached, through the help of the system</u>

Through the system, one can conclude that the most probable causes are low blood pressure and raptured brain aneurysm. There might also be a slight likelihood that the patient has hyperglycaemic attack, myocardial infarction, and a even decreased likelihood of septic shock.

<u>Q: Describe how you found the process</u>

The process was easy and I could easily follow what are the possible outcomes. The explanation facility helped me understand more how the system came to such a conclusion. The reasoning also seemed very logical to me.

User 2 – Test Case 3 (Rule Management)

Q: Describe how you found the process and any difficulties found.

The process was easy to understand although it was a little bit time consuming.

Q: Mention any possible improvements to make the system easier to use

I think the system performs quite well, although if the process for entering rules could be made faster the system would be slightly better.

User 2 – General Questions

Q: Do you think it would be more helpful to diagnose problems, as opposed to finding information by searching through the Internet?

Yes for sure. I think the idea of having knowledge in this manner can help a lot in finding solutions to the problems one can face. If it includes knowledge on various topics, I think it would be extremely useful and could save a lot of time.

Q: Do you think it can replace an expert, if sufficient information is inputted in the system?

Yes I think it can be used similar to a *personal expert.* Although I think it needs quite a lot of information to be able to replace experts. Also, I think people will still rely on experts to solve their problems but it can help the user have a much idea before visiting an expert.

User 3 – Test Case 1 (Backward Chaining)

<u>Q: Log the main causes of the problem, through the help of the system</u>

One morning, you went to start your car but it refused to start. You need to know what are the most probable causes. You are to consult Paramenides to help you diagnose the problem.

You are to imagine that you know of the below symptoms, and you are unsure of anything else about the car:

- Headlight were left on
- Engine compression poor is likely
- There is fuel in the tank
- Oil grade is correct
- Starter motor is not defective (you replaced it last month)
- Weather is not cold
- Engine is switched off

<u>Q: Describe how you found the process</u>

The process is very simple and intuitive to follow. After defining your problem, all you need to do is answer the simple questions asked by the system and in any case you're not familiar with such a question, the system gives you also the option to omit the question. After going through these set of steps, the system will provide you with a very well structured outlining all your possible problems and their causes.

User 3 – Test Case 2 (Forward Chaining)

<u>Q: Log the conclusions (possible diseases) reached, through the help of the system</u>

The system has outlined that by having the above symptoms, the most probable problem the patient could have is low blood pressure followed by raptured brain aneurysm. Other possible problems which are less likely are:

- Hyperglycaemic attack
- Acute renal failure
- Myocardial Infarction

The least likely problem contributing towards the above symptoms is the possession of septic shock.

<u>Q: Describe how you found the process</u>

The system is very easy to use. All you need to do is provide your symptoms using English like words and the system will outline the stored possible symptoms. By simply choosing your symptoms, the system will update the possible problems which are contributing towards the listed symptoms accordingly .

User 3 – Test Case 3 (Rule Management)

<u>Q: Describe how you found the process and any difficulties found.</u>

The process was easy to follow, but it is a very boring task and can take quite some time.

<u>Q: Mention any possible improvements to make the system easier to use</u>

Making the section for entering rules more easier, and possibly eliminating the need to have to split the objects into different objects, attributes and values

User 3 – General Questions

<u>Q: Do you think it would be more helpful to diagnose problems, as opposed to finding information by searching through the Internet?</u>

Yes it can be very useful to diagnose problems because it can eliminate a lot of searching for information via traditional search engines. It can help you find the solution to a problem more easily.

<u>Q: Do you think it can replace an expert, if sufficient information is inputted in the system?</u>

I think it can be used to replace an expert in certain areas. For example, I like to repair my car myself whenever a fault occurs. When it occurs, I normally try to find information on the internet to find the cause and fix the problem, as I do not have the expertise a mechanic would have to locate the problem by intuition. If the system could include a thorough knowledge base on car diagnostics, it would help me a lot and I would be happy to rely on it since it can explain the reasoning and I can draw my own conclusions on what seems right or not. I think in areas more critical like the medical test case, people would be more skeptical

6.4 Overview

In the *system testing* the aim was to ensure that the basics of the inference engine are working successfully. We have performed various test cases and the inference engine managed to reason correctly on all the provided cases. Recursion and Mixed inference have also been included because those are a more *complicated* form of reasoning. The main reason behind the system testing was to ensure that it works on small cases to bring out any weaknesses in the system. Since a large knowledge base is made up of a larger list of rules, the reasoning will stay the same and thus if one can prove that it is working for smaller knowledge bases, it will also work for larger knowledge bases.

The *human testing* results have proved quite interesting. The three test cases provided to each user tested the system on its three main aspects. Initially, the users were given a brief explanation what the system is about and an overview of how to use it. The certainty factors were also explained to give the users a better idea of how the uncertainty management works, so that they can answer questions more effectively. The overview of the system was relatively short, in order to let them experiment with the system and get feedback on its general feel and ease-of-use. They were then handed the questionnaire, which they were expected to fill in (which were listed previously).

There was consensus by all users that the backward chaining and forward chaining modules were relatively easy to use and worked as expected. All users managed to reach the same conclusions in both forward chaining and backward chaining. Forward chaining required no interpretation of rules as the possible conditions were clearly outlined to the user. Every user managed to understand what were the possible conditions given the symptoms.

The backward chaining system provided a flowchart-like structure and highlighted which nodes contributed to the end result. The users could then follow the tree and be able to see the conclusions that the system has reached. All users managed to conclude that the main problems were the same.

The main weakness of the system seems to be the rule management. Most of the users have agreed that it is a time-consuming job inputting rules, and is the most part that can be improved from the system. The main problem in rule management is to make it easy to use for the users, while being able to parse the rules as need by the knowledge base in order to be able to parse them effectively.

Since the conditions of the rules need to be split into an object-attribute-value triple, users sometimes find it difficult to categorise the problem in that way. We could have enabled the user to enter conditions as free text, however the system is not able to do any real processing on the condition, and to be able to perform forward chaining and backward chaining on the condition, the user must ensure that it written almost the same. Also, no type of inheritance can be done because the system performs inheritance based on the objects and attributes that are being linked together.

This can be seen as a kind of trade-off between ease-of-use and functionality. The ideal seems to be letting the user enter the rule in natural language (plain English), and the system is able to parse it and split into tokens as needed. This would involve analyzing the phrase and retrieving the subject and action amongst others. This is an entirely distinct subject called Natural Language Processing and is not the scope of this dissertation.

In general, the users all agreed that the system could be quite effective and can save a lot of time from searching all over the internet, if it contains a good amount of knowledge bases. Another conclusion that most users reached were that the system would not replace real-life experts, but it would be able to give a much better indication of the problem. This conclusion might have been reached because one of the test cases involved medical diagnosis, and most people would not be confident in letting a computer program decide for their well-being. They would be more comfortable in using the expert system on less-critical subjects like car diagnosis.

One user described the system as a *personal expert*; that would be a very ideal word to describe the general system functionality.

Given the nature of the project, it is not easy to test it on various domains. However, correct functioning of the system depends a lot on the rules in the knowledge bases. The system has no means to detect incorrect rules, and such rules could lead to incorrect diagnosis. The test cases have proved that given correct rules, the system manages to reach a reasonable conclusion. The evaluation of the system was very satisfactory. We got various positive feedback through the questionnaire, and if the number of knowledge bases are increased it can prove extremely useful

6.5 Summary

In this chapter we have provided means to evaluate the functionality of the system. This was split into two sections: *System Testing* and *Human Evaluation.*

System testing involved creating a list of test cases covering the basic functionality of the system. The system was then tested on these cases to confirm its functionality. *Human Testing* involved creating a questionnaire that covers its main functional aspects: *Backward Chaining*, *Forward Chaining*, and *Rule Management.* The questionnaire provided three test cases to the users, who were asked to fill it in based on their experience and results obtained through the system. In general all users reported positive feedback on the backward chaining and forward chaining modules, but said that more could be done to make the rule management faster. The main problems in this are due to the trade-off between ease-of-use and functionality for the inference, and we have to try to balance it out and came up with the implemented methodology.

7 Conclusion & Future Work

7.1 Future Work

The project managed to address various aspects related to expert system. However, mainly due to lack of time, some features had to be left out which would have made the system more functional and complete.

The system could have been tested on a larger population and more knowledge bases. This could give us more insight on any problems that we may not yet have catered for.

