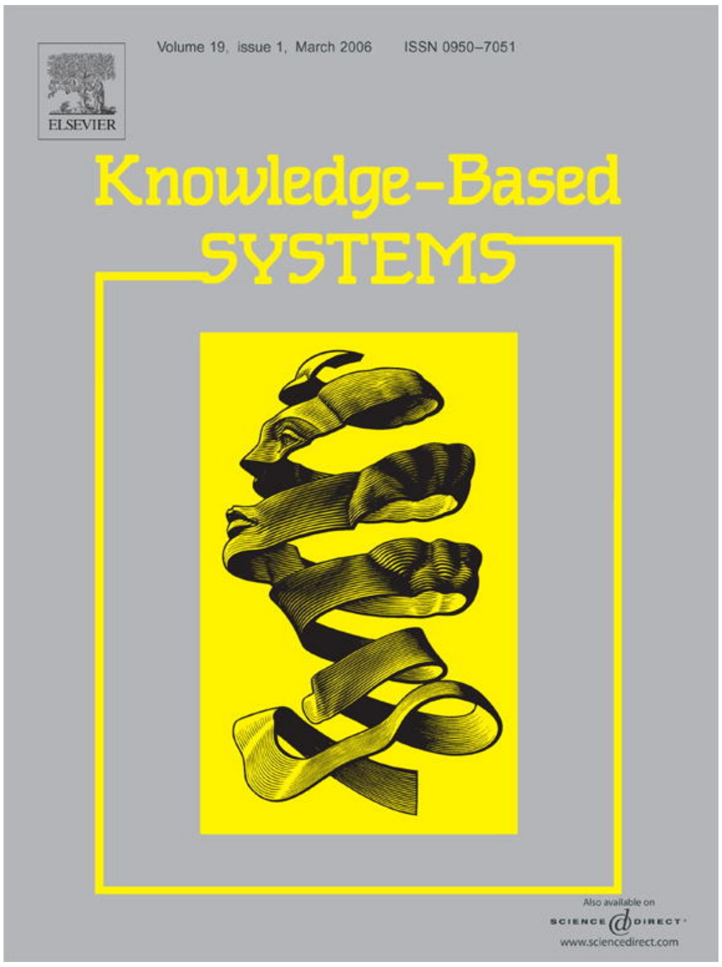

Volume 19, issue 1, March 2006
ISSN 0950-7051
ELSEVIER
Knowledge-Based
SYSTEMS
Also available on
SCIENCE DIRECT
www.sciencedirect.com

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Knowledge-Based Systems 19 (2006) 92–101

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

Translating a knowledge base into HTML

Andreja Andrić [a,*], Vladan Devedžić [b,1], Marko Andrejić [c,2]

[a] *Department of Computer Sciences and Communications, State University of Milan, 20135 Milano, Italy*
[b] *FON—School of Business Administration, University of Belgrade, P.O. Box 52, Jove Ilića 154, 11000 Belgrade, Serbia and Montenegro*
[c] *Military Academy, SNO, Neznanog Junaka 38, 11000 Belgrade, Serbia and Montenegro*

Received 17 October 2003; accepted 10 November 2005
Available online 19 December 2005

Abstract

The problem treated in this paper is the slow response of inference engines, especially in a multi-user environment. In this paper, we present a solution that caches all the possible answers to all the possible consultations, by means of translating the entire knowledge base into a set of HTML documents. We prove that the consumption of memory in this case is acceptable (depends linearly on the number of rules in the initial knowledge base), and that the translational algorithm has polynomial complexity. This solution outperforms any other possible solution, since in this case the infering time becomes equal to zero.
© 2005 Elsevier B.V. All rights reserved.

Keywords: HTML; Translational algorithm; Web; Software legacy; Expert system; Multi user environment

1. Introduction

The expert systems technology failed to bring the expert knowledge to the wider public, because of the difficulties with maintenance and distribution of such systems [2]. In addition to the expense and complexity involved in building systems that attempt to behave as human experts, disseminating such applications to large audiences is logistically difficult. The World Wide Web offers an opportunity for producers and consumers of expert systems. With the proliferation of access to the Web we now have appropriate conditions for bringing expert systems technology—and perhaps the personal expert experience—to a broader audience in a variety of settings. Nowadays the same possibility is extended even more by the technology of wireless devices. This fact has introduced a new challenge to the expert system builders: to port existing expert systems to the World Wide Web, and additionally to make them accessible by means of wireless devices.

On the other hand, a serious drawback of the expert system technology is that the inference engines are most commonly slow, and take much of the CPU time. This problem is additionally intensified having several users accessing the system concurrently, which is the case when considering expert systems on the World Wide Web. With this paper we intend to present one possible solution to these problems.

2. Asking the same question more than once

To deploy a Web-enabled ES, there are a number of architectural approaches from which we may want to start. The most common is the HTML-CGI architecture: the user interacts with HTML entry forms in a Web browser; information entered by the user is sent to the Web server which forwards it to the Common Gateway Interface (CGI) program which then replies with new HTML pages [1,11]. All the expert functionalities reside on the server side (in the CGI program) but the user interacts with it using a standard Web browser.

Another option might be a distributed client–server architecture—a downloadable Java applet contains the user interaction portion of the expert system, and communicates directly with the server application using a socket connection or other inter-program communication mechanism—some of the expert behavior resides in the client, some in the server [12].

