'What if?', 'Are You Sure?' and 'Why?' over Default Reasoning Systems

Johan Winberg

February 6, 2003

Contents

Abstract

Default reasoning systems have practical potential in diagnosis problems. For example, a system may hypothesize that a patient has a certain disease, based on a set of symptoms, test results, and the patient's medical history. Moreover, as additional facts become known, the system's hypothesis may change.

We developed a web-based interface that enables 'what if?', 'why?', and 'are you sure?' type querying over default reasoning systems. 'What if?' enables the quick specification of a case and the observation of its effect on all output variables. 'Why?' is a deeper capability that employs search to identify the relevant facts that lead to a given output. Finally 'are you sure?' identifies any unknown facts that, if specified, would reverse a given output. 'Are you sure?' and 'Why' require sophisticated search algorithms.

We evaluate our algorithms using the default reasoning system Z-log over a series of test knowledge-bases.

Chapter 1

Introduction

Self-care and self-diagnosis are important these days. Imagine that a person were to have a set of symptoms and is worried about what his condition might be. Going to the doctor on such an occasion could both be costly and time consuming. An easier way to ease his problems would be to have a system that allowed him to state what symptoms he has and what other factors that applies to his case and get a diagnosis. After getting a diagnosis the following question would be if the diagnosis is really correct or if it could be a less worrying condition or possibly a more serious condition? The system could then list any symptoms that would change the diagnosis. After getting this information the person could check if he has got any of these symptoms and decide if his condition would require medical treatment or if it is just a minor disease that will not require medical attention. This way he could possibly lessen his worries and save both time and money. Worried parents are also a group that could have use for this type of system. Suppose that they have a six month old baby that has a fever and a cough with certain properties, are these symptoms so serious that they should bring their child to the hospital or is it just a minor condition that could be treated at home? In the case that the user suffers from hypochondriasis it would also save hospital resources. Patients who think they need medical attention when they are in fact perfectly healthy takes up time for doctors that they could have spent treating patients who really need it.

Another example of a diagnosis type problem is when a car is malfunctioning.

Suppose that a car is making a knocking sound and begins to consume more fuel than it normally does. For some make and model this might be just a minor problem while for some other make and model this could be a sign of a serious problem that might cause a total engine failure. Ignoring the problem and just keep using the car could cause more severe damage to it. On the other hand bringing the car to a car repair shop when there is just a minor problem that the owner easily can take care of himself is an unnecessary expense. A reasoning system taking characteristics and behavior of the car as input and giving the possible causes for the behavior along with suggested actions to be taken could solve this problem. The car owner would only have to bring his car to the car repair shop when it was really needed.

Computer hardware interacting with software is an area that sometimes requires a great deal of expertise if you want to find sources of various errors that can occur. However if you know what the error is, almost anyone that can use a mouse and a keyboard can correct the error. If for example a printer is malfunctioning a user could just fill in a form in a diagnosis system stating the printers behavior and what software and hardware he is using. Then the system could give a list of probable causes and how to correct the malfunction in each case. Suppose that the printer is only printing black pages, the system could then tell the user that the most probable cause is in the printers configuration, however if the black pages occurred when printing from a certain application the problem occurs because of an option that has been selected in the application.

This type of capability is not currently available. Nor shall it be in the short term. However this thesis moves toward providing such capabilities.

This thesis considers some interface issues that need to be addressed for such diagnosis problems. The goal is to develop methods that will help to determine the quality of the results produced by a system. We use *Z-log*[12] as inference engine when we implement our user interface and analysis functions.

1.1 Organization of this Thesis

In chapter two we review the background of expert systems, default reasoning and explanation. In chapter three we describe the design and implementation of a graphical interface for *Z-log* incorporating 'what if?' querying, 'are you sure?' and 'why?'. 'Are you sure?' is also mentioned as sensitivity analysis and 'why?' is also mentioned as relevance analysis. In chapter four we evaluate this interface over some example knowledge bases. In chapters five and six we talk about some problems and performance issues with this system.

Chapter 2

Background

2.1 Expert Systems

A reasoning system is a system that can provide answers to questions posed by a user. The first efforts to construct such systems aimed to be complete problem solvers that could use general methods to solve general classes of problems. This approach did not however give many results that were of real use. It turned out that a better way to construct a reasoning system was to specialize in one specific area of knowledge[6] which is small enough for the system to be able of knowing virtually everything within the area.

An expert system consists of two main parts, an inference engine and a knowledge base (Fig. 2.1). The inference engine does not have to be specific for the area of knowledge that the system is used for. Its only purpose is to do reasoning over the data that is sent to it using a stored knowledge base. The inputs the inference engine gets is a knowledge base and a set of facts that specify the problem to be solved.

The knowledge base consists of rules and facts that model the way an expert within the area reasons. A good knowledge base is very important if the system is to be of any use. A good inference engine without a good knowledge base would be similar to a compiler without any programs.

Usually the knowledge base is built in collaboration with a few experts that describe to the knowledge engineer how they reason in various kinds of problems. The

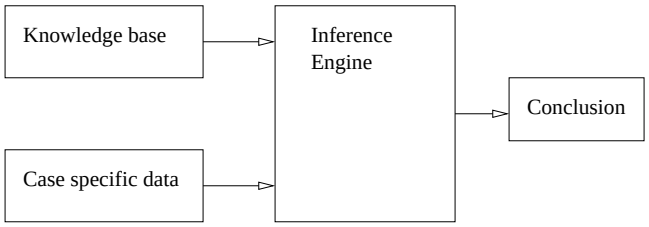

Figure 2.1: An overview of a typical expert system.

knowledge engineer then tries to put together general rules from the examples that the expert has made.

An early knowledge-based expert system was DENDRAL[4] that was used to identify molecular structures from mass spectrometry data. The naive solution for this problem would have been to check all the combinations until a matching structure was found. This would have resulted in a very slow system. Instead of doing this a large number of special rules for identifying common structures were used which reduced the number of combinations that needed to be checked significantly.

MYCIN[17] is an example of an existing medical diagnosis system that is designed to identify various blood infections. MYCIN has no model of the problem in the inference engine, the knowledge base and the inference engine are separate parts of the system. Given a set of symptoms MYCIN will give a diagnosis and suggest a treatment, it can also explain how it reasoned to make the conclusion it did. MYCIN decides how certain a diagnosis is with a truth-functional system called certainty factors.

MYCIN have never been used in practice by medical doctors but has influenced other expert systems. The reason why it has not been used in practice is not that it does not work, in fact it could perform as well as some of the experts that had helped to develop it, but the problem with who is responsible for the diagnosis that it makes. A system that was similar to MYCIN was R1[11] which was used to help configure orders for computer systems. R1 consists of thousands of rules and was used for the DEC computer corporation and was estimated to save millions of dollars per year.

MYCIN used certainty factors, however this is a weak probabilistic model, and

runs into serious difficulties if both diagnostic and causal patterns of inference are required together. Pearl [14] and others developed the Bayesian network approach which essentially is a compact representation of the full joint probability distribution of the variables. Bayesian networks may be specified using far fewer parameters than a naive approach. The system Pathfinder[5] represents a very impressive expert system that diagnosed rare lymph node diseases.

Though Bayesian networks are very accurate, they do require significant engineering costs in developing both the causal model of a domain, and, especially difficult, obtaining the parameters for the domain. Often one would like to express the rules of a domain as a set of typicality statements that do not require numerical parameters. This leads into the area of default reasoning systems.