The system could implement a *Natural Language Processing* engine that is able to analyse phrases and be able to deduce the object, attribute and value it is acting upon. Natural Language Processing is a vast topic on its own and commands extensive research on its various subjects. This is one of the main reasons why this was not implemented in our project, as it is not directly related to expert systems and would add a considerable amount of work to the system. NLP would be extremely useful when the user is entering phrases to search for goals and knowledge bases, and especially for rule management. Human evaluation concluded that enabling users the enter rules in plain English would make it much easier and faster. Enter a rule as "If weather is cold and starter motor is not working then car is not working", then having to split into objects, attributes and values and entering those as the antecedents and consequents is a more logical process for the non-computer savvy user.

Another feature the system could improve upon is *rule diagnostics*. These would cater for *logical* errors in the rule base, which cannot be detected by the testing methodologies mentioned in the evaluation. Although the current explanation can visually show the rule base and be used for such diagnostics, such specific tools would help in pointing out rules that do not make sense.

Another feature that would prove beneficial if *Paramenides* is to be used by the general public is moderation of the knowledge bases and versioning control. However these features involve no research to be done but add considerably to the work involved in the system. This was not the point of this dissertation, and has thus been left out.

7.2 Concluding Remarks

The project has served as a great tool for learning about expert systems to the author. We have made an extensive literature review on various aspects related to expert systems, encompassing a good number of topics in such a broad subject. Understanding the inner workings of an expert system requires intricate knowledge about logic, fundamentals of expertise and knowledge representation. Expert systems can nowadays be considered a *buzz* word and sometimes it is abused and used out of context.

We have created an expert system that is available to the general public through a web-based interface, which requires no installation and configuration from the user's end. The expert system is able to perform both backward and forward chaining inference. It also provides a visual explanation system for its reasoning. We have also seen that the concept of uncertainty is required in such an expert system, and we have analysed the various uncertainty management methodologies and implemented the most methodology deemed appropriate.

The system also mimics the very successful idea on which *Wikipedia*[82] is based, by allowing its users can to manage knowledge bases and insert their expertise.

Expert systems have always been considered out-of-reach, and have only been used by large companies who could afford their development and maintenance. The aim was to make expert systems more available to the general public and through Paramenides, we hope to create a global learning resource for anyone, which is created from humans to humans. The final aim is to create an online manifestation of human expertise that is permanent and freely available.

The project has managed to reach most of the aims mentioned earlier in the first chapter. It is easily accessible through the internet, is able to cater for generic knowledge bases, able to perform both backward and forward chaining inference and can explain its reasoning quite well. Another aim of the system was to create an easy-to-use interface to be able to cater for a large number of users. While the frontend interface that performs the backward and forward chaining is quite easy to use, the rule management interface can be slightly confusing and time consuming. Improvements to this interface have been mentioned in the future work.

Finally, we will conclude this dissertation with a quote from John Naisbitt, an American author and public speaker (59), which is extremely relevant to the topic in question and the age we live in:

> "*We are drowning in information but starved for knowledge.*"

[82] Wikipedia (http://www.wikipedia.org) is an online encyclopedia where its users can anonymously contribute to its articles.

Bibliography

1. Story. *Writing.* [Online] [Cited: April 5, 2009.] http://www.mesopotamia.co.uk/writing/story/sto_set.html.

2. Definition of 'Expert'. *The Free Online Dictionary.* [Online] [Cited: April 10, 2008.]

3. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison-Wesley, 1998. pp. 1-2. 0-201-87686-8.

4. —. *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison-Wesley, 1999. 0-201-87686-8.

5. **Simpson, Stephen G.** *Mathematical Logic.* Department of Mathematics, The Pennsylvania State University. 2008. pp. 3-22.

6. **Franklin, J. and Daoud, A.** *Proof in Mathematics: An Introduction.* s.l. : Quakers Hill Press. 1-876192-00-3.

7. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison Wesley, 1999. pp. 145-148. 0-201-87686-8.

8. **Laun, Wolfgang.** *Predicate Calculus and Jess.* Vienna, Austria : Thales Rail Signalling GesmbH, 2009.

9. *The Calculus of Logic.* **Boole, George.** 1848, Cambridge and Dublin Mathematical Journal, Vol. 3.

10. **Ahmadi, Mahnaz.** *Boolean Algebra.* McMaster University. 2004.

11. **Guarino, Nicola and Giaretta, Perdaniele.** *Ontologies and Knowledge Bases.* National Research Council, University of Padova. Padova, Italy : s.n.

12. **P. Patel-Schneider, P. Hayes, I. Horrocks.** *OWL Web Ontology Langauge Semantics and Abstract Syntax.*

13. **D. Fensel, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider.** *OIL: An Ontology Infrastructure for the Semantic Web.* s.l. : IEEE Intelligent Systems, 2001.

14. SWCLOS: A Semantic Web Processor on Common Lisp Object System. *Semantic Web.* [Online] [Cited: April 10, 2009.] http://iswc2004.semanticweb.org/demos/32/.

15. **Tim Berners-Lee, Nigel Shadbolt, Wendy Hall.** *The Semantic Web Revisited.* s.l. : IEEE Intelligent Systems, 2006.

16. **Randall Davis, Howard Shrobe, Peter Szolovits.** *What is a Knowledge Representation?* MIT AI Lab, MIT AI Lab and Symbolics, Inc., MIT Lab for Computer Science. 1993.

17. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* 4th Edition. s.l. : Course Technology, 2005. p. 71. 0-534-38447-1.

18. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison-Wesley, 1998. pp. 294-314. 0-201-87686-8.

19. **Boden, Margaret A.** *Mind as Machine: A history of cognitive science.* s.l. : Oxford University Press, 2006. p. 840. 0199241449.

20. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison Wesley, 1999. p. 11. 0-201-87686-8.

21. **Segaller, Stephen.** *Nerds 2.0.1: A Brief History of the Internet.* s.l. : TV Books, 1998. 1575001063.

22. World Internet Usage Statistics News and World Popoulation Stats. *Interent World Stats.* [Online] [Cited: May 1, 2009.] http://www.internetworldstats.com/stats.htm.

23. **Dictionaries, Oxford.** *Concise Oxford English Dictionary.* 11th Edition. s.l. : Oxford University Press, USA, 2008. 978-0199548415.

24. **A. Aamodt, E. Plaza.** *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* University of Trondheim, Institut d'Investigacio en Intelligencia Artificia, CSIC. 1994.

25. **McCarthy, John.** *Some expert systems need common sense.* Computer Science Department, Stanford University. 1984.

26. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. 1999. pp. 1-2. 0-201-87686-8.

27. **Barr Avron, Edward A. Feigenbaum.** *The Handbook of Artificial Intelligence.* s.l. : Heuris Press, 1981. 08657600547.

28. How Intelligent is Deep Blue? *New York University.* [Online] 1997. [Cited: April 11, 2009.]

http://www.nyu.edu/gsas/dept/philo/courses/mindsandmachines/Papers/mcdermott.html.

29. **Ayse Pinar Saygin, Ilyas Cicekli, Varol Akman.** *Turing Test: 50 Years Later.* Department of Cognitive Science, Department of Computer Engineeing,, University of California, Bilkent University.

30. **Cawsey, Alison.** *Rule-Based Systems.* Department of Computing and Electrical Engineering, Heriot-Watt University. Edinburgh : s.n.

31. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. pp. 78-81. 0-534-38447-1.

32. **Jackson, Peter.** *Introduction to Expert Systems.* 3rd Edition. s.l. : Addison Wesley, 1998. pp. 102-105. 0-201-87686-8.

33. **Sowa, John F.** *Semantic Networks.* 1987.

34. **Peirce, Charles Sanders.** *The Collected Papers of C. S. Peirce.* 1931-35. pp. 293-341. Vol. 4, Available online at http://www.existentialgraphs.com/peirceoneg/existentialgraphs4.372-417.htm and http://www.existentialgraphs.com/peirceoneg/existentialgraphs4.418-529.htm.

35. **Sowa, John F.** *Conceptual Graphs.* 1999.

36. **Sasu, Lucian.** *Entity-Attribute-Value Modelling.* Faculty of Mathematics and Informatics, University of Brasov. 2008.

37. **Waterman, Donald Arthur.** *A Guide to Expert Systems.* s.l. : Addison-Wesley, 1987. pp. 73-79. 0201083132.

38. **Richard Fikes, Tom Kehler.** *The Role of Frame-Based Representation in Reasoning.* s.l. : Communications of the ACM, 1987.

39. **Jackson, Peter.** *Introduction to Expert Systems.* s.l. : Addison-Wesley, 1999. pp. 87-93. 0-201-87686-8.