Let us consider the following scenario. A user, let us call him A, consults the expert system (E), seeking an advice from it, in an attempt to solve a problem from the corresponding domain. The system asks A to answer several questions, and A

* Corresponding author. Tel.: +39 02 50316346.
E-mail addresses: aandreja@dico.unimi.it (A. Andrić), devedzic@galeb.etf.bg.ac.yu (V. Devedžić).
[1] Tel.: +381 11 3950856; fax: +381 11 461221.
[2] Tel.: +381 11 505 388/36278.

gives to E the required information. During the consultation, E comes to a number of inferences, and finally, on the basis of the user input and on the basis of the inferences brought up, gives an answer—an assessment, a diagnosis, or something similar. After a little while, another user B starts a new consultation. B has the very same problem, and during the consultation he gives exactly the same answers as A did previously. In the end B gets the same answer as A, as we expected. The machine could not guess in advance that B would give the same answers as A did before. Nevertheless, the fact stands that it spent an amount of time to infer the conclusions which it also found just a minute ago, during the first consultation. If we additionally assume that E is a large system and has a vast amount of domain knowledge encoded in a great number of rules, then the time needed to infer the same conclusion twice can be long. Is it possible to store the inference process during a consultation in memory somehow, thus avoiding inferring the same results more than once?

The idea of storing answers to frequent queries in order to speed up the response of the system is well known as *caching*. In this paper, we will consider one possible extrapolation of this idea, which, as we will show, has a number of advantages, namely an expert system formed as a passive tagged document. In the case of a Web enabled system, it should be considered as HTML or XHTML-based. In the case of a system accessible by a mobile device, it should be considered as cHTML or WML-based.

3. The basic idea

Conventional programs, such as compilers for example, deal with an 'infinite input'. This means that a compiler can process an unlimited number of different sequences of symbols on its input. This is possible because of the fact that the compiler processes all the possible inputs in exactly the same way. The task of a compiler is well formalized, and there exists a universal procedure that works in all cases.

On the other hand, expert systems solve completely different type of problems. The problems that are common for expert systems cannot be strictly formalized. Expert systems are intended for solving problems for which there is no straightforward solution, but only a fine differentiation of special cases. The number of such special cases could be enormous, but in any case it is limited, hence the number of all possible inputs to an expert system must be limited [4,5].

Of course, the user can enter any input data, but the expert system will surely check only for a limited number of possibilities, through checking for a limited number of special values and intervals. Suppose the expert system is designed in such a way that, while interviewing the user, instead of letting the user input anything and then checking that input to a limited number of values and intervals, the system offers to the user the very same values and intervals. Expert systems are most often designed in this way [5]. This is more comfortable for the user, and does not introduce additional restrictions.

The number of all the possible answers that an expert system can give is also limited. An expert system will not ask the same questions during different consultations, but the overall number of possible questions is also limited, as well as the number of partial results during the inferring process.

On the basis of the above considerations, it is obvious that the overall number of states in which an inference engine can reside in any moment of a consultation is also limited. A conclusion that follows is that all the solutions and partial inferences that an expert system may come to at any time can be stored somewhere in advance. In that case the machine does not have to infer anything at all, but just to 'recall' what it should infer under particular circumstances.

This last consideration reminds us of the well-known knowledge acquisition problem: an expert can be very good in solving problems in some particular domain, but he/she may not be capable of explaining what rules of thumb he/she actually uses. While mastering some skill, in the beginning, we hold strictly to some learned rules, and we think of every step. The problem solving process is slow. While the practicing period passes, we 'forget' the rules themselves, and get some sort of unconscious memory of what should be done in particular circumstances. We actually recall what we have done numerous times, occasionally forgetting what part of the learned knowledge we are currently using. Returning to our example expert system, we are stating here that a framework of hypertext documents easily makes this remembrance of conclusions possible.

A question that naturally appears here is the question of problem space size and eventual appearance of combinatorial explosion. In other words, two questions might turn up:

1. Is the ratio between problem space size and space occupied on disc in this approach acceptable? Or, differently, what is the dependency between the number of generated HTML documents and the number of rules in the original knowledge base?
2. How long does it take us to 'compile' the original knowledge base to a corresponding collection of HTML documents?

Let us first try to give an answer to the first question. Let us try to give an assessment of how the upper bound of the number of generated HTML pages H depends on the overall number of rules in the initial knowledge base N. For the sake of clarity and simplicity, in this paper we focus on rule-based, backward-chaining mechanisms only. Our ongoing work includes similar analysis of forward-chaining mechanisms, as well as of some other inference techniques.

Backward-chaining inference engine first considers the rules that give the final answer in their THEN part. Let the number of these rules be $N_1 \leq N$. For each of these rules, the inference engine analyzes premises that have to be satisfied in order to satisfy the final conclusion. Then for each of these premises, recursively, it looks for all the rules that satisfy that particular premise in their THEN parts. If for some particular premise there are no such rules, the engine asks the user for the necessary information.

To answer the first question we need the overall number of premises for which there is no rule that satisfies them.

The HTML pages with questions to the users will be generated from such premises. Each final answer will also generate one HTML page with that answer. It is obvious that the number of generated answer pages is equal to N_1.

Thus, it holds that $H = P + N_1$, where P is the overall number of premises for which there is no rule that satisfies them. Let us call these premises 'empty premises'. It is reasonable to assume that the maximum number of empty premises per rule is a constant, i.e. that it does not depend on the number of rules in the knowledge base. Let us call this number k.

The maximum for P is not hard to determine. If N is the overall number of rules, then P is not greater than the maximum number of nodes in a rooted tree [7] with N edges, which (nodes) are incident to exactly one edge, multiplied by k.

Consequently, $P_{\max} \leq N_1 \cdot k \leq N \cdot k$. Thus, $H \leq N \cdot (k+1)$.

In the worst case, H depends linearly on N, so we can conclude that an uncontrolled increase of the number of generated HTML documents cannot happen.

To answer the question concerning the efficiency of the Rules$\rightarrow$HTML algorithm, we will first outline the algorithm itself.

4. The 'Rules to HTML' translation algorithm

As we said before, for any rule-based expert system with backward-chaining reasoning mechanism the following holds:

(1) The number of possible conclusions is finite, and known in advance.
(2) The number of possible entries is either finite or it can be made finite without giving up generality.
(3) The number of possible explanations ('why' and 'how' features) is also finite, and known in advance.