2.2 Default Reasoning

When trying to model how humans reason there is a need for a mechanism that can copy our ability to know properties of things without having to be told about every specific detail. When we get told about an item we assume that it has certain properties depending on what it is. This is just because items of that type usually have those properties. For example birds usually have beaks and wings and can fly, so if we are told that something is a bird we will most likely assume that it has those properties. This is the case even though we have not got any specific information about it. Additional information about the object can also make us change our minds, for example if we get told that the bird is a sculpture we would not think that it could fly anymore.

Default reasoning is designed to meet such demands. The term *default* comes from the fact that if we have no specific information that can help us answer the question, we use the *default rules* for the world, the information that tells us how things *typically* work. Default reasoning is said to be nonmonotonic. This means that the set of possible conclusions can grow and shrink as new facts become known. In monotonic logic conclusions that has already been made can not be retracted and new information can never override information that has been deduced previously.

This leads to that the set of possible conclusions will only grow bigger when new facts are revealed or new rules are added.

There are two general approaches to default reasoning, the logistic approach and the probabilistic approach.

In the logistic approach the world is represented by a set of hard facts and a set of default rules. The hard facts are things that will always be true in the world, these facts can not be overridden or changed. Things that are not certain and can change are left for the default rules to decide. The default rules work in what is called a procedural way. With the procedural approach a rule like "if A then B" means that if A is true then B will get the value true and that value will be permanent. If several default rules assign different values to the same property the order in which the default rules are applied will decide the result. This poses a problem since it is hard to make a working priority system that will cause the default rules to be applied in an order such that correct results are produced. A working priority system is also likely to get big and unpractical.

In the probabilistic approach decisions are made by selecting the conclusion that has the highest probability. Rules in the probabilistic approach work in the empirical way. The empirical interpretation of the rule "if A then B" is that any world where A is true and B is false is impossible. In a probabilistic default reasoning system a world where A is true and B is false is not seen as totally impossible but it is seen as very unlikely. Each hard rule in the knowledge base form a possible world. These worlds will all be more or less probable and can be assigned probability weights. The default rules will work as constraints on these weights and can increase or decrease the probabilities of all the possible worlds. This is often represented with setting the probabilities to a number very close to 0 for false or very close to 1 for true. Since it represents that the conclusion is highly probable and not certain the value will not be set to exactly 0 or 1, it is usually represented with ϵ and $1 - \epsilon$ where ϵ is a very small number approaching 0[1]. A big advantage of the probabilistic approach is that even if there is not enough information to get an answer by the use of logic rules the probabilistic system can still say what is the most probable result. Examples of semantics for probabilistic reasoning are system-Z[15], conditional entailment[7],

maximum entropy entailment[8][3] and lexicographical entailment[10][2].

2.3 Explanation

From early on it was recognized that it is important that users be able to obtain explanations of expert system results[17]. Clearly this is true in the medical domain where one would never accept a diagnosis without justification since it could be a matter of life and death. Thus it is important that an expert system can explain why it makes the decisions it does and give some form of justification for its beliefs. Such a justification can be the set of rules and facts that caused the system to make the decision it did, this is called an explanation.

The following three definitions is how we view an [12] explanation in the context of default reasoning. φ represents the hard rules, Δ represents the set of default rules, L represents the propositional languages used and ϕ represents the conclusion.

Definition 1 *The triple* (φ', Δ', L') *is an* **explanation** *of* $\varphi \vdash_z \phi$ *under* L *iff* $\Delta' \subseteq \Delta$, $\varphi \models \varphi'$ *and* $\varphi' \vdash_z \phi$ *in the knowledge base* Δ' *under* L'.

This means that a subset of the knowledge base that gives the same value for the query as the initial knowledge base forms an explanation for the result. Under this definition the whole knowledge base is an explanation. This is not such an interesting explanation to get since it is quite obvious and do not really tell us about the specific details that lead to the concluded result. What we want is a minimal explanation.

Definition 2 *An explanation* $e = (\varphi', \Delta', L')$ *of* $\varphi \vdash_z \phi$ *is* **minimal** *if* $\neg\exists e' = (\varphi'', \Delta'', L'')$ *such that* e' *is an explanation of* $\varphi \vdash_z \phi$ *and* $\varphi' \models \varphi'' \wedge \Delta'' \subseteq \Delta' \wedge L'' \subseteq L' \wedge e \neq e'$.

When this definition is applied the explanations produced will not contain any facts or rules that are irrelevant to the concluded result. A loose way of explaining the definition would be to say that there can be no subsets of an explanation that are also explanations. Just having a minimal explanation is however not enough in the default setting. We might end up answering a different question than the one that was

actually asked and just give an answer that would produce the same result. Therefore the following definition is needed.

Definition 3 *An explanation* $e = (\varphi', \Delta', L')$ *of* $\varphi \vdash_z \phi$ *is in the* **context** *of* φ, Δ *and* L *if* $\forall\varphi''\forall\Delta''\forall L''(\varphi \models \varphi'' \models \varphi' \wedge \Delta' \subseteq \Delta'' \subseteq \Delta \wedge L' \subseteq L'' \subseteq L) \Rightarrow (\varphi'', \Delta'', L'')$ *is an explanation of* $\varphi \vdash_z \phi$.

For a contextual explanation there exists no supersets that are not explanations. If there exists such a superset the explanation is noncontextual. This definition ensures that any relevant information in the original facts will also be in the explanation.

Chapter 3

Approach

3.1 Black-box Analysis

Analysis over a reasoning system can be done in two main ways, *white-box analysis* and *black-box analysis*. *White-box analysis* means that the analysis mechanisms can see everything that is happening inside the reasoning system and takes advantage of how the reasoning system works inside. This can be used to optimize the analysis methods for the specific system. This however makes the analysis methods very specific to the system that is currently used and how it is implemented, changes in the implementation can make the analysis methods useless. *Black-box analysis* (Fig. 3.1) on the other hand does not consider how the reasoning system works inside. All it needs to know is what input it needs to give the reasoning system and what possible output the reasoning system will give. This makes the analysis methods independent of how the reasoning system is implemented and it can also work with different reasoning systems. For someone who is implementing an inference engine it can also be nice to have a simple way of incorporating analysis methods in their engine. Having a set of black-box analysis methods implemented as a java interface provides such a mechanism which could give an inference engine access to analysis methods with just a few lines of code. The downside is that no advantages can be taken of how the reasoning system is implemented, any system dependent optimization have to be left out.

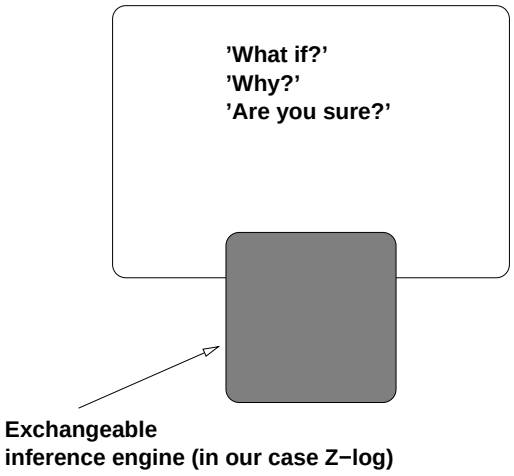

Figure 3.1: A diagram over the basic structure of the black-box design.

We have used *black-box analysis* to do sensitivity analysis and relevance analysis. In the implementation of the algorithms we use the default reasoning system *Z-log*[13] and its reasoning engine *ZEngine*. The implementation is however not depending on the use of *Z-log*, the methods could easily be moved to a different reasoning system and be used there.