40. **Richard Forsyth, Chris Naylor.** *The hitch-hiker's guide to artificial intelligence.* s.l. : Taylor & Francis, 1986. pp. 22-28. 0412281309.

41. **Horatiu Cirstea, Claude Kirchner, Michael Moossen, Pierre-Etienne Moreau.** *Production Systems and Rete Algorithm Formalisation.* Reseau National des Technologies Logicielles. 2004.

42. **Forgy, Charles.** *A network match routine for production systems.* 1984.

43. —. *On the efficient implementation of production systems.* Carnegie-Mellon University. 1979. Ph.D. Thesis.

44. *Rete: A Fast Algorithm for the ManyPattern/Many Object Patern Match Problem.* **Forgy, Charles.** 19, 1982, Artificial Intelligence, pp. 17-37.

45. *The Execution Kenrel of RC++: Rete*, A Faster Rete with treat as a special case.* **Ian Wright, James Marshall.** 1, 2003, International Journal of Intelligent Games and Simulation, Vol. 2, pp. 36-48.

46. **Doorenbos, Robert B.** *Production Matching for Large Learning Systems.* Computer Science Department, Carnegie Mellon University. 1995.

47. —. *Production Matching for Large Learning Systems.* Computer Science Department, Carnegie Mellon University. 1995. pp. 7-33.

48. **Jackson, Peter.** *Introduction to Expert Systems.* s.l. : Addison-Wesley, 1998. p. 166. 0-201-87686-8.

49. **Colin Howson, Peter Urbach.** *Scientific Reasoning.* s.l. : Open Court, 2005. 081269578X.

50. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. pp. 207-234. 0-534-38447-1.

51. **Shafer, Glenn.** *Dempster-Shafer Theory.* 2002. (Available online at: http://www.glennshafer.com/assets/downloads/articles/article48.pdf).

52. **Kari Sentz, Scott Ferson.** *Combination of Evidence in Dempster-Shafer Theory.* Sandia National Laboratories. 2002. SAND 2002-0835.

53. **Bruce G. Buchanan, Edward H. Shortliffe.** *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* [ed.] Edward H. Shortliffe. s.l. : Addison-Wesley, 1987. pp. 263-271. 978-0201101720.

54. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. pp. 268-279. 0-534-38447-1.

55. **Hellmann, Martin.** *Fuzzy Logic Introduction.* Computer Science, University of California. Berkely : s.n., 2001.

56. MYCIN (Artificial Intelligence Program). *Britannica Online Encyclopedia.* [Online] [Cited: April 12, 2009.] http://www.britannica.com/EBchecked/topic/400270/MYCIN.

57. MYCIN. *University of Surrey.* [Online] [Cited: April 12, 2009.] http://www.computing.surrey.ac.uk/ai/PROFILE/mycin.html.

58. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. pp. 192,212-217,227,240,242,245. 0-534-38447-1.

59. John Naisbitt Quotes. *BrainyQuote.* [Online] [Cited: April 18, 2009.] http://www.brainyquote.com/quotes/quotes/j/johnnaisbi382513.html.

60. *Encyclopedia Britannica.* s.l. : Encyclopedia Britannica, 2004. 978-0852299616.

61. Ontology Components. *Wikipedia.* [Online] [Cited: April 9, 2009.] http://en.wikipedia.org/wiki/Ontology_components.

62. Web Ontology Language. *StateMaster.* [Online] [Cited: April 9, 2009.] http://www.statemaster.com/encyclopedia/Web-Ontology-Language.

63. Flip-flop (electronics). *Wapedia.* [Online] [Cited: April 10, 2009.] http://wapedia.mobi/en/Flip-flop_(electronics)#1..

64. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. pp. 12-14. 0-534-38447-1.

65. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 10. 0-534-38447-1.

66. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 69. 0-534-38447-1.

67. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 70. 0-534-38447-1.

68. **Agnar Aamodt, Enric Plaza.** *Case-Based Reasoning: Foundational issues, Methodological Variations, and System Approaches.* University of Trondheim, Institut d'Investigacio en Intelligencia Artificial. 1994.

69. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 72. 0-534-38447-1.

70. **Sowa, John F.** *Conceptual Graphs.* 1999.

71. **Richard Fikes, Tom Kehler.** *The Role of Frame-based Representation in Reasoning.* s.l. : Communications of the ACM, 1985.

72. **Horatiu Cirstea, Claude Kirchner, Michael Moossen, Pierre-Etienne Moreau.** *Production Systems and Rete Algorithm Formalisation.* Reseau National des Technologies Logicielles. 2004.

73. **Doorenbos, Robert B.** *Production Matching for Large Learning Systems.* Computer Science Department, Carnegie Mellon University. 1995. p. 14.

74. **Joseph C. Giarratano, Gary D. Riley.** *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. 0-534-38447-1.

75. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 285. 0-534-38447-1.

76. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 272. 0-534-38447-1.

77. —. *Expert Systems: Principles and Programming.* s.l. : Course Technology, 2005. p. 299. 0-534-38447-1.

78. The Burning of the Library of Alexandria. *eHistory.* [Online] [Cited: April 11, 2009.] http://ehistory.osu.edu/world/articles/ArticleView.cfm?AID=9.

79. Glossary. *Net Pedagogy Portal.* [Online] [Cited: Aprl 29, 2009.] http://www.thewebworks.bc.ca/netpedagogy/glossary.html.

Appendices

A. Contents of CD

The CD contains the project files as well as the website deployment files if one would wish to install the website on his own web server. It also contains copies of the filled in questionnaires, and knowledge bases obtained for testing.

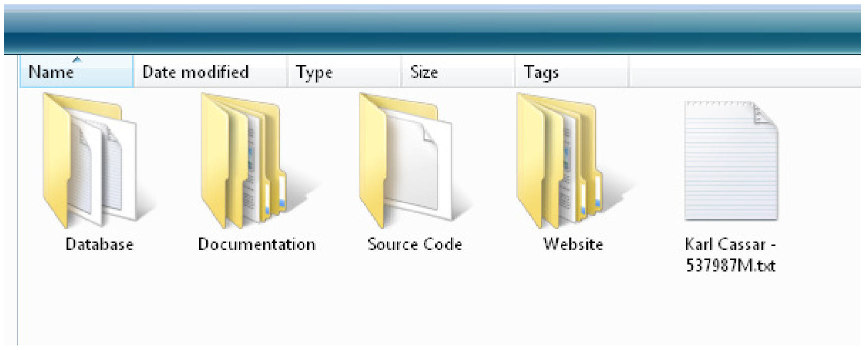


Figure A-1: Contents of CD

The above figure shows the contents of the CD. Below is a brief overview:

- **Database**: Contains SQL Script files that can be used to create the database. It contains two files: *Structure_data.sql* contains the database structure and some sample data, while *structure_only.sql* contains just the database structure.
- **Source Code**: Contains the source code of the project
- **Website**: Contains the deploy files for the website, should the reader wish to install the website on his own web server
- **Documentation:** A soft-copy of this dissertation in Word 97-03, Word 07 and PDF format.

B. Installation Notes

Since the project distributable is a website, it is not a simple installation procedure to use the website. We recommend that one uses the online website which is publicly accessible at http://www.paramenides.com.

However, if one needs to install it on his own web server, listed below are the requirements:

- Microsoft .Net Framework v3.5
- Microsoft IIS6/7
- ASP.Net
- MySQL v5.1.33 or greater

It is quite lengthy to describe the process involved in installing each of the required software components. The user will be referred to online resources that can serve as a guide to installation.

B. 1. Installation Guides

Microsoft .Net Framework v3.5:

http://www.microsoft.com/downloads/details.aspx?FamilyId=333325FD-AE52-4E35-B531-508D977D32A6&displaylang=en

IIS 6 (Windows Server 2003 / XP)

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/750d3137-462c-491d-b6c7-5f370d7f26cd.mspx?mfr=true

IIS 7 (Windows Server 2008 / Vista)

http://www.windowsnetworking.com/articles_tutorials/Installing-IIS-70.html

Microsoft ASP.Net (IIS6 – Windows Server 2003 / XP)

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/library/IIS/7ecaa5f3-5499-4887-8c9d-00aba71125df.mspx

Microsoft ASP.Net (IIS 7 – Windows Server 2008 / Vista)

http://learn.iis.net/page.aspx/472/how-to-install-aspnet-11-with-iis7-on-vista-and-windows-2008/

MySQL:

http://dev.mysql.com/doc/refman/5.1/en/installing.html

B. 2. Website Deployment

Once all required components have been installed, one needs to deploy the website in the web server. This involves copying the website files found in [CD\Website] to a folder on the web server, and creating a new website in IIS.