Hence, it stands that for any expert system of this kind, exists a hypertext representation that replicates the system's functionality entirely. The goal is to construct an algorithm that does this translation in the general case.

4.1. Assumptions and constraints

We present here such an algorithm for a restricted (but important) class of rule-based systems—those in which the premises in the If-part (Left-Hand Side) of each rule take only the form of logic clauses. It is possible to derive similar algorithms for other categories of rule-based systems.

The rules in the expert systems of the kind described above have the general form:

IF condition THEN action

where the condition is a logical function of arbitrary number of subconditions:

$f(\text{condition}_1, \text{condition}_2, \ldots, \text{condition}_n) = \text{condition}$

Any logical function can be represented as a Disjunctive normal form (DNF), which does not use brackets.

Any disjunction of the form

IF a OR b THEN c

can be represented by two rules:

IF a THEN c

IF b THEN c

Hence, to represent an arbitrary logical function of any number of variables by rules, it is sufficient to use AND and NOT, without brackets.

Every condition takes the form of

$a\rho b$

where ρ is a relation.

Every action can be said to be like: 'put a and b into a new relation ρ_1'.

4.2. Phases of translation

The translation has three phases:

First phase: translation of rules from the variable-relation level to the condition level.

Second phase: translation of rules from the condition level to a graph of all the possible paths in evaluating the variables.

Third phase: merge the nodes from the previous phase into the hypertext documents

It is important to say here that in this study we tried to offer a solution that is language-independent, thus avoiding any reference to any particular software tool or language for building expert systems. We did not fully implement the third phase, since it is clearly language dependent and it is not yet clear how to do it without taking into account specific features of the language in which the initial system is built. So, in our solution, the result of the second phase is already a hypertext presentation, which deals with boolean variables obtained from the first phase. Hence we only discuss the first two phases here.

Before moving on to description of the phases of translation, we shall introduce an important term: circular inference. Let us consider the following example set of rules:

1. IF a THEN b
2. IF b THEN c
3. IF c THEN a

Suppose that the inference engine is trying to detect whether a is TRUE or not. It will start from the rule number 3, since it is the only rule that contains the variable a on its right hand side. Since, according to the rule 3 a depends on c, it will try to evaluate the rule number 2. It will then discover that c depends on b, so in the next step it will try with the rule number one, and discover that b depends again on a, so the circle is closed. This set of rules is obviously invalid, since it does not lead to a conclusion. The algorithm proposed also detects these cases.

4.2.1. First phase of the translation

This first step is straightforward—denote each relation used in the system a new boolean identifier. For example, let us assume that in the system some X is checked whether it is 1 or 2 or 'between 3 and 7', and Y is checked whether it is 'bird', 'elephant' or 'something else'.

Then, we shall introduce the following new boolean variables:

$a1=$"$X=1$"
$a2=$"$X=2$"
$a3=$"$3\leq X\leq 7$"
$a4=$"Y is bird"
$a5=$"Y is elephant"
$a6=$"Y is something else"

Clearly, there exist some dependencies between some of the new boolean variables. But, we state here that these dependencies are of no importance for the translation. Omitting them does not harm the generality, but adds to it.

4.2.2. Second phase of the translation

Now the rules in the knowledge base have the form as in the following example:

$a!bc = f$

which means IF a AND NOT b AND c THEN f, where a, b, c and f are boolean variables. Brackets are not supported. Supported logical operators are AND and NOT.

The algorithm has the main loop that calls the procedure *processLeftHandSide*. Two data structures are maintained: the goal stack, and the variable stack, both initially empty. The main loop works as follows:

1. Generate the header of HTML document that will serve as the starting page, *start.htm*. Inside it, open an unordered list, with the tag <ul>. Initialize PageID with 0.
2. Find all the variables that never appear on the left-hand side of any rule. Put them on the goal stack. Pick the variable from the top of the goal stack and mark it as 'current'.
3. Take the last rule with the current variable on its right-hand side and make it the 'current rule'.
4. Initialize the string variable YES to contain the name of the current variable, concatenated with the string 'AnswerYes.htm'. Initialize the string variable NO to contain the name of the current variable, concatenated with the string 'AnswerNo.htm'.
5. Generate two HTML files, with the names that are now in variables YES and NO. In these two files, write '<current variable> is true' and '<current variable> is false', respectively. For example, if the current variable is alpha, two files would be generated: alphaAnswerYes.htm and alphaAnswerNo.htm. The first will contain the text 'alpha is true', the other 'alpha is false'.
6. Put the current variable onto the variable stack. Call the procedure processLeftHandSide, with the following sequence of attributes: current rule, YES, NO. Put the resulting string into NO. Empty the variable stack.
7. Going through the rules backwards (from the current rule to the first one) find a rule that also has the current variable on its right-hand side. If it is found, consider it as the current rule, and go to step 6. Otherwise, continue to step 8.
8. In start.htm, add a new line in the unordered list. The new line will contain the name of the current variable, and will be hyperlinked to the file whose name is in NO. Pop the current variable from the goal stack. If the goal stack is empty, continue to step 9. Otherwise, pick the next variable from the goal stack and go to step 4.
9. Close the unordered list and finish the start.htm file.