3.2 'What if'

A simple way of using the query-engine is to set the truth values of a set of inputs variables and watch the effect on the output variable. The query-engine will then answer what truth-value the output gets for the given facts. Using this simple way of querying can make it hard to explore how changes in the truth-values of the facts in the knowledge base affects the value of a query. It might also be interesting to see the values of several queries at once, not just one at a time. 'What if' querying is a technique that allows adding facts and changing the values of facts in the knowledge base and watching what effect the changes has on a set of output variables immediately. This is useful for example in diagnosis systems where the user might want to try what possible output variables different input variables can be a sign of and what

might happen if additional input variables are added.

To achieve this effect, a graphical interface is used. The user declares the propositions that should work as input and the propositions that should work as output in the knowledge base. The program then builds a set of colored radio-buttons for the inputs and a truth-table for the outputs. The values that can be selected is *true*, *false* and *undecided*. Leaving an input as undecided has the same effect as not mentioning it in the set of facts when the simple query method is used. The output facts take on the same set of values, *true*, *false* and *undecided*.

This is a simple example that shows how the interface works. It also shows one of the basic properties of default reasoning, the ability to make a conclusion and then retract and change it when further information is given. The question that we want an answer for is if Tweety, who is a penguin, can fly.
Our knowledge base consists of these three rules:

- Birds can fly.
- Penguins are birds.
- Penguins can not fly.

We start with giving the system the most general information we have about Tweety, the fact that he is a bird (Fig. 3.2). The system answers that Tweety can fly but we want to be certain so we ask for a sensitivity analysis for the value of the fact flies. Now we get a suggestion for a case that would change the conclusion. The suggestion is that if the input proposition penguin is true then the output proposition flies will be false. Since we know that Tweety is a penguin we can give that information to the system and get the new conclusion that Tweety can not fly. Finally we can make a relevance analysis (Fig. 3.3) for the value of the fact flies and see that the reason why Tweety can not fly is that he is a penguin.

The underlying mechanism of this interface is a repeated use of the simple query method. For each declared output fact a query will be run with the selected input facts. When one query for each output has been run the outputs will be displayed in a truth-table. This process is made each time the user changes a value in the input

Figure 3.2: The first step in deciding if Tweety can fly.

Figure 3.3: Checking what facts that are relevant for knowing that Tweety can not fly .

facts. Even though it is actually several queries it looks like it is only one query to the user. The user can select values for the inputs by selecting true, false or undecided on the radio-buttons and the results will be updated instantly in the truth-table.

The graphical interface uses colors to make it clearer what value the facts have at the moment. Green for true, red for false and a pale yellow for undecided. If the user tries to select a value that would contradict with a hard rule the interface will ignore the selection and stay in its present state, it will also display an error message.

3.3 'Are you sure?'

A simple answer of just *true*, *false* or *undecided* is often not adequate. One additional thing that is good to know about a conclusion is how certain the system is that the conclusion is correct. An analysis of how certain a conclusion is is called *sensitivity analysis*. The goal of the sensitivity analysis is to find any combination of input variables that can alter the conclusion.

Sensitivity analysis is needed to get users to trust the conclusions the system makes. If no combinations that can alter the conclusion can be found or if the only combinations that can alter the conclusion is unlikely to happen, it is likely that the conclusion is correct. It might also turn out that the conclusion is not that certain. In that case the user can fill in values for the undecided facts that can alter the conclusion and do another sensitivity analysis. This can be used to gradually step towards a certain conclusion.

The number of combinations that needs to be checked in the worst case is $O(3^n)$ where n is the number of declared facts that are set to undecided. That is all the combinations of true, false and undecided for all the declared input facts that are set to undecided. This happens if there are no combinations of undecided facts that can change the conclusion. The search is conducted in a way such that the combinations with one undecided fact set to true or false is tested first, then the combinations with two facts and so forth up to the combinations with n facts. A superset of a combination that has already been reported is rarely interesting so any combinations that are supersets of combinations that has already been reported will not be reported.

A complexity of $O(3^n)$ means that any algorithm that uses memory in direct proportion against the complexity will run out of memory a long time before the time complexity gets too big to handle[16]. The solution to this is to use some form of depth-first search. Depth-first search only needs to keep the path of nodes from the root to the goal-node in memory at the same time.

We also want the combinations to turn up in the order mentioned earlier, with the smallest combinations first. This is achieved by using iterative deepening search[9], that is we do depth-first search to depth one, then depth-first search to depth two, and continue in this way for each depth up to depth n.

The children of each node are generated by adding one fact to the current combination. The facts added are the undecided facts that are not present in any of the parent nodes up to the root node. One child will be added for true and one for false for each fact. This means that the root node will have $n * 2$ children, the next level will have $(n - 1) * 2$ possible children to each node down to the last level that will only have 2 children.

We want to have the possibility to find all combinations of input facts that can alter the conclusion. Therefore the path from the root-node to the current node is stored so that we can resume the search later without having to check any combinations of input facts that we have already checked.

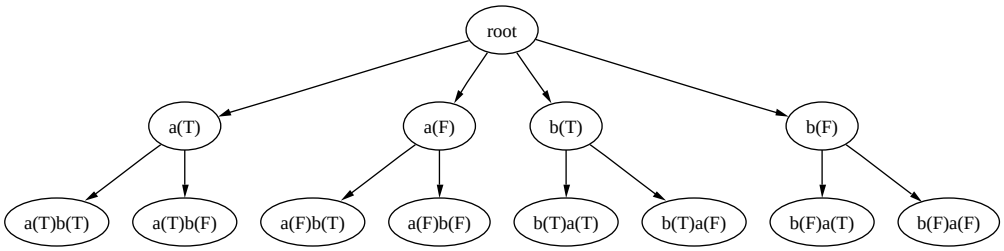

Figure 3.4: This is the full search-tree with two facts, a and b, F and T stand for true and false.

This algorithm generates all the combinations of true, false and undecided (Fig. 3.4) for all the declared facts that are set to undecided *with respect to order*. We do not care about order for our analysis and this generates a lot of extra nodes. In

fact we get $2^n * n!$ nodes instead of $3^n - 1$. Luckily there is a way of avoiding this in the sensitivity analysis. If we number the children of every node from left to right starting at zero, studying the appearance of a fully spanned tree will show that if a nodes number is n, then all children of that node (Fig. 3.5) with numbers up to $(n \; div \; 2) * 2 - 1$ will be combinations that we have already found.

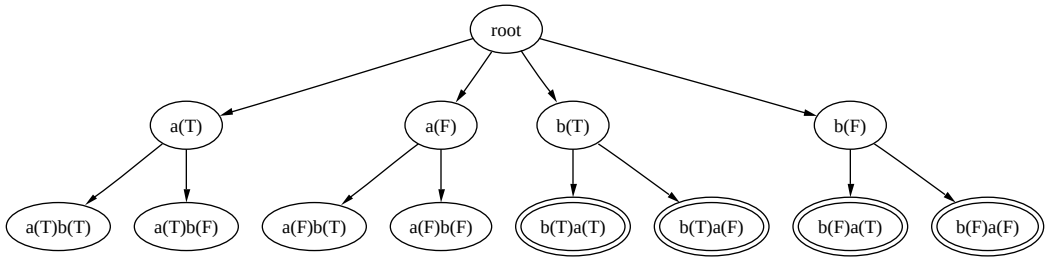

Figure 3.5: This is the same search-tree as in the previous picture with the nodes that represent repeated states marked with double circles.