For more information on this process the reader is referred to http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/33e0a51a-5f8a-40f2-9923-cdd604e1a812.mspx?mfr=true

The user needs to import the database into MySQL. The database scripts can be found in [CD\Database\]:

- *strucutre_only.sql*: Contains only the database structure, void of data
- *structure_data.sql*: Contains both database structure and some sample data, including the test user.

The following link refers to a guide on importing databases in MySQL: http://www.clockwatchers.com/mysql_dump.html

Finally, a new username and password must be created in MySQL, giving FULLL access to the *expertsystem_fyp* database. Following are the credentials required:

Username: ***expertsystems***

Password: ***expertsystems***

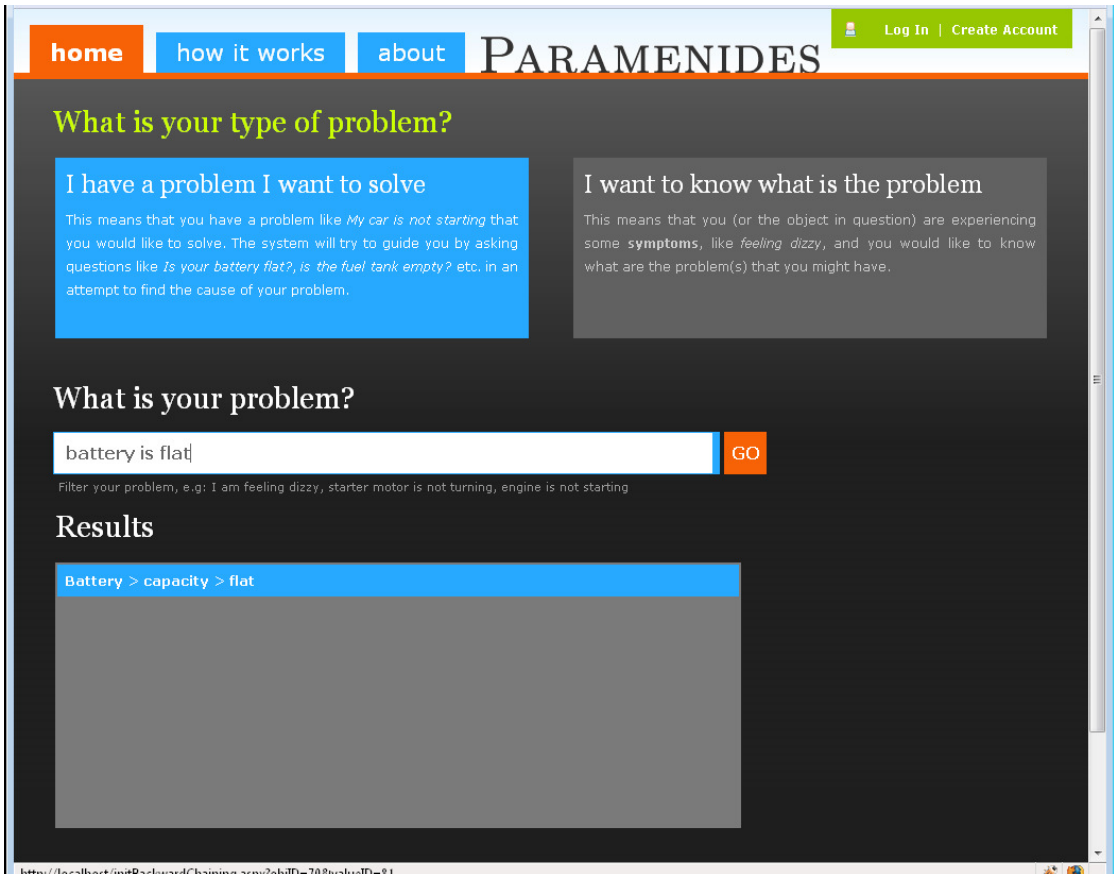
The following guide contains information on how to add a new user in MySQL: http://dev.mysql.com/doc/refman/5.1/en/adding-users.html

Preferably, you should use a GUI tool like MySQL Administrator (http://dev.mysql.com/downloads/gui-tools/5.0.html) to make the process easier.

C. User Manual

The user guide will be split into several different sections, each explaining the main components of the system. The website is mainly split into two main parts: Frontend and Member's Area and each part is then sub-divided further.

C. 1. Frontend



What is your type of problem?

I have a problem I want to solve

This means that you have a problem like *My car is not starting* that you would like to solve. The system will try to guide you by asking questions like *Is your battery flat?, is the fuel tank empty?* etc. in an attempt to find the cause of your problem.

I want to know what is the problem

This means that you (or the object in question) are experiencing some **symptoms**, like *feeling dizzy*, and you would like to know what are the problem(s) that you might have.

What is your problem?

battery is flat

GO

Filter your problem, e.g: I am feeling dizzy, starter motor is not turning, engine is not starting

Results

Battery > capacity > flat

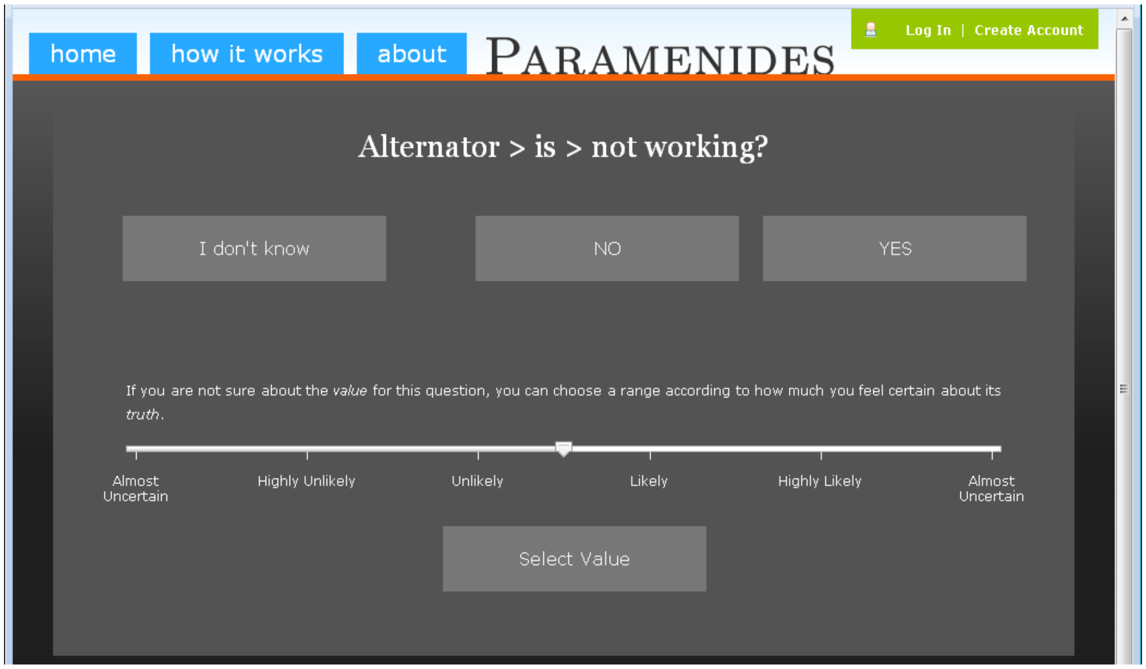
Figure C-1: Main page

From the front page, the user can choose to solve two different types of problem solving: Backward Chaining and Forward Chaining. The currently selected method in

Figure C-1 has the backward chaining selected. The user can then enter keywords based on his problem in the search field, and the system will display the conditions stored in the knowledge base in the results.

C. 1. 1. Backward Chaining Search

The user selects a problem, like for example *Battery > is > flat* (The result in Figure C-1). The system will then ask questions based on the selected goal.



If you are not sure about the *value* for this question, you can choose a range according to how much you feel certain about its *truth*.

Figure C-2: A question asked by the system

The user is then presented with various answers that he may reply to, to the question being asked. The top three buttons – *I don't know, No* and *Yes* are answers based on discreet values, and should be used when the user is either certain of the answer, or he has no idea. The bottom part contains a slider that allows the user to enter his belief in the outcome. Please note that this section is not always visible because not every answer can contain with uncertainty. Some answers only commend a *Yes, No,* or *I don't know* response.

Once the process is ready, the system will provide an explanation to its reasoning.