The procedure processLeftHandSide takes as its input a rule that we call the current rule, and two strings, YES and NO. The result is a string. It works as follows:

1. Take the rightmost variable of the left-hand side of the current rule. If there are no variables, just return YES and exit the procedure.
2. If that variable is already on the variable stack, then check if it is at the bottom. If it is, then report that the circular inference is detected and exit the program. Otherwise, proceed to the next variable, to the left, and go to step 2. If a variable is found that is not already on the variable stack, go to step 3. If there are no more variables, just return YES and exit the procedure.
3. If the variable appears on the right-hand side of any other rule, go to step 4. Otherwise, go to step 5.
4. If the unary operator NOT was applied to the current variable, then call *processLeftHandSide* with the following sequence of arguments: rule found in step 3, NO and YES, and the result put into YES. Otherwise, call *processLeftHandSide* with the following sequence of arguments: rule found in step 3, YES and NO, and the result put into YES. Take the next rule (counting upwards) with the same variable on its right-hand side (if it exists) as the current rule, and go to step 1. If there are no more rules of that kind, return YES.
5. Generate a new HTML document with a question about the value of the current variable. The name of the document is the pageID, concatenated with '.htm'. It contains the question, with the answers 'yes' and 'no'. For example, 'the value of The_car_cannot_move is: Yes/No'. If the unary operator NOT was not applied to the current variable, then the text 'yes' will point to the file whose name is in YES, and the text 'no' will point to the file whose name is in NO. If NOT was applied, then the other way round. Put the name of this document into YES, and proceed to the next variable to the left. If there is any, then increase pageID and go to step 2. Otherwise, if all the variables are processed, then return YES and exit the procedure.

The main loop and the procedure processLeftHandSide can be seen conveniently on the flow charts in Figs. 1 and 2. Steps 2, 4 and 5 from processLeftHandSide are elaborated in more detail, and embraced in additional boxes indicating proper steps.

Now, the start.htm file contains a choice of the questions that the system can answer (these are, in fact, goals). The user

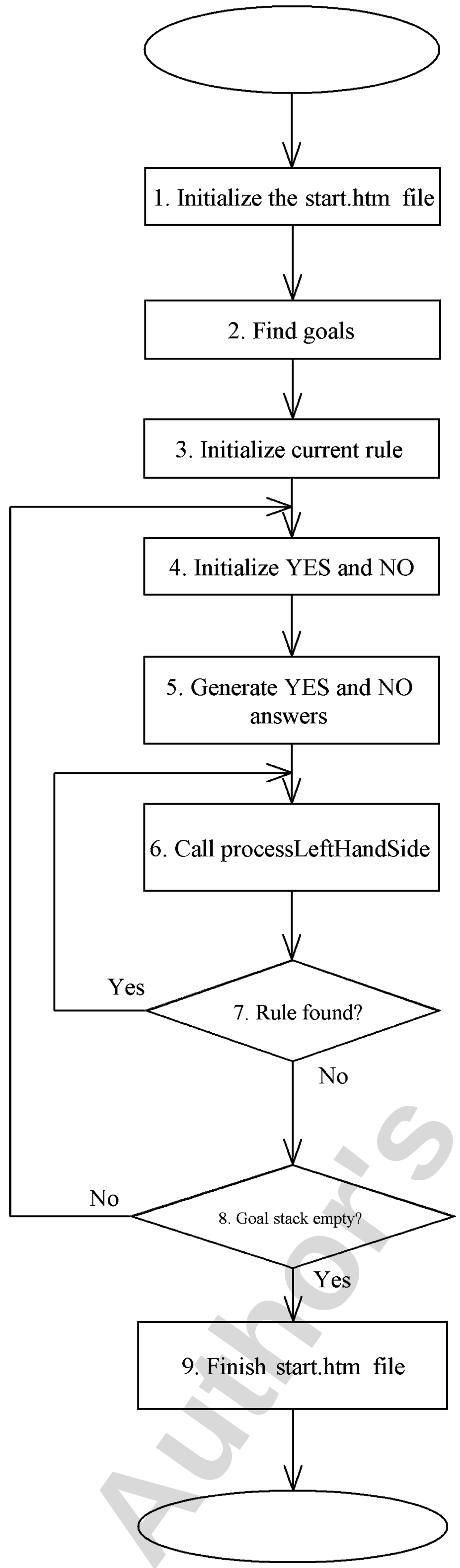

1. Initialize the start.htm file
2. Find goals
3. Initialize current rule
4. Initialize YES and NO
5. Generate YES and NO answers
6. Call processLeftHandSide
7. Rule found?
Yes
No
8. Goal stack empty?
No
Yes
9. Finish start.htm file

Fig. 1. Flow chart of the main loop.

clicks on one of them, and the consultation begins. It proceeds normally as in the initial expert system.

The following section shows that the complexity of the above algorithm is polynomial.

4.3. The algorithm complexity

The complexity of the algorithm clearly depends on three parameters: the number of rules, the maximum number of variables that can appear on the left-hand side of any rule, and the knowledge base structure itself. Under the term 'knowledge base structure' we here refer to the way in which the rules are chained. Let N be the overall number of rules in the knowledge base. Let a be the maximum number of variables that can appear on the left-hand side of any rule. The goal here is to prove that for any knowledge base structure, the time for evaluating a variable will be a polynomial function of a and N. The first issue we have to deal with is to count and collect the variables which never appear on the left-hand side of any rule. This is easy to check, and is done only once. The complexity of this part is clearly polynomial.

In any case, the overall number of variables which never appear on the left-hand side of any rule can be at maximum N, i.e. equal to the number of rules. In this case each rule has a distinct variable on its right-hand side, and all the variables on these right-hand sides are different. In general, some rules can have the same variable on their right-hand sides, and some variables may appear on the left-hand sides of some other rules.

Looking at the main loop of the algorithm, we see that the algorithm picks up the first goal from the stack, and then checks for all the rules that have this variable on their right-hand sides. The procedure *processLeftHandSide* is called for every such a rule. Then the current variable is removed from the goal stack, and the process is repeated until the goal stack becomes empty.

It is easy to see that in this way, in any case, each rule will be considered at most once. In this main loop, in other words, only the rules that have the goals on their right-hand sides will be considered.

Hence, the complexity function K is smaller or equal to:

$$K \leq N \cdot K_1$$

where K_1 is the complexity of the *processLeftHandSide* procedure.