The result of the sensitivity analysis is presented to the user in two ways (Fig. 3.6). In the input area the undecided facts that can change the conclusion will get their border colors changed, green if the fact should be true to get a changed conclusion, red if the fact should be false to get a changed conclusion. There is also a textual explanation saying what facts that should be changed and to what values they should be changed and finally what the conclusion changes to. The next combination that can change the conclusion (if any exists) can be found by pushing the *next* button that appears after the explaining text.

An example of the basic way to use the sensitivity analysis was shown in the description of the 'what if?' interface. We first stated that Tweety is a bird and from that we could conclude that he could fly (Fig. 3.2). Then we did sensitivity analysis and discovered that if he was specific type of bird called penguin he would not be able of flying. Since Tweety is in fact a penguin we could conclude that he can not fly.

Figure 3.6: An example of how the result of the sensitivity analysis is presented.

3.4 'Why'

As mentioned earlier a simple answer of true, false or undecided might not be that useful by itself. Some form of justification from the system stating what made it draw the conclusion it did is needed. If the system can show how the facts that made the system draw a conclusion we identified thus enhances the trust users can put to the system. This is very important if the system is going to help users draw conclusions. If the users do not trust the conclusions given by the system and check for answers by themselves the system loses much of its purpose.

The part of the knowledge base that leads to the conclusion is called an *explanation* of the conclusion. The entire knowledge base is an *explanation* of the conclusion. This is of course not a very interesting thing for the system to tell the user. What we want is a *minimal explanation*. A *minimal explanation* is an *explanation* that you can not remove any facts or rules from and still get the same conclusion. We also want the *explanation* to be in the *context* of the original knowledge base. To be in *context* means that it is not enough that the *explanation* gives the same conclusion, it must also contain the information that originally led to the conclusion. These concepts we defined in section 2.3.

The explanation can be divided into two parts, finding the relevant facts and finding the relevant rules. What we do with our relevance analysis is finding the relevant facts. Finding the relevant rules could be done using the same algorithm as we use to find the relevant facts, this is however not implemented in the current system.

The relevance analysis is divided into three parts. First we find a possible minimal explanation, then we do a check to see if the explanation is contextual, if it is we are done, otherwise we have to backtrack and start over with the first step.

Searching for a possible minimal explanation is done by removing combinations of facts until we have a combination of facts that we can not remove any more facts from and still get the same conclusion. In the worst case we have to check $2^n - 1$ combinations to get all the possible minimal explanations where n is the number of declared facts that are set to either true or false.

Even though this search generates less combinations than the sensitivity analysis we still have a complexity of $O(2^n)$. This means that just like with the sensitivity analysis we can not use a memory bound technique. If we did use a memory bound technique the space complexity would become too big to handle a long time before the time complexity gets too big for users to accept. We have selected to use depth-first search which is a technique that is not memory bound.

So far we see big similarities with the sensitivity analysis, there are however a few differences that will make the search work in a very different way. First of all we want a minimal explanation, not an explanation with a specific number of facts. So iterative deepening makes no sense to use here, we will simply use depth first search.

The children of each node are generated by removing one fact from the set of facts in the parent node. In the root node we have the initial set of facts that lead to the conclusion we want an explanation for. So the root node will have n children and the next level will have $n - 1$ children down to the last level that will have 1 child. This means that the fully spanned search tree (Fig. 3.7) will have $\sum_{i=1}^{n} i!$ nodes, that is all the combinations *with respect to order*. A node is a possible minimal explanation if none of its possible child nodes are contextual explanations. This means that we have to expand the child nodes before we know if we have a possible minimal

explanation. This in turn means that we can not use the way of avoiding repeated states that we could in the sensitivity analysis. The number of possible states is also too big to keep track of in memory so in the worst case we will actually have to go through all the $\sum_{i=1}^{n} i!$ combinations. However, we will always find the first possible minimal explanation in at most n steps, it is only if this explanation turns out to be non-contextual that we have to continue the search.

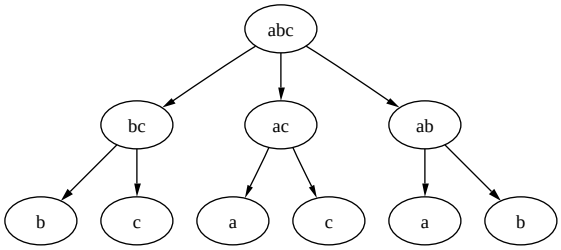

Figure 3.7: The search-tree for the initial search for a minimal explanation in the relevance report with the facts a, b and c.

We can not store all possible combinations to check for repeated states but the number of possible unique minimal explanations is a much smaller number. These combinations can be stored in memory when they are found together with the result of the context check for that combination. This has two advantages, firstly the combinations reported to the user will not be repeated states and secondly if we have done context check for a combination we do not have to do that again.

The context check is very similar to the sensitivity analysis. It can be viewed as a sensitivity analysis with the possible minimal explanation as the decided facts and the facts that are set to true or false in the knowledge base but are not present in the possible minimal explanation as undecided facts. The only difference is that the values for the "undecided" part will not be turned to anything else than what they actually are in the knowledge base.

Context checking is done by testing if there is any combination of facts from the original knowledge base that can be added to the possible minimal explanation that we have found that would change the conclusion. If there exists a combination like that the possible minimal explanation is non-contextual.

The number of combinations that the context check needs to check in the worst case is $O(2^n)$ where n is the number of declared facts that are set to either true or false in the knowledge base and that are not present in the possible minimal explanation. Again we have a complexity that makes it impossible to use any algorithm that uses up memory. So we know that a form of depth first search is desirable. Supersets of a combination that will show that the possible minimal explanation is non-contextual is not interesting, we just want to find the first set that can show that the possible minimal explanation is non-contextual. We also want to find that as quickly as possible if it exists. So we want to use a search that produces the smaller sets first, for this reason we have chosen iterative deepening.

The children of each node is generated by adding one fact to the current combination. The facts added are the declared facts that are set to either true or false in the knowledge base and that are not present in the current node. So the root node will have n children and the bottom level nodes will have 2 children (the level below that is not interesting since that would contain the original set of facts) (Fig. 3.8).

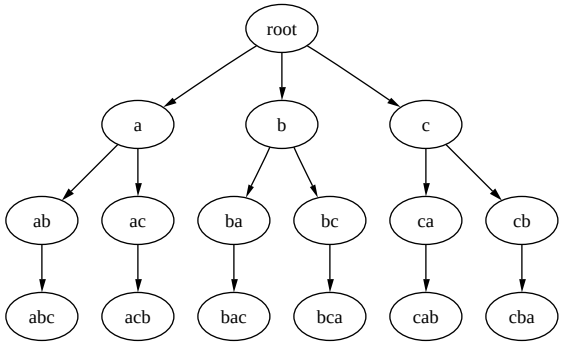

Figure 3.8: The search-tree for context checking with the facts a, b and c.

We can avoid repeated states in the context check in a similar way to how we avoid repeated states in the sensitivity analysis (Fig. 3.9). If we number the children of every node from left to right starting at zero studying the appearance of a fully spanned tree will show that if a nodes number is n, then the first n children of that node will be combinations that we have already found.