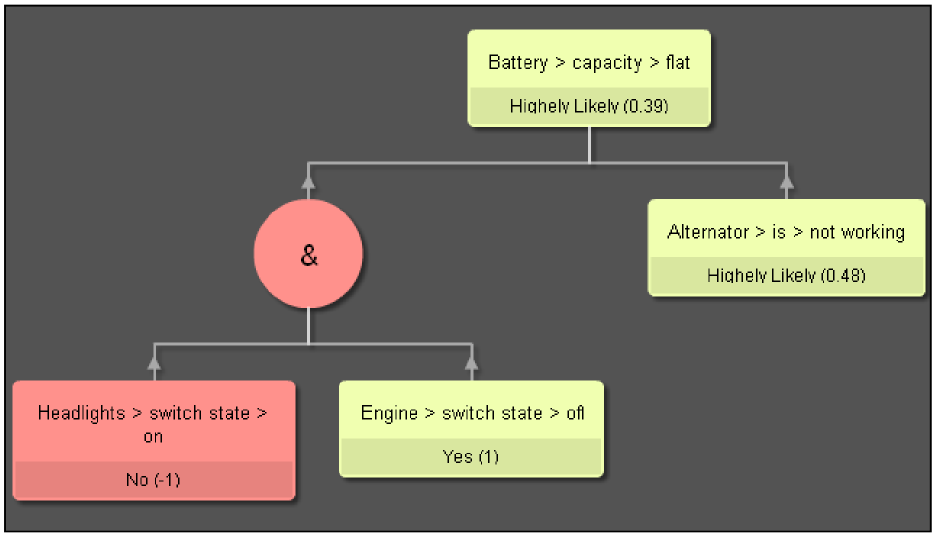
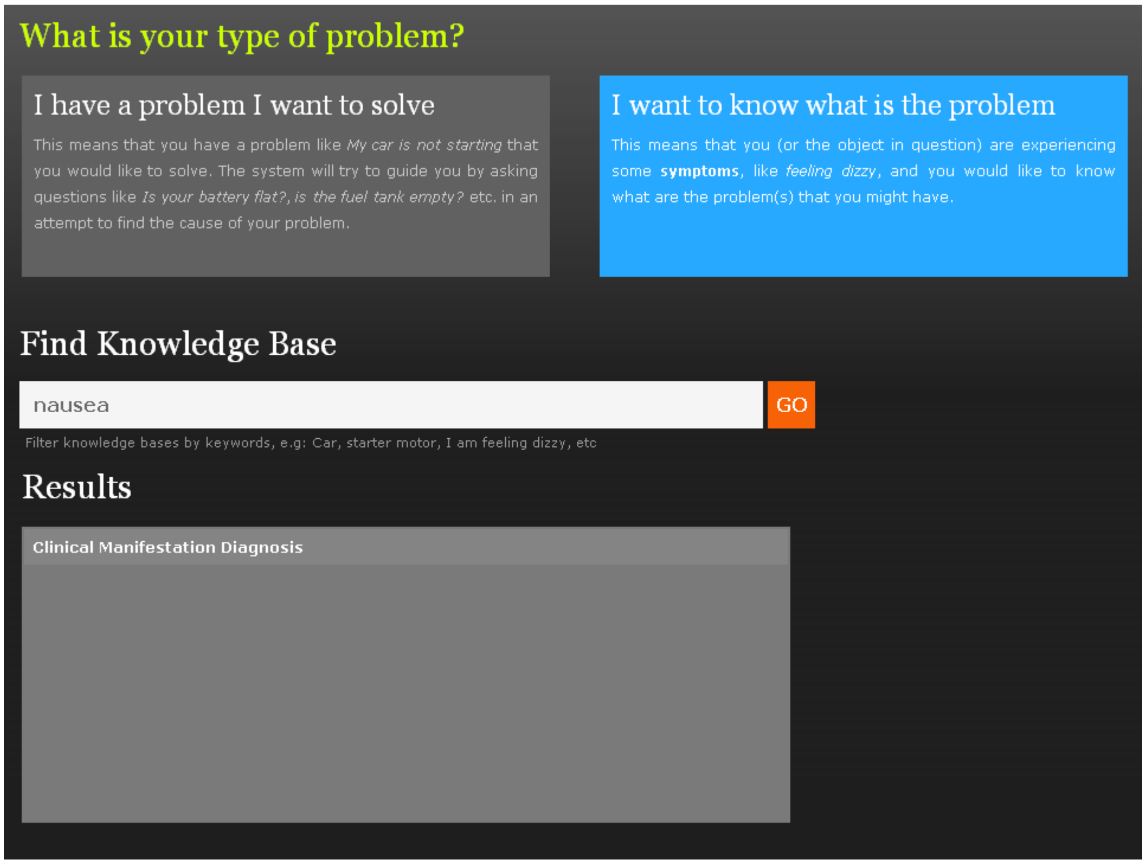

Battery > capacity > flat
Highely Likely (0.39)
&
Alternator > is > not working
Highely Likely (0.48)
Headlights > switch state > on
No (-1)
Engine > switch state > off
Yes (1)

Figure C-3: An explanation for the simple example being used

The user can see the different conditions tested by the system, and they are marked green for *positive* nodes, and red for *negative* nodes. The green nodes are contributing towards their parent node's conditions while the *red* nodes are not contributing at all. From the above graph, we can conclude that the main reason that the capacity is flat is because that the alternation is highly likely that it is not working. The numbers in the brackets () show the certainty factor, which is a numerical number from -1 to 1 which shows the degree of belief in the condition, with -1 representing absolute negativity ('No' answer) and +1 representing absolute positive ('Yes' answer).

C. 1. 2. Forward Chaining Search



What is your type of problem?

I have a problem I want to solve

This means that you have a problem like *My car is not starting* that you would like to solve. The system will try to guide you by asking questions like *Is your battery flat?, is the fuel tank empty?* etc. in an attempt to find the cause of your problem.

I want to know what is the problem

This means that you (or the object in question) are experiencing some **symptoms**, like *feeling dizzy*, and you would like to know what are the problem(s) that you might have.

Find Knowledge Base

nausea

GO

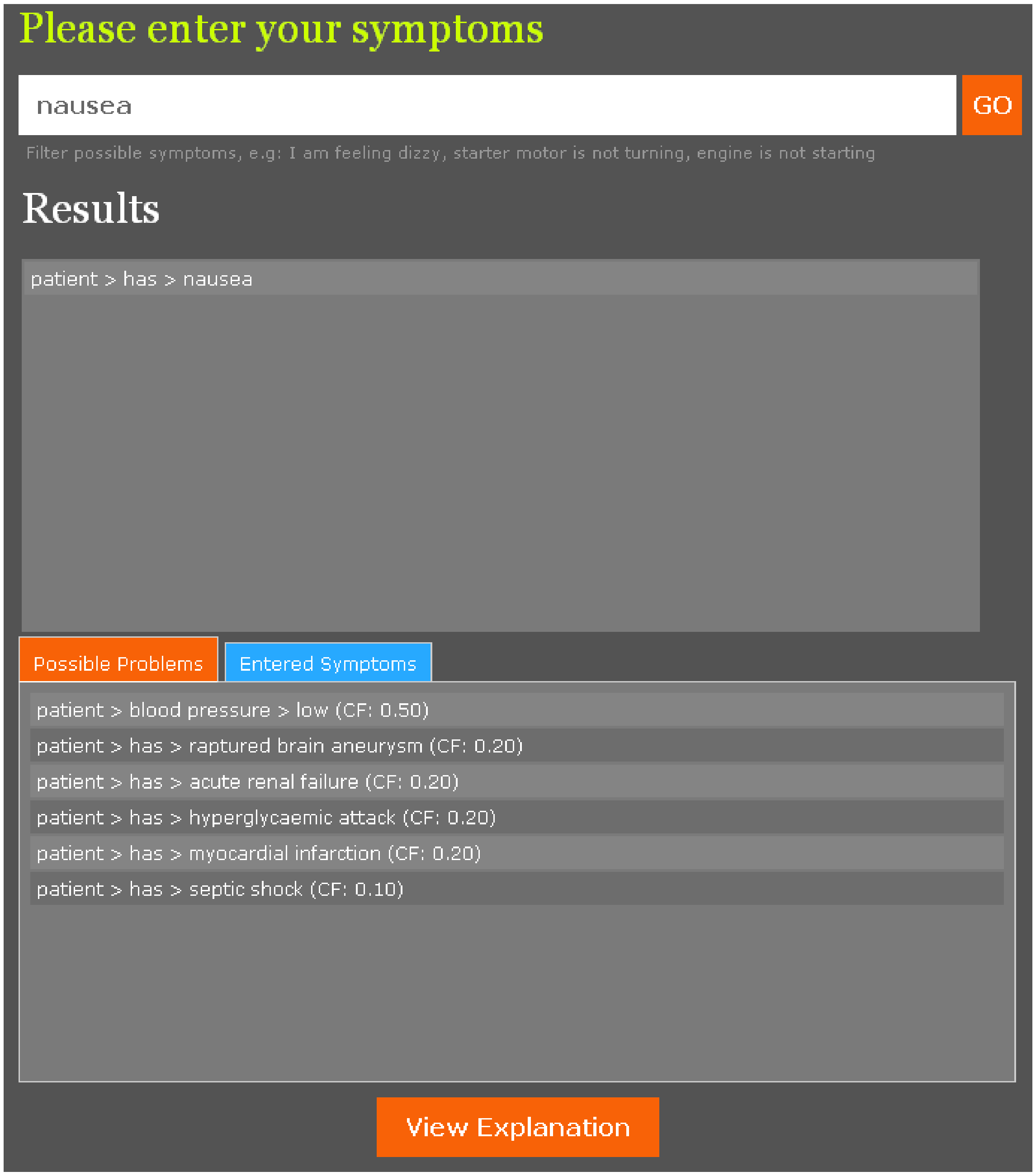
Filter knowledge bases by keywords, e.g: Car, starter motor, I am feeling dizzy, etc