However, the recursive procedure *processLeftHandSide*, called for one rule, can recursively process at most N rules. Processing any rule more than once is prevented by means of checking the variable stack. Namely, when the procedure *processLeftHandSide* is nonrecursively called for one rule, the variable stack is initialized, and no variable that appears on the left-hand side of any rule can be processed more than once. If we define the processing step as the processing of a single variable on the left-hand side, then processing a single rule takes at most a steps. So, the procedure *processLeftHandSide*, nonrecursively called for one rule can take at most $r \cdot a$ steps. One step, however, includes checking the variable stack. Hence the complexity function K_1 of the *processLeftHandSide* is smaller or equal to:

$$K_1 \leq N \cdot K_2$$

where K_2 is the complexity function of the procedure that checks the variable stack. The size of the variable stack,

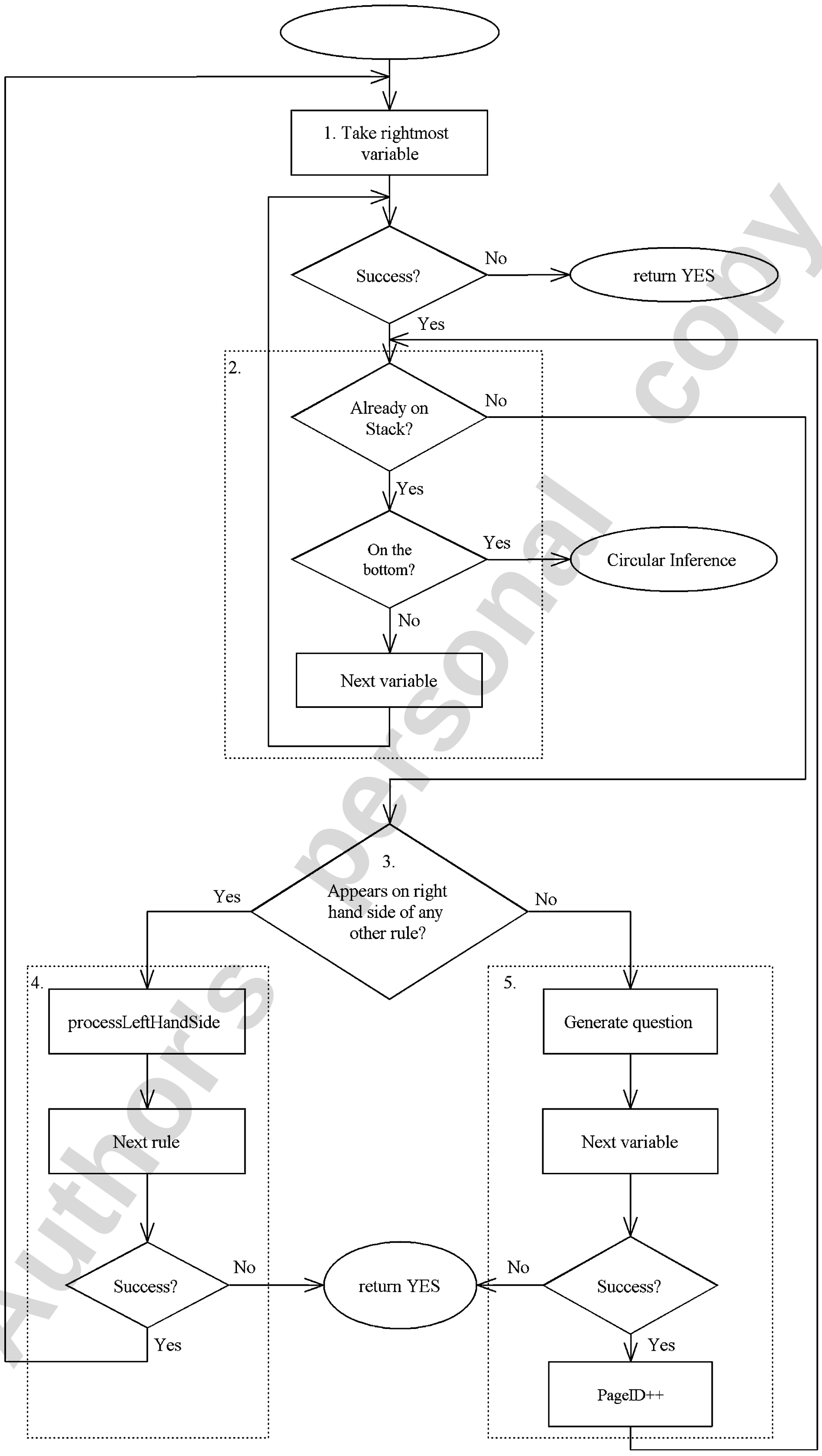


Fig. 2. Flow chart of the procedure processLeftHandSide.

however, must be smaller than $N \cdot a$, since no variable can occupy it more than once, in a single nonrecursive call to *processLeftHandSide*.

Consequently, the complexity of the whole algorithm is smaller or equal to:

$$K \leq N^3 \cdot a^2$$

and, hence, it is clearly polynomial. The following Section 5 shows how a hypertext document could be practically formed to act as an expert system.

5. A simple example

Suppose that we have a knowledge base with frames and rules, already developed using a conventional expert system shell. Then it is possible to develop a kind of a *translator*—a program which translates that particular knowledge base to a framework of mutually interconnected HTML documents. Each document represents one state of the inference engine. Links for a particular state lead to other states, representing the possible answers a user can give to a particular question.

We will illustrate this on a tiny example from the area of diagnostics of malfunctions on automobile engines. Suppose we have seven rules. These rules state as shown in Table 1.

Assume also, as before, that backward-chaining is used. The goal is to discover the cause of the proposed malfunction. Rules with smaller ordinal numbers have greater priority than those with larger ordinal numbers.

We can identify all the possible consultations with this system. All these consultations are given in a dialogue form in Table 2.