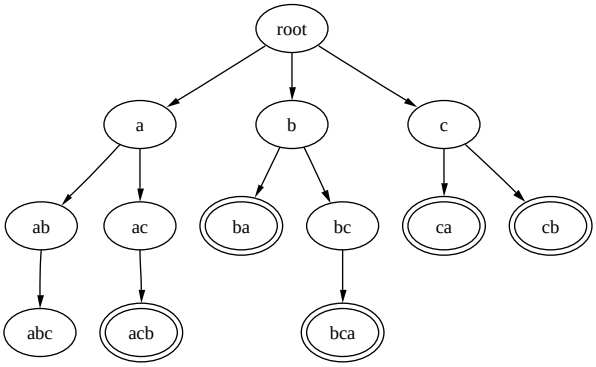

Figure 3.9: This is the same search-tree as in the previous picture with the nodes that represent repeated states marked with double circles.

Figure 3.10: An example of how the result of the relevance analysis is presented.

The result of the relevance analysis is reported with a textual explanation (Fig. 3.10) that shows the names of the facts that form a minimal contextual explanation.

3.4.1 Fast 'Why'

With a large number of facts the relevance analysis becomes very slow. This is expected since the complexity of the search is exponential. The main reason of the slow performance is the context check. If we remove the context check and only do the initial depth first search we will always find a minimal explanation in at most n steps, where n is the number of declared facts that are set to either true or false.

Removing the context check can of course give an explanation that is not in the context of the original set of facts, but it will give a minimal explanation. This can be used to find out what values that are needed to give a given value to a fact. It can also be used to see the difference between a contextual and a non-contextual explanation.

The fast relevance report uses the same algorithm as the relevance report but without the context check. The backtracking that the context check causes if it finds that the possible minimal explanation is non-contextual will of course never happen.

Chapter 4

Example Knowledge Bases

4.1 Simple Overrides

Section 2.2 points out that an explanation needs to be in the context of the problem that the system is giving an answer to. This is a simple example that demonstrates the difference between an explanation that is in the context of the original problem and one explanation that is not in the context of the original problem. Both of the explanations are minimal explanations to the problem. This demonstrates the systems capability of finding contextual explanations.
Our knowledge base consists of these three rules:

- Cheese is tasty.
- Dirty cheese is not tasty.
- If we are starving then dirty cheese is tasty.

We set the three input propositions cheese, dirty and starving to true and can see that we get the expected value TRUE for the output proposition tasty. When we do relevance analysis with context check (Fig. 4.1) we get the answer that the facts that are relevant for the conclusion is all of the three facts cheese, dirty and starving which is the only minimal contextual explanation for the conclusion.

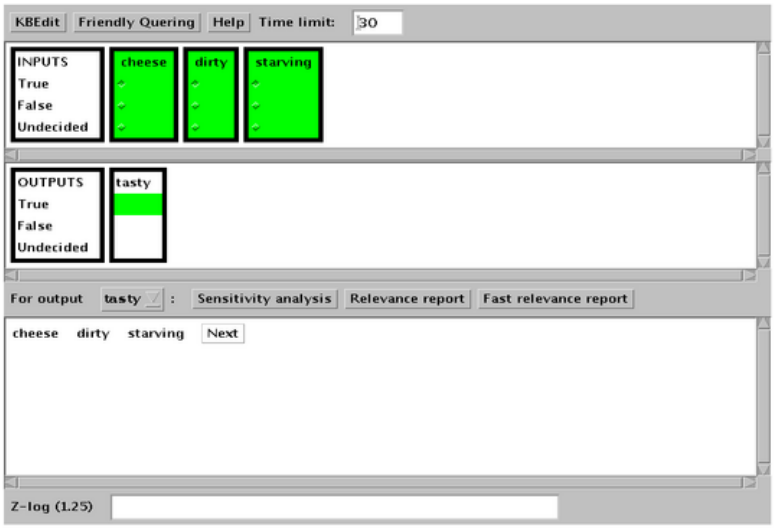

Figure 4.1: This shows the result of relevance analysis with context check.

Figure 4.2: This shows the result of relevance analysis without context check.

If we do the same query but use the fast relevance analysis (Fig. 4.2) which does not do a context check we get the answer that the only fact that is relevant for the conclusion is cheese. This is quite obviously a minimal explanation since there are no facts that can be removed from it and it gives the same conclusion as the initial set of facts (because of the first rule that cheese is tasty). However the explanation is not in the context of the original question since there is one fact (dirty) in the initial set of facts that can be added to the explanation that would lead to a different conclusion.

4.2 The Majority Function

The majority function is an important function because it often shows the worst case performance for a logical reasoning system. The function takes an arbitrary number of parameters and if more than half of the parameters are true it returns true. It is also interesting to see what explanations the system will give since it is not obvious what the answer will be. For our tests we use a majority function for the four facts a, b, c and d.
The rules needed for the knowledge base to implement the majority rule for these facts will look like this:

- Typically we do not have a majority.
- If any combination of three facts with all facts as true exists then we have a majority.

Here (Fig. 4.3) we have put all of the four facts as true and get the expected result true. When we ask for a relevance analysis the system answers that the relevant facts for the conclusion are b, c and d. If we push the next-button the system will continue to show the other combinations containing three facts (the lowest number of facts that will be a majority).

If we have only two of the facts as true we no longer have a majority. If we ask for a relevance analysis (Fig. 4.4) the system answers that the relevant fact for the conclusion is the fact true, this is the systems response if a general default rule have taken effect, in this case the rule saying that we typically do not have a majority.

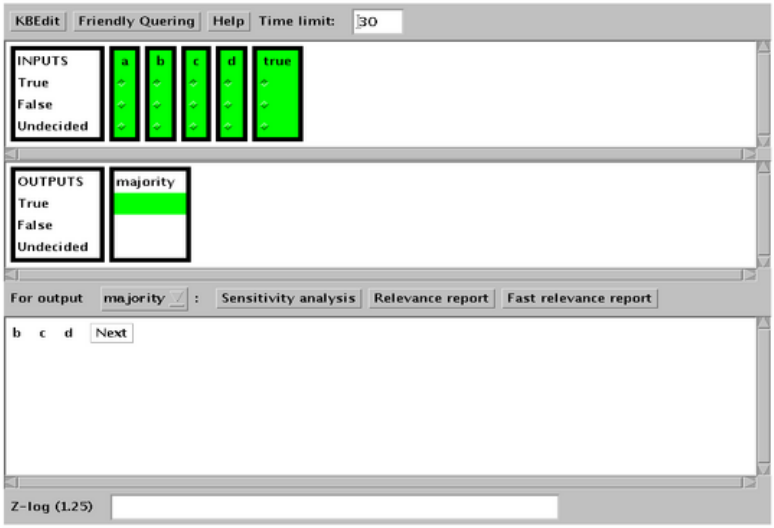

Figure 4.3: The result of relevance analysis for the majority function with more than a majority of the input facts set to true.

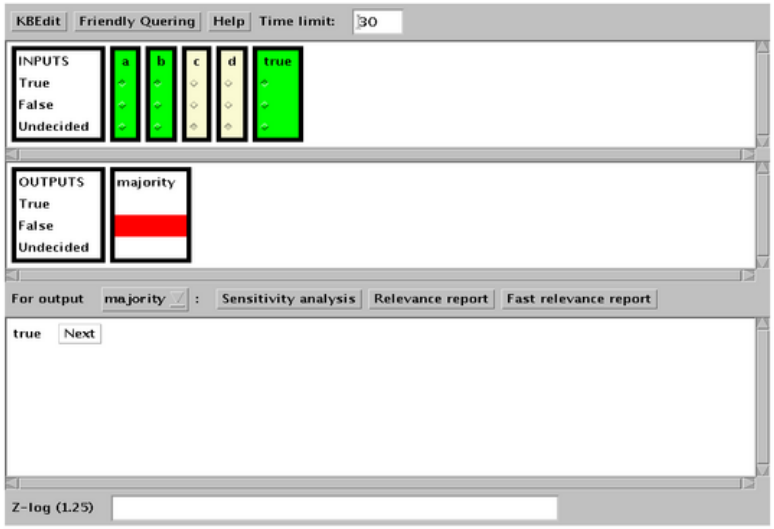

Figure 4.4: The result of relevance analysis for the majority function with less than a majority of the input facts set to true.