Results

Clinical Manifestation Diagnosis

Figure C-4: Forward chaining enabled, and 1 knowledge base result

A forward chaining search can also be performed from the frontend. Figure C-4 shows the user searching for a knowledge base containing information on the problem *nausea*. The system has found one related knowledge base. The user can then click on the knowledge base to start entering his symptoms

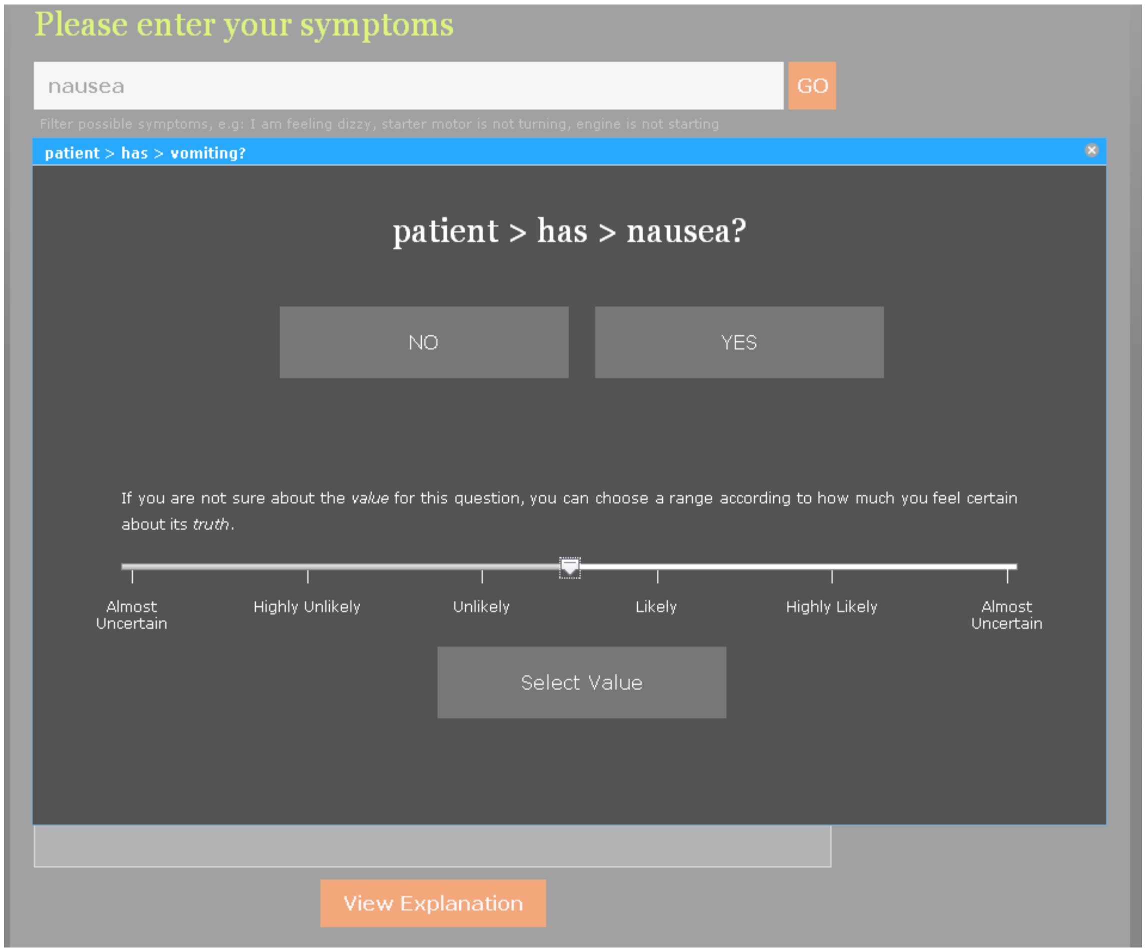

Please enter your symptoms
nausea
GO
Filter possible symptoms, e.g: I am feeling dizzy, starter motor is not turning, engine is not starting
Results
Possible Problems
Entered Symptoms
View Explanation

patient > has > nausea

patient > blood pressure > low (CF: 0.50)
patient > has > raptured brain aneurysm (CF: 0.20)
patient > has > acute renal failure (CF: 0.20)
patient > has > hyperglycaemic attack (CF: 0.20)
patient > has > myocardial infarction (CF: 0.20)
patient > has > septic shock (CF: 0.10)

Figure C-5: Forward-Chaining screenshot, with several symptoms already entered and possible problems found

The user can enter symptoms by first entering keywords related to his symptoms in the search field. The system will then look for related conditions, and present them to the user in the results box. The user can then select a result and enter its value through a dialog that is now visible on the screen (Figure C-6). The entered value can be similar to the questions asked by the backward chaining system, and can include one of Yes and No (I don't know is disabled since the user has selected the condition). The user can also enter the value based on his certainty.


Please enter your symptoms
nausea
GO
Filter possible symptoms, e.g: I am feeling dizzy, starter motor is not turning, engine is not starting
patient > has > vomiting?
patient > has > nausea?
NO
YES
Almost Uncertain
Highly Unlikely
Unlikely
Likely
Highly Likely
Almost Uncertain
Select Value
View Explanation

If you are not sure about the *value* for this question, you can choose a range according to how much you feel certain about its *truth*.

Figure C-6: Forward chaining module dialog for a value

The user can also choose a *possible problem* from the list shown, and the system will perform a backwards chaining search with the selected problem as the goal. This can prove useful to check with more certainty about a problem, since the system is then able to guide you with related questions/symptoms that you may not have thought about.

Apart from entering symptoms, the user can also choose to view the explanation for the current conclusions. This can be done by clicking on the **View Explanation** button.

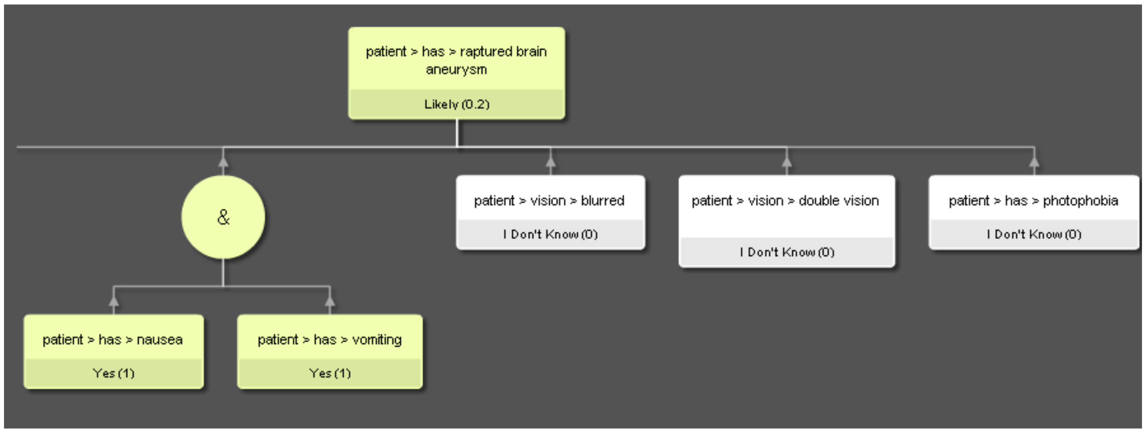

patient > has > raptured brain aneurysm
Likely (0.2)
&
patient > vision > blurred
I Don't Know (0)
patient > vision > double vision
I Don't Know (0)
patient > has > photophobia
I Don't Know (0)
patient > has > nausea
Yes (1)
patient > has > vomiting
Yes (1)

Figure C-7: Explanation for one of the conclusions reached

As one can see, the explanation is similar to the one given by the backward chaining module. However, in forward chaining the *goals* are the possible problems as opposed to the goal selected by the user in backward chaining.