On the basis of these consultations we can identify states of the system. These are:

1. The starting state is the state in which the system asks the user what the problem is.
2. The problem is in insufficient engine strength. The system asks if an increased consumption of fuel is noticed.
3. The problem is in insufficient engine strength. An increased consumption of fuel is noticed, too. The system concludes that the cause is ‘Valves burnt up or used up or Piston rings used up’.
4. The problem is in insufficient engine strength. An increased consumption of fuel is not noticed. The system concludes that the cause is ‘Valve coil broken’.
5. The problem is an increased consumption of fuel noticed during drive. The system concludes that the cause is ‘Valves burnt up or used up or Piston rings used up’.
6. The problem is an increased consumption of motor oil noticed during drive. The system asks the user whether he/she noticed something from the following list: insufficient engine strength, or increased consumption of fuel, or oil spots under parked car.
7. The problem is an increased consumption of motor oil noticed during drive. Also, the strength of the motor is insufficient. The system concludes that the cause is in piston rings used up.
8. The problem is an increased consumption of motor oil noticed during drive. Also, the consumption of fuel is increased. The system concludes that the cause is in piston rings used up.
9. The problem is an increased consumption of motor oil noticed during drive. The oil spots under car are also noticed. The system concludes that the cause of this malfunction is engine oil leak.
10. The problem is an increased consumption of motor oil noticed. No other symptom is noticed. The system concludes that the cause of this malfunction is ‘Valve reins used up’.

The inference graph of the above decision-making process is represented in Fig. 3. The nodes of this graph represent the states of the system, and the edges represent the state transitions following from the corresponding user inputs. Each node is represented with a textbox containing the text that should be seen on the corresponding HTML page. Each node is also numerated with a number inside a circle.

Each state can be represented with one HTML page. State No. 1 would contain the question ‘What is the problem?’ and the three possible answers to this question: ‘Insufficient engine strength’, ‘Increased consumption of fuel’, and ‘Increased consumption of motor oil.’. These answers will be hyperlinked, pointing to the HTML documents representing states 2, 5 and 6, respectively.

Table 1
Rules for automobile engines

	IF	THEN
1	Problem is ‘Insufficient engine strength’ AND Engine spends too much fuel	Problem is caused by ‘Valves burnt up or used up’ or ‘Piston rings used up’
2	Problem is ‘Insufficient engine strength’ AND Engine doesn’t spend too much fuel	Problem is caused by ‘Valve coil broken’
3	Problem is ‘Engine spends too much fuel’	Problem is caused by ‘Valves burnt up or used up’ or ‘Piston rings used up’
4	Problem is ‘Too much engine oil spent’ AND Engine strength is insufficient	Problem is caused by ‘Used up piston rings’
5	Problem is ‘Too much engine oil spent’ AND ‘Engine spends too much fuel’	Problem is caused by ‘Used up piston rings’
6	Problem is ‘Too much engine oil spent’ AND ‘Oil spots under the parked car noticed’	Problem is caused by ‘Engine oil leak’
7	Problem is ‘Too much engine oil spent’	Problem is caused by ‘Valve reins used up’

Table 2
All the possible consultations with the example system

1	E: What is the problem with your engine? U: Engine strength insufficient E: Does your car spend too much fuel? U: Yes E: Possible causes of this malfunction are: Valves burnt up or used up or Piston rings used up
2	E: What is the problem with your engine? U: Engine strength insufficient E: Does your car spend too much fuel? U: No E: The cause of this malfunction is: Valve coil broken
3	E: What is the problem with your engine? U: My car spends too much fuel. E: Possible causes of this malfunction are: Valves burnt up or used up or Piston rings used up
4	E: What is the problem with your engine? U: Too much engine oil spent E: Do you feel that your engine strength is insufficient during driving? U: Yes E: Possible cause of this malfunction: Piston rings used up
5	E: What is the problem with your engine? U: Too much engine oil spent E: Do you feel that your engine strength is insufficient during driving? U: No E: Does your car spend too much fuel? U: Yes E: Possible cause of this malfunction: Piston rings used up
6	E: What is the problem with your engine? U: Too much engine oil spent E: Do you feel that your engine strength is insufficient during driving? U: No E: Does your car spend too much fuel? U: No E: Did you notice oil spots under your parked car? U: Yes E: The cause of this malfunction is: Engine oil leak.
7	E: What is the problem with your engine? U: Too much engine oil spent E: Do you feel that your engine strength is insufficient during driving? U: No E: Does your car spend too much fuel? U: No E: Did you notice oil spots under your parked car? U: No E: The cause of this malfunction is: Valve reins used up

E denotes the expert system, and U denotes the user.

The states No. 3, 4, 5, 7, 8, 9 and 10 contain the diagnoses the system can give. The corresponding HTML documents will contain only the problem diagnoses, and links to the starting page.

The state no. 2. will contain a question ‘Is an increased consumption of fuel noticed?’ and two answers: ‘Yes’ and ‘No’, which lead to states 3 and 4, respectively.

The state No. 6 will contain a question ‘Did you notice something from the following list:’ and four possible answers: ‘Insufficient engine strength’, ‘Increased consumption of fuel’, ‘Oil spots under parked car’, and ‘Nothing of the above’. These answers will lead to the states No. 7, 8, 9 and 10, respectively.

After this small example, in which we outlined a method of transforming a conventional expert system into a Web-based one formed as a passive Web document, a natural question can appear: OK, this works well for a small system, but is it really applicable to real-world problems? To answer this question, let us now consider a larger-scale example, i.e. a real expert system that is completely built using the above described technique.

6. The *Defector* system

The *Defector* is a recently developed expert system for technical state diagnostics of cars and other technical systems. It is based on classical frames-and-rules methodology, and its knowledge consists of about 400 rules and 90 frames. For inferring conclusions, the backward-chaining method is used.

The knowledge that is incorporated in the developed system has been acquired through collaboration with numerous experts in construction, diagnostics and technical systems maintenance. A part of the knowledge base is directly gathered from everyday experience in detecting malfunctions.

Defector was initially designed, developed, and tested using a conventional expert system shell. In the next step, a small translator program called *deftohtm* was written in the C language to automatically translate the knowledge base into about 450 mutually interconnected HTML files. *Deftohtm* was not intended to be a general translator of knowledge bases, but only to translate this particular one. Usage of *Defector* requires only a standard Web browser. Fig. 4. depicts one of *Defector's* pages.