To make a knowledge base that implements the majority function we need $\binom{n}{n/2+1}$ rules where n is the number of declared input facts. In the worst case an explanation will be found in $n/2$ steps for such a knowledge base. For sensitivity analysis the worst case will take $n/2+1$ steps to solve.

4.3 The Parity Function

The parity function is just like the majority function a common function that is often used to test a systems worst case performance. The function takes an arbitrary number of parameters and returns true if an even number of parameters are true and false if an odd number of parameters are true. For our tests we use a parity function for the four facts a, b, c and d.
This is the knowledge base used to represent the parity function for these four facts:

- Typically we have parity.
- If only one of the facts are true we do not have parity.
- If any combination of two facts with both facts as true exists we have parity.
- If any combination of three facts with all facts as true exists we do not have parity.
- If any combination of four facts with all facts as true exists we have parity.

This (Fig. 4.5) shows the case where all the four facts are set to true, since we have an even number of facts we get the value true for parity. When we ask for a relevance analysis we get the answer that all the four facts are relevant for the conclusion which seems reasonable since a removal of any single fact would lead to a changed conclusion.

With three of the facts set to true we get the value false for parity. When we ask for relevance analysis (Fig. 4.6) we get the same answer as in the previous test, all the facts are relevant which makes sense for the same reason, removal of any single fact would lead to a changed conclusion.

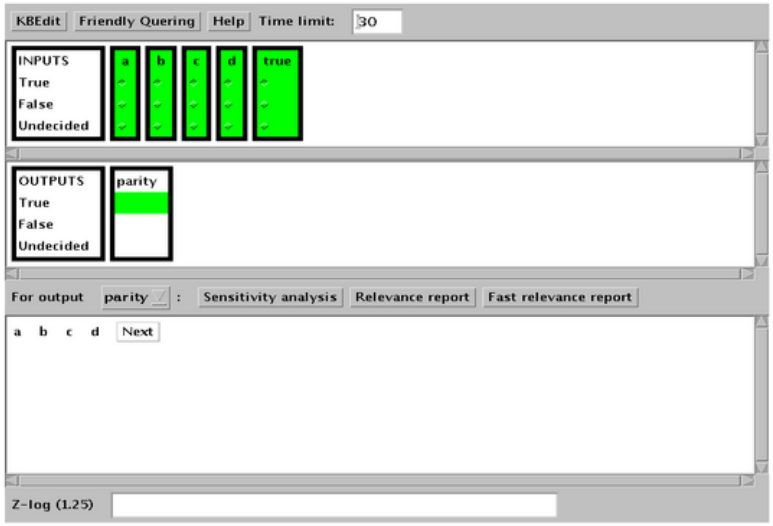

Figure 4.5: The result of relevance analysis for the parity function with four input facts set to true.

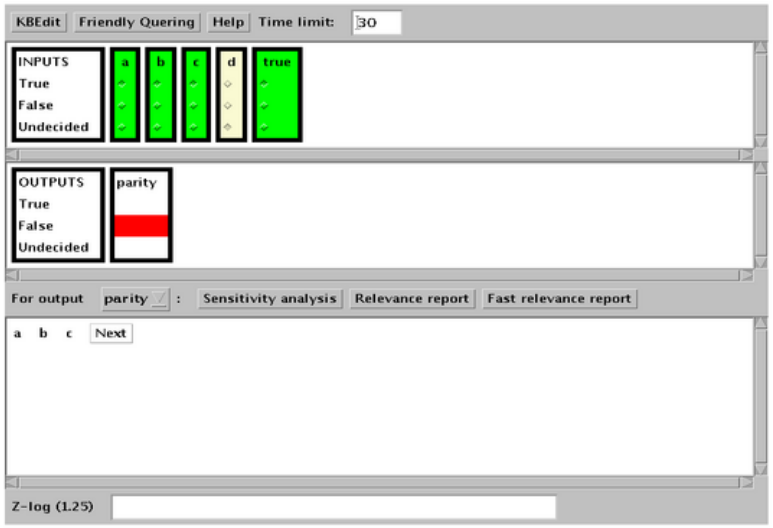

Figure 4.6: The result of relevance analysis for the parity function with three input facts set to true.

A knowledge base that implements the parity function needs $1 + \sum_{i=1}^{n} \binom{n}{i}$ rules, where n is the number of declared input facts. An explanation will always be found in $1 + k$ steps where k is the number of declared input facts that are set to false. The sensitivity analysis for the parity function will always find a result in one step since if there are declared input facts that do not have a value set, any of those facts changed to true will change the conclusion and if there are no declared input facts that are undecided the conclusion can not change.

4.4 Simple Medical Diagnosis

In the introduction we talked about a system that could be used for medical diagnosis. This is a very simple example showing the basic idea of how to use the *Z-log* system for diagnosis problems.

Figure 4.7: The first step in the medical diagnosis.

Here (Fig. 4.7) we can see the first step of the diagnosis. The user have submitted one symptom, he has a cough, and some basic information about himself, he is an adult male. The system responds with a possible diagnosis which is that the user has a cold. When the user asks for a sensitivity analysis it turns out that the diagnosis

is not certain, if he vomits it is not that likely that his disease is a cold.

Figure 4.8: The second step in the medical diagnosis.

Submitting the value true for vomit shows (Fig. 4.8) that the most likely diagnosis is that the user has the flu. Another sensitivity analysis for this case shows that if he is vomiting or coughing blood, flu would not be a likely diagnosis, however this is not the case for the user.

To make the diagnosis more certain the user selects the value false for blood which does not change the diagnosis. Another sensitivity analysis is made which shows (Fig. 4.9) that now there are no facts in the knowledge base that can change the conclusion.

Now when we have a certain diagnosis we can do a relevance analysis which shows (Fig. 4.10) that the relevant facts for the diagnosis of flu is cough and vomit.

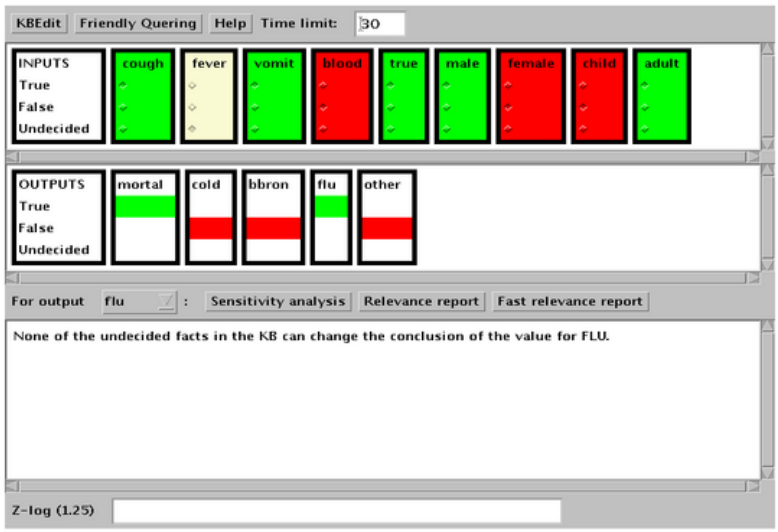

Figure 4.9: The third step in the medical diagnosis.