From the explanation, the user can also choose to go back to the forward chaining system to continue entering facts.

C. 2. Member's Area

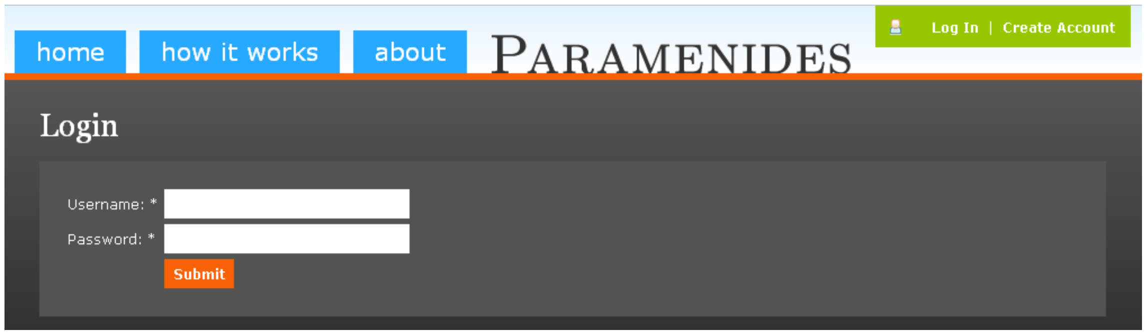


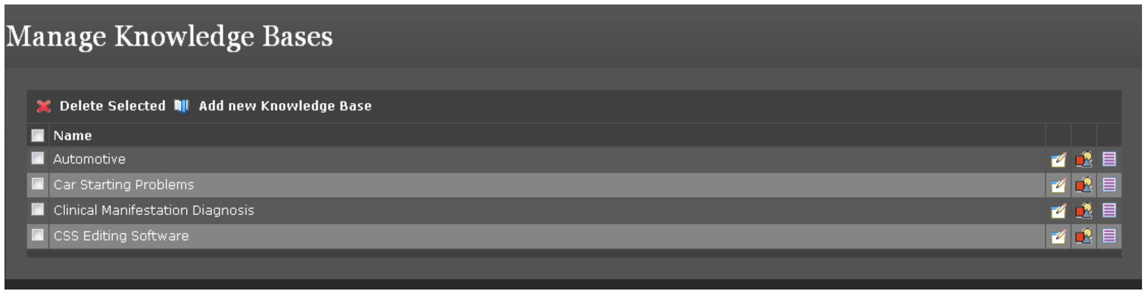
Figure C-8: Login screen, showing the top-right panel

From the top right panel (in green), the users can choose to log in or create an account with the system. For testing purposes, the users can use the already created account with:

Username: test

Password: test

Since the system currently offers no moderation system for its knowledge bases, all users have access to all knowledge bases. The project's aim is a proof of concept, and if the website is to be deployed on a larger scale it must include such a feature that is also explained in more detail in the *Future Work* section in the final chapter.



Name
Automotive
Car Starting Problems
Clinical Manifestation Diagnosis
CSS Editing Software

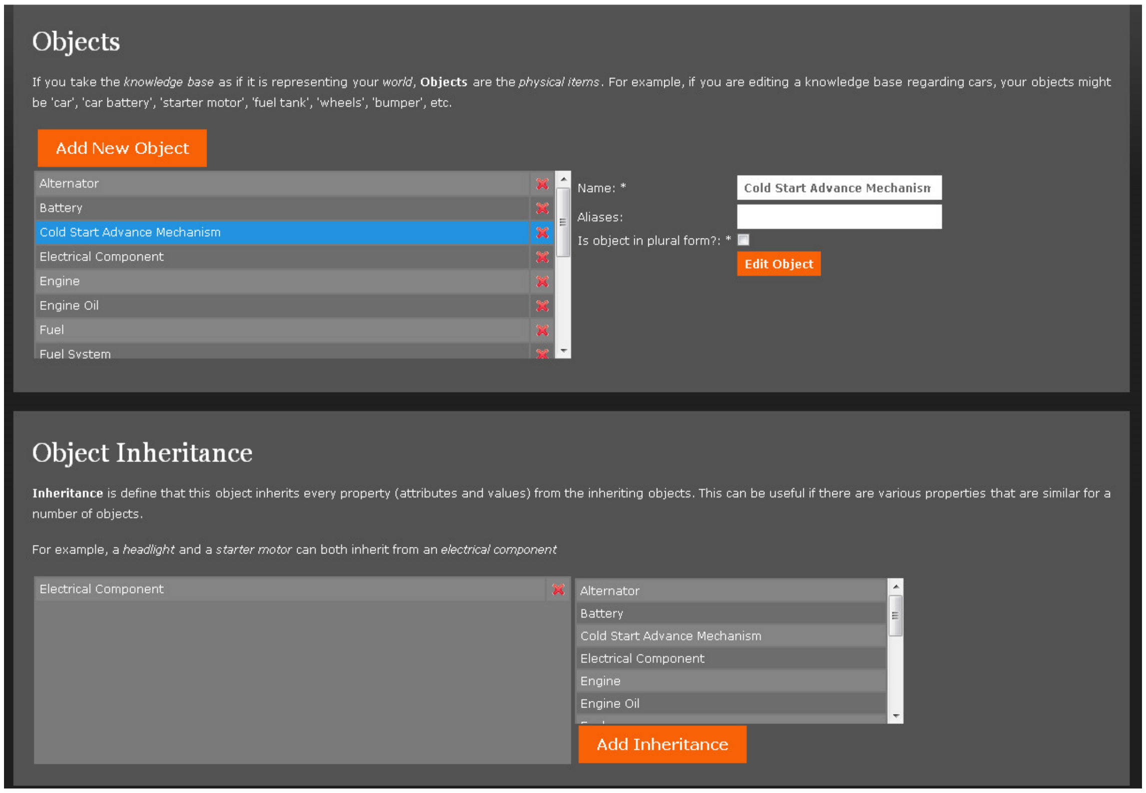
Figure C-9: Knowledge Base Listing

From this screen, the user can choose to create new knowledge bases and manage existing ones. The middle icon, consisting of a set of shapes allows you edit the objects, attributes and values of the knowledge base, while the rightmost icon (the blue listing) allows you to manage the rules.

C. 2. 1. Managing objects, attributes and values

From the next screen, one can enter the objects, attributes and values. Each object can have a list of attributes, while each attribute can have a list of values. These can then be used form a condition in the form of *Object > Attribute > Value*, such as *Car > Is > Not Working.*

Apart from entering these values, users can also define *inheritance* between objects. An object A that inherits from object B means that it will take all of Object B's attributes and values also as its own. The following screenshots show the various sections for managing these object-attribute-value triples.



Objects

If you take the *knowledge base* as if it is representing your *world*, **Objects** are the *physical items*. For example, if you are editing a knowledge base regarding cars, your objects might be 'car', 'car battery', 'starter motor', 'fuel tank', 'wheels', 'bumper', etc.