7. Discussion

The advantages of the described method of building Web-based expert systems are obvious. To make the inference process happen in such an expert system no software is needed at all, except the Web server, which already exists on a server machine, regardless of the expert system itself. Any Web browser is sufficient to make an access to such a system. The system works on any platform. The system responds practically immediately, and the response speed depends solely on the performance of the Internet connection used. The problems with managing simultaneous access of more than one user are completely avoided.

The HTML language makes possible to insert multimedia contents easily and to design a high-quality user interface. The multimedia contents, pictures, sound, animation, etc. can additionally illustrate the conclusions the system gives, and almost make the conventional explanation mode unnecessary.

The pages with final conclusions may contain hyperlinks to the relevant pages on the World Wide Web. For example, when the system gives the following diagnosis: ‘This problem is caused by the motor oil leak’, the word ‘leak’ contains a hyperlink. A click on this link gives a detailed explanation why and under what circumstances an oil leak happens, combining

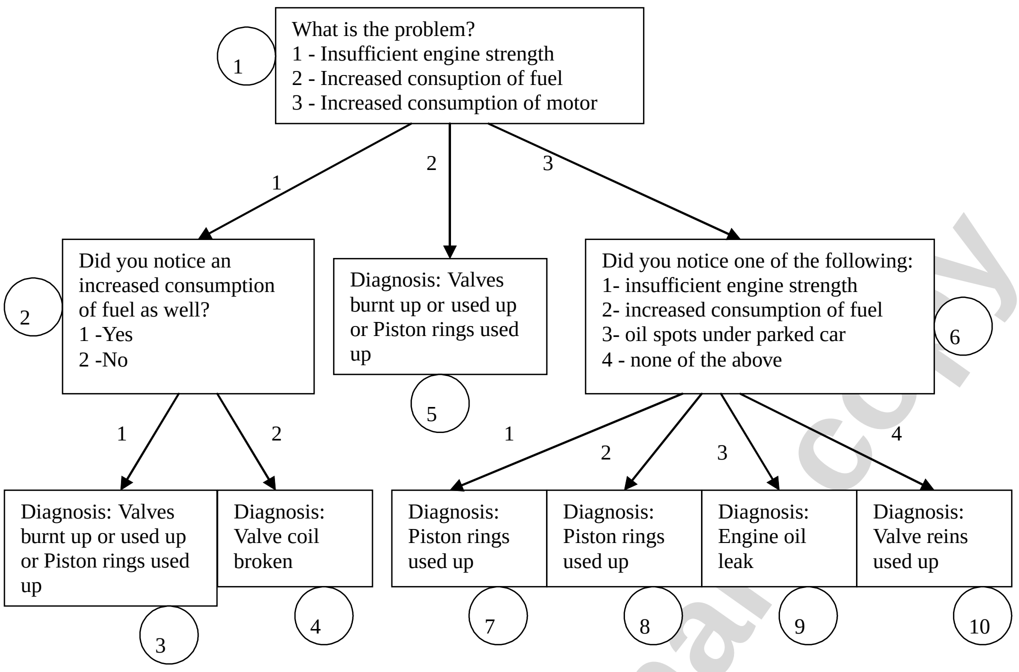

What is the problem?
1 - Insufficient engine strength
2 - Increased consuption of fuel
3 - Increased consumption of motor
1
Did you notice an increased consumption of fuel as well?
1 -Yes
2 -No
2
Diagnosis: Valves burnt up or used up or Piston rings used up
5
Did you notice one of the following:
1- insufficient engine strength
2- increased consumption of fuel
3- oil spots under parked car
4 - none of the above
6
Diagnosis: Valves burnt up or used up or Piston rings used up
3
Diagnosis: Valve coil broken
4
Diagnosis: Piston rings used up
7
Diagnosis: Piston rings used up
8
Diagnosis: Engine oil leak
9
Diagnosis: Valve reins used up
10

Fig. 3. Inference graph corresponding to the example presented in the text.

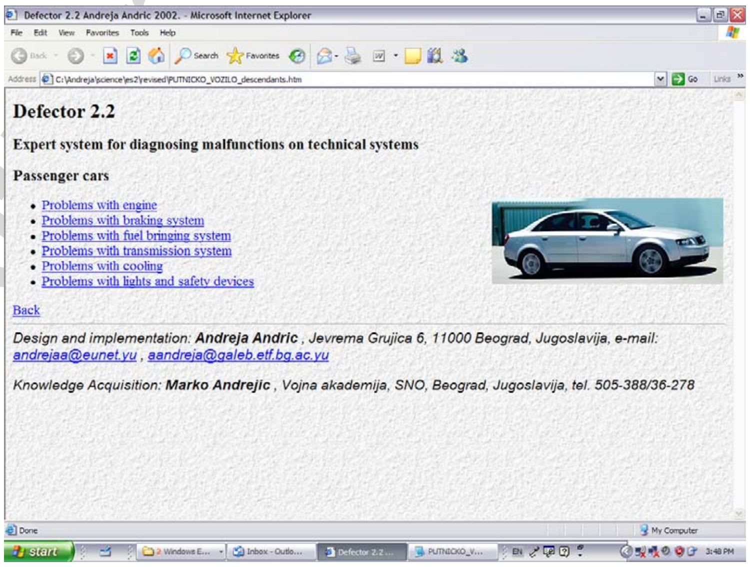
text, illustrations, and other hyperlinks. The program which translates the original knowledge base into a collection of HTML pages could insert these hyperlinks automatically. In order to make this possible, these insertions could be hardcoded into the translator program, or, otherwise, a table should be provided, which links every final state to a list of corresponding hyperlinks [6]. An important general hyperlink 'details' could be also added. This hyperlink opens a page with hyperlinks to Web pages of repair shops, shops with spare parts, advices of good connoisseurs of the problem, automobile parts manufacturing companies, etc.