Figure 4.10: The fourth step in the medical diagnosis.

Chapter 5

Results

5.1 Performance Issues

With exponential or even factorial complexities there is of course going to be problems with performance. Since the complexity measure is a worst case scenario and not the average case it also means that a fairly small number of facts can take a long time to analyze while a big number of facts might just as well be analyzed quickly depending on the properties of the current knowledge base. Since the algorithms conserve memory, the space complexity will not become a problem unless the knowledge base is huge, the problem is the time complexity.

5.1.1 Time measurements

The time complexity of the sensitivity analysis and the relevance analysis is highly dependent on the structure of the knowledge base. This means that all time measurements are specific for that case and can not be seen as a function of the number of rules and facts in the knowledge base. However we can use a knowledge base that is likely to produce worst case behavior or close to worst case behavior. With such a knowledge base we can get and idea of at which point the time complexity might become unwieldy.

Sensitivity analysis

For sensitivity analysis we can test the worst case for a given number of facts with a pretty simple knowledge base. We need two rules:

- If a is true then x is true.
- If a is true and the rest of the facts in the knowledge base are false then x is false.

What we do is to set the fact a to true and set no values for the rest of the facts, then we ask for a sensitivity analysis. The search now goes through all of the possible combinations since the only combination that can change the conclusion is if all the undecided facts are false, this is the last combination that will be generated by the search. This graph (Fig. 5.1) shows that if we have more than ten undecided facts

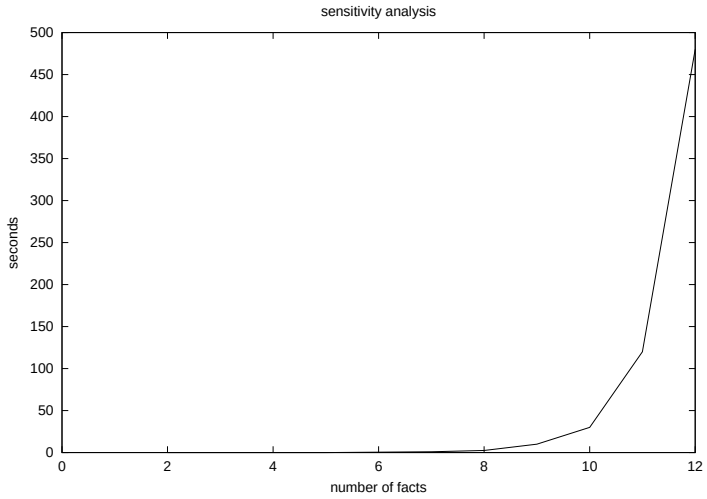

Figure 5.1:

there is a risk that the sensitivity analysis will become unwieldy.

Relevance analysis

The knowledge base needed to get the worst case for relevance analysis would be complicated to construct so the knowledge base used here is not the worst case, it

will however be close to worst case behavior.

- $a \rightarrow x$
- $a \wedge b \rightarrow \neg x$
- $a \wedge b \wedge c \rightarrow x$
- $a \wedge b \wedge c \wedge d \rightarrow \neg x$
- $\vdots$

For each fact we add one rule so that the result will alternate between true and false. The resulting knowledge base will have a nested structure that can give series of noncontextual explanations. As we can see in this graph (Fig. 5.2) the relevance

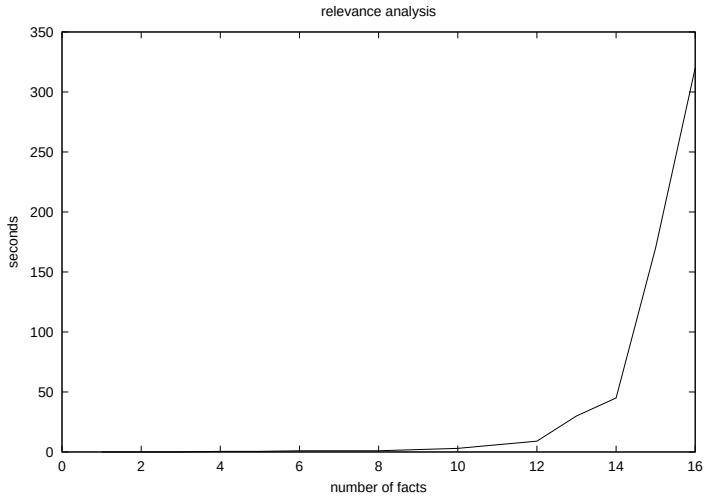

Figure 5.2:

analysis can handle about fourteen facts before it starts to get unwieldy.

Time Limit

One way of "solving" the problem with time complexity is with a time limit for the analysis. This approach is implemented in our system. The user can specify the

number of seconds that he thinks that it is acceptable to wait for an answer. If the calculation takes longer than the specified time the calculation is aborted and the user will get a message that the time limit was exceeded. The user can then either set a higher time limit until he gets an answer or accept that the question asked is too complex to be answered within a reasonable time.

Having a time limit is of course not a good solution. It is just a way of making the system easier to handle for the user. Instead of having to restart the system because a query is not finished within a reasonable time, the reasonable time can now be chosen by the user and the system can keep running.

Lexical Analysis

If a rule R in a knowledge base contains a fact f, f can be said to be lexicographically bound by R. It is likely that the facts in a knowledge base will not be bound by all the rules and that they can be divided into groups that are bound by the same group of rules and are unaffected by the rest of the knowledge base. This connection can be used to possibly improve the performance when handling a bigger knowledge base. Lexical analysis of the knowledge base can be done and the knowledge base can be partitioned. If the knowledge base is partitioned in this way the analysis methods can be used on the partitions instead of the entire knowledge base. This will give the effect of working on several smaller knowledge bases instead of one big knowledge base.

Performance of Inference Engine

The performance of the inference engine used by the analysis methods affects the systems performance to some degree. *Z-log* could be further optimized which would increase the performance of the whole system. It would also be possible to use some other inference engine that has better performance than *Z-log*. This would allow for analysis over slightly bigger knowledge bases although it would not be a complete solution to the performance issues of the analysis methods. If a full 'why' with rulebase minimization is to be incorporated a more optimized inference engine is also

desirable.

5.2 Interface Quality

The interface was designed to be simple to handle and understand. Both the input variables and the output variables are structured as truth tables and the colors *green*, *red* and *yellow* are used to represent the values *true*, *false* and *undecided* respectively. These are two commonly used analogies that most people understand. There are also texts that explain what values the colors and positions in the tables stand for so that users that are not familiar with these concepts can still understand the system.

There is no need to understand default reasoning or to have any previous experience with expert systems to use the interface. If a user would still fail to understand how to use the interface there is an explaining help text.

Chapter 6

Discussion and Future Work

6.1 Problems and Limitations

6.1.1 Realistically Sized Knowledge Bases

A real medical diagnosis system might have to reason over hundreds of rules and possible symptoms. As previously stated in the section about performance issues the analysis methods in their current form have no way of solving a problem like this within reasonable time. Therefore it is not realistic to say that the system in its current form can be used to solve any diagnosis problems in practice. We do not have any knowledge bases implementing realistic medical diagnosis at this point either.

6.1.2 Platform Independence

The *java* language which have been used to implement the analysis methods and the applet interface is meant to be platform independent. This is however not really the case. The GUI can look very different on different platforms and in different browsers. The applet should run under any system using an *java* 1.1 virtual machine or greater but the GUI might behave strangely (this is the case with *netscape* on unix type systems). To really get the intended behavior from the applet we recommend that the applet is run with *mozilla* on a UNIX type system using the *java* 1.4 virtual machine. Most of these problems can probably be solved by converting the applet to

use the swing API instead of the awt API.