Alternator
Battery
Cold Start Advance Mechanism
Electrical Component
Engine
Engine Oil
Fuel
Fuel System

Name: * Cold Start Advance Mechanism

Aliases:

Is object in plural form?: *

Object Inheritance

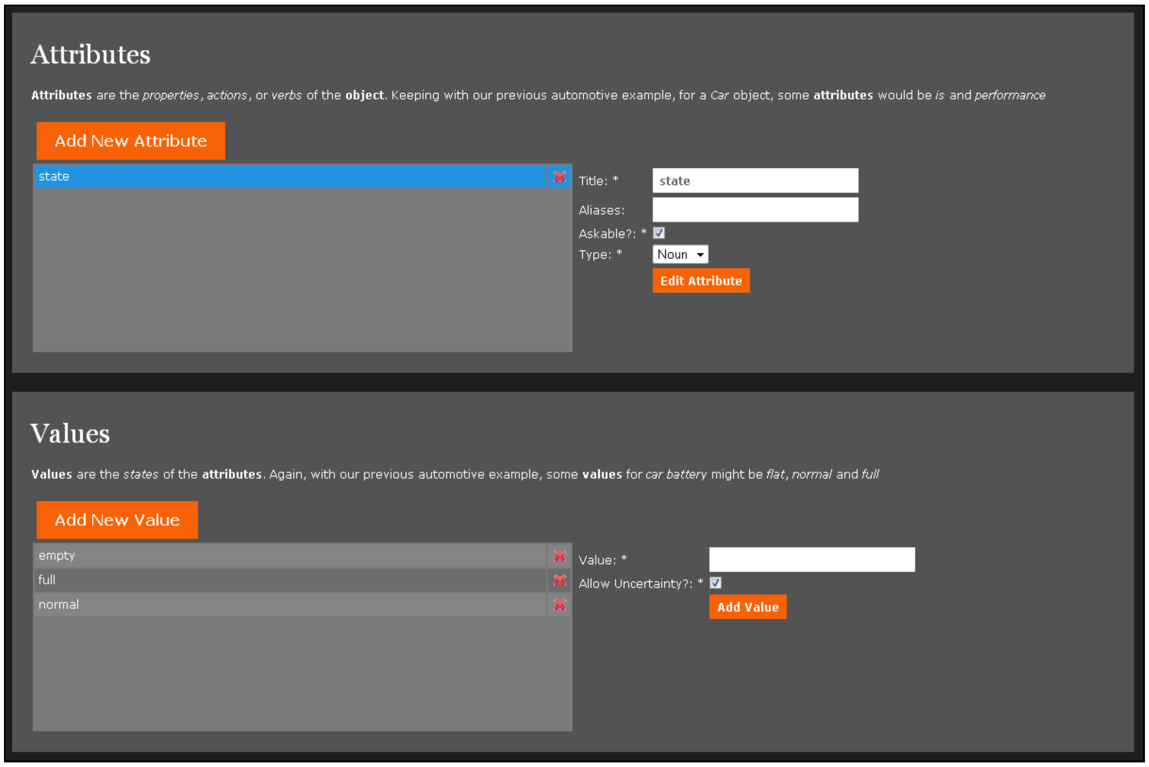
Inheritance is define that this object inherits every property (attributes and values) from the inheriting objects. This can be useful if there are various properties that are similar for a number of objects.

For example, a *headlight* and a *starter motor* can both inherit from an *electrical component*

Electrical Component

Alternator
Battery
Cold Start Advance Mechanism
Electrical Component
Engine
Engine Oil

Figure C-10: Screenshot showing the objects, and object inheritance of the selected object



Attributes

Attributes are the *properties*, *actions*, or *verbs* of the **object**. Keeping with our previous automotive example, for a *Car* object, some **attributes** would be *is* and *performance*

state

Title: * state

Aliases:

Askable?: * ✓

Type: * Noun

Values

Values are the *states* of the **attributes**. Again, with our previous automotive example, some **values** for *car battery* might be *flat*, *normal* and *full*

empty

full

normal

Value: *

Allow Uncertainty?: * ✓

Figure C-11: Screenshot showing the attributes of the selected object, and the values of the selected attribute. The selected object is *fuel tank*

C. 2. 2. Rule Management

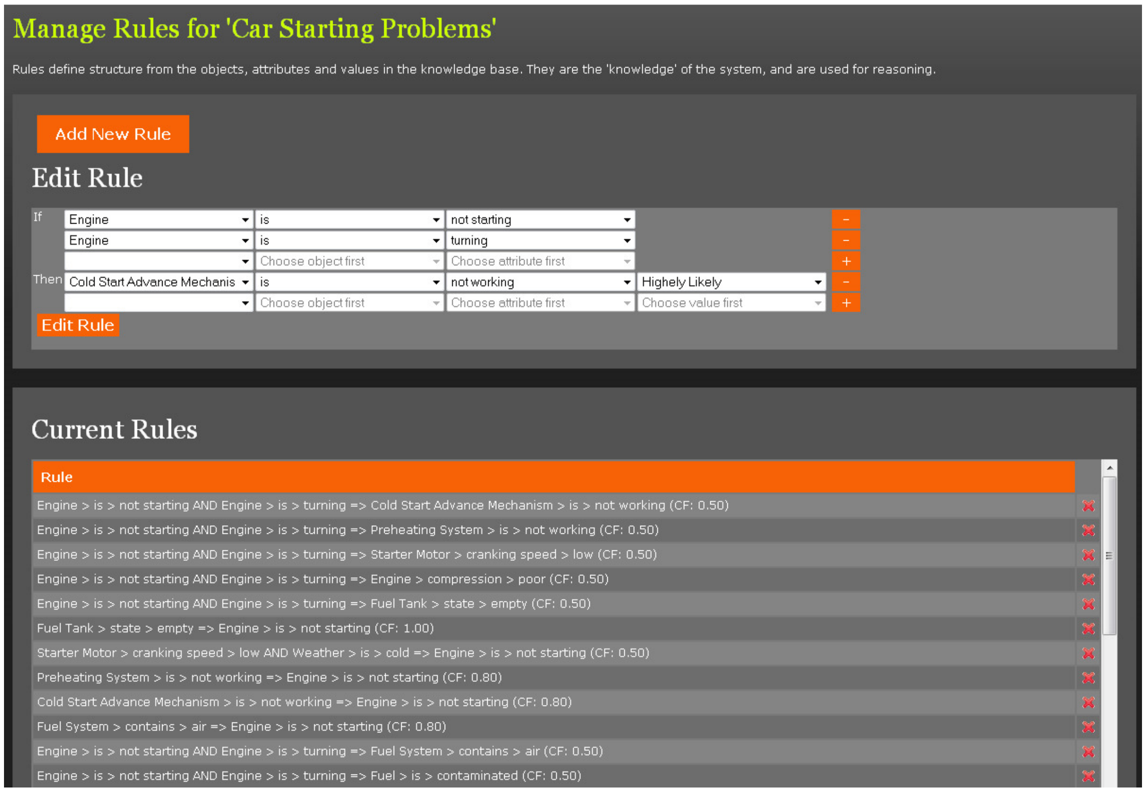
From this section of the system, the user can manage the rules of the knowledge base. The rules have to be defined on the objects, attributes and values defined in the earlier mentioned section, and so it makes sense that first one has to split the domain of the knowledge base as a list of objects, attributes and values.

This sections shows all existing rules, and the user can select exiting rules to edit them. The user also has the option to add new rules, and a rule can be made up of any number of antecedents, followed by any number of consequents.

All antecedents are ANDed together. The rule is in the form of:

IF {Antecedents} THEN {Consequents}

Each antecedent/consequent is selected by selecting the object, attribute and value related to the condition being modelled. For consequents, one must also enter the *certainty factor* if available, which can range from Yes to No, and possibly even allowing uncertain values like 'Almost certain', 'Unlikely', etc. Figure C-12 shows this section.



Manage Rules for 'Car Starting Problems'

Rules define structure from the objects, attributes and values in the knowledge base. They are the 'knowledge' of the system, and are used for reasoning.

Edit Rule

If	Engine	is	not starting	
	Engine	is	turning	
		Choose object first	Choose attribute first	
Then	Cold Start Advance Mechanis	is	not working	Highely Likely
		Choose object first	Choose attribute first	Choose value first

Current Rules

Rule
Engine > is > not starting AND Engine > is > turning => Cold Start Advance Mechanism > is > not working (CF: 0.50)
Engine > is > not starting AND Engine > is > turning => Preheating System > is > not working (CF: 0.50)
Engine > is > not starting AND Engine > is > turning => Starter Motor > cranking speed > low (CF: 0.50)
Engine > is > not starting AND Engine > is > turning => Engine > compression > poor (CF: 0.50)
Engine > is > not starting AND Engine > is > turning => Fuel Tank > state > empty (CF: 0.50)
Fuel Tank > state > empty => Engine > is > not starting (CF: 1.00)
Starter Motor > cranking speed > low AND Weather > is > cold => Engine > is > not starting (CF: 0.50)
Preheating System > is > not working => Engine > is > not starting (CF: 0.80)
Cold Start Advance Mechanism > is > not working => Engine > is > not starting (CF: 0.80)
Fuel System > contains > air => Engine > is > not starting (CF: 0.80)
Engine > is > not starting AND Engine > is > turning => Fuel System > contains > air (CF: 0.50)
Engine > is > not starting AND Engine > is > turning => Fuel > is > contaminated (CF: 0.50)

Figure C-12: Screenshot showing the rule management section