Since the reasoning system is implemented as an applet it is hard for users to load or save their own knowledge bases in the system. The only way of doing this today is to edit the test knowledge bases we have provided with the system.

6.1.3 Dependence of Underlying System

A problem encountered during the implementation of the system was working with a system that depended on code that was written by someone else. It happened on more than one occasion that bugs that were suspected to originate from new code in fact originated from the inference engine. The inference engine was used in a way that was possible but not foreseen when the original system was constructed.

6.1.4 No Rulebase Minimization for Explanation

The relevance analysis provided by the system does not give a full explanation of why it reasons the way it does. It only shows what facts that are relevant and gives no clues about what rules that are important for the conclusion.

6.2 Future Work

6.2.1 Full 'Why'

The explanation has two parts, finding a set of facts that are relevant to the conclusion and finding a set of rules that show why the system made the decision it did. After finding a set of relevant facts that form a minimal contextual explanation the same algorithm used to find those facts can be used on the rules.

Each set of facts that form a minimal contextual explanation can of course have different corresponding sets of rules that they form a full explanation with.

Even though the same algorithm may be used to find the relevant rules it will most likely demand a lot more coding to work together with *Z-log* than what was needed to find the relevant facts. Finding the rules will also be heavier computationally

because the knowledge base will have to be recompiled between every change in the rules. Implementing the full relevance analysis will most likely demand that *Z-log* is optimized or that another inference engine with better performance is used.

6.2.2 Java Interface Definition

One of the biggest benefits from the black-box approach is that the algorithms are very general. The only thing needed from the entailment engine is a query-method. With little effort the current analysis methods could be modified and put into a java interface definition so that any entailment engine could just implement the interface and get the sensitivity analysis and the relevance analysis.

6.2.3 Lexicographical Analysis

As mentioned in the section about performance issues the knowledge base can be partitioned and thereby making calculations less demanding. An automatic function to do this could be incorporated into the *Z-log* system so that analysis over bigger knowledge bases could be possible. A better interface for constructing the rules of the knowledge base could also be incorporated into this system. The only way to construct a knowledge base for the system today is to write down all the rules and facts manually.

6.2.4 Swing and AWT

The GUI API used for the applet is awt which is an old API with limited capabilities. Swing is a newer GUI API which is meant to replace awt and can supply more capabilities to the interface which can make it easier to use. The reason for using awt is that it is supported and supplied with default installations on most platforms which is not the case with swing. A future improved version of the applet should be converted to use swing.

6.2.5 Applet Issues

Applets are best suited for simple applications that are not intended to be used heavily. If the system grows and incorporates all of the features mentioned above it would probably be best if the system would be converted to a normal application. This could improve the performance somewhat and give the system access to functions that are not accessible by applets for security reasons like writing to files on the local computer.

Chapter 7

Conclusions

7.1 Importance of 'Expertise in a box'

In today's technical society, we are confronted with more and more need for expertise. However much of the expertise required is rather detail oriented. In areas like medical diagnosis, configuration of technical systems or finding sources of errors in technical systems, the expert is not using any of his creative thinking, he is just answering questions based on knowledge that he acquired earlier. In such situations the question could just as well have been answered by an expert system.

Experts often demand to be well payed for their services which means that a company might not afford the amount of experts that it would really need. Even if the needed amount of experts could be afforded it might not be possible to hire as many experts as desired simply because the amount of people that have the required knowledge do not exist. A bonus is that the expert system might be able of finding the answer to the problem quicker than the expert would have done.

Most areas of knowledge today are so large that it is almost impossible for one person to know everything within an area. So even someone who is considered to be an expert within his field of knowledge might need help from other experts with questions about knowledge related to his expertise. For example a medical doctor often has a general knowledge of medicine but only acquire more specific knowledge about a certain group of diseases or medical problems. If the patient happens to have

a condition that lies outside his specialization the only thing he can do is usually to send the patient to another doctor.

We bring more and more machines into our homes. When using these devices the user might encounter problems such as a complicated installation process or problems with performing a specific task with the device. Calling for a professional to solve the problem for you might be expensive and customer help-desks are often hard to reach. There is also a need for expert knowledge when you are purchasing a technical device, if you have no knowledge about the area you might get a machine that can not fulfill your requirements. It is also possible that you might get a machine that has lots of features that you do not really need and hence is more expensive.

We can envision systems that encode the rules of these various areas and help to save both time and money. Systems of this type may very well be necessary if we are going to expand the technological society. Human minds are a limited resource and using creative minds for tasks that could just as well have been solved by machines is a waste.

7.2 Summary of Contributions

This thesis has covered expert systems and default reasoning and explained the need for analysis methods that can tell the user how the system reasons over given facts. We have also described the implementation of an interface with incorporated analysis methods for the default reasoning system *Z-log*. The interface is implemented as a java applet and can handle 'what if?' type querying and has methods for sensitivity analysis and relevance analysis written with a black box approach. Using complexity measures we have shown that the system has some performance issues that makes the system unwieldy with larger knowledge bases. The system can however be a nice tool for demonstrating analysis over a default reasoning system with more moderately sized knowledge bases. Approaches for solving or making these performance issues less severe are also proposed. We have demonstrated how the system works and handles different situations using some simple examples that demonstrate possible areas of use or are important because they test the system in tricky situations.

7.3 Acknowledgments

I would like to thank my instructor Michael Minock for giving good advice both in the practical and theoretical parts of the work and for his assistance in the process of writing this thesis. I would also like to thank him and the Department of Computing Science at Umea University for letting me do my thesis work there.

Bibliography

[1] Ernest Wilcox Adams. *The Logic of Conditionals*. D. Reidel, Dordrecht, 1975.

[2] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment, 1993.

[3] R. Bourne and S. Parsons. Maximum entropy and variable strength defaults, 1999.

[4] B. G. Buchanan, G. L. Sutherland, and E. A. Feigenbaum. Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In Bernard Meltzer, Donald Michie, and Michael Swann, editors, *Machine Intelligence 4*. Edinburgh University Press, Edinburgh, Scotland, 1969.

[5] Heckerman D., Horvitz E. J., and Nathwani B. N. Toward normative expert systems: The pathfinder project., 1990.

[6] E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg. On generality and problem solving: A case study using the DENDRAL program. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 6*. Edinburgh University Press, Edinburgh, Scotland, 1971.

[7] H. Gefner. Default reasoning: causal and conditional theories, 1992.

[8] Moisés Goldszmidt, Paul Morris, and Judea Pearl. A maximum entropy approach to nonmonotonic reasoning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, 1990. AAAI Press.

[9] Richard E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 1985.

[10] Daniel J. Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15(1):61–82, 1995.

[11] J. McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19(1), 1982.

[12] Michael Minock and Hansi Kraus. Z-log: A system-z (and $\hat{Z}$) 'system'. Technical Report 02.05, The Univeristy of Umea, Umea, Sweden, May 2002.

[13] Michael Minock and Hansi Kraus. Z-log: Applying system-z. In *Proceedings of the Joint European Conference on Logic in Artificial Intelligence (JELIA): Systems Session*, Consenza,Italy, 2002. Springer-Verlag.

[14] J Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA., 1988.

[15] J. Pearl. A natural ordering of defaults with tractable applications to default reasoning. *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge, pages 121–135.*, 1990.

[16] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[17] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier/North-Holland, Amsterdam, London, New York, 1976.