On the other hand, consistency and maintenance of such a system could be a problem. The original expert system should be developed with a conventional expert system shell, as stated above, that introduces several new steps in the development process:

- building a translator program,
- translating the whole system.

These new steps and possible errors that can emerge out of them call for additional attention during the development and maintenance process.

All possible conclusions are already inferred and memorized. The memory consumption is by far larger than with the classical technology, but the increase of speed is enormous, and today the CPU time consumption is much more expensive than the consumption of memory.


Defector 2.2 Andreja Andric 2002. - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address
C:\Andreja\science\es2\revised\PUTNICKO_VOZILO_descendants.htm
Go
Links
Done
My Computer
start

Defector 2.2

Expert system for diagnosing malfunctions on technical systems

Passenger cars

- Problems with engine
- Problems with braking system
- Problems with fuel bringing system
- Problems with transmission system
- Problems with cooling
- Problems with lights and safety devices

Back

Design and implementation: ***Andreja Andric*** *, Jevrema Grujica 6, 11000 Beograd, Jugoslavija, e-mail: andrejaa@eunet.yu , aandreja@galeb.etf.bg.ac.yu*

Knowledge Acquisition: ***Marko Andrejic*** *, Vojna akademija, SNO, Beograd, Jugoslavija, tel. 505-388/36-278*

Fig. 4. One of Defector's pages.

An illustrative, although rather simple car troubleshooter can be found in [10]. It is a conventional Web site, written by hand, and therefore much harder for updating, in comparison to the solution proposed here.

The solution presented has its limitations. In its pure form it is applicable only to 'pure' expert systems, or advisory systems. If the initial system requires some calculations or other procedural actions, then we must apply a hybrid solution. For example, if the system requires some calculations, they may be provided by exploiting Java applets or CGI scripts. The relatively large number of generated HTML documents is not a problem, but if we introduce the explanation support facility, the matters become a little more complicated.

The explanation support facility could be provided in an identical way (because of the limited number of possible explanations), which additionally increases the number of generated HTML documents. This solution is good if the disc space consumption is acceptable. The other solution is to give up the conventional explanation utility. Explanations of the type "The diagnosis is… because of the rule 142 which states…" are of no much value for the user anyway. Instead of this kind of explanations, the above mentioned illustrations could be sufficient.

It is very common nowadays to use a templating system of one sort of another to format content retrieved from a database on the fly in response to a consumer request. A possible alternative implementation could be based on this approach.

Some interesting questions for future study would be: can the web link in general be used to mimic the rule-based inference process? Or can the inference process be encoded as web links? In our daily life, we often look for answers by clicking links from one web site to others. If the links are defined and organized according to rule definitions, then the answer can be more easily found. That is because some rule-based knowledge is encoded into the link and the link becomes knowledge link, which would be an interesting topic for semantic web research.

Another interesting question that might appear here is whether the result of the above described procedure can be considered to be an expert system at all. The 'compiled knowledge' which is the result of the translational procedure lacks the transparency and self-explanation ability which is the most esential property of the expert systems. But, as we explained in the previous sections, the resulting set of WML pages, accessed by a mobile device acts identicaly as the original system. So, the question moves to a more philosophical field of whether some two entities that behave in exactly the same way have the same essence, as addressed in the famous Turing test.

8. Conclusion

In this paper, we presented a concept of a Web-based expert system organized as a passive Web document. We discussed the problem of running the same inference process more than once in conventional expert systems, and proposed a solution to this problem. We outlined a method for translating a knowledge base into a framework of HTML documents and used a small example to illustrate the method. We also discussed the amount of memory and time needed to do the desired translation. We presented *Defector*, a recently developed Web-based expert system for diagnosing malfunctions on cars as a larger-scale example. We discussed the advantages of this concept, and gave an idea of future work in this direction.

This study could help expert system developers make their products faster and more attractive. Publishers of those knowledge bases such as trouble-shooting, help-desk or recommender systems [3,8,9] to a large number of customers can, starting from the results of this study, make their consulting or recommendation systems more attractive and more profitable. Theoretical results of this study could also lead to exploring new ways of knowledge representation. Finally, from the practical part of this study, the users will have an opportunity to get a high quality expert knowledge on car malfunctions any time, at no cost.

Our future work has two directions. First, we are currently designing algorithms for translating other types of knowledge bases into collections of HTML documents. Second, we are improving the *Defector* system by extending its knowledge base and by making its user interface design more attractive.

References

[1] A. Mitrovic, Porting SQL-Tutor to the web, in: C. Peylo (Ed.), Proceedings of the International Workshop on Adaptive and Intelligent Web-based Educational Systems, 2000, pp. 50–60.

[2] H. Eriksson, Expert systems as knowledge servers, IEEE Expert 11 (3) (1996) 14–19.

[3] J. Breese, D. Heckerman, K. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, Madison, WI, 1998.

[4] J.C. Giarratano, G.D. Riley, Expert systems: principles and programming, Course Technology, third ed., 1998, ISBN 0534950531.

[5] J. Durkin, Expert Systems: Design and Development, first ed., McMillan Col. Division, New York, 1998, ISBN 0023309709.

[6] L. Hardmann, L. Rutledge, D. Bulterman, Automated generation of hypermedia presentations from pre-existing, tagged media objects, Proceedings of the Second Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98, Pittsburgh, USA, 1998.

[7] N. Deo, Graph Theory with Applications to Engineering and Computer Sciences, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[8] P. Resnick, H. Varian. Recommender systems, Special issue on recommender systems, Communications of the ACM 40 (3) (1997). http://www.acm.org/cacm/MAR97/resnick.html

[9] R. Sinha, K. Swearingen, The role of transparency in recommender systems, in: Proceedings of ACM CHI 2002 Conference on Human Factors in Computing Systems Conference Companion, 2002.

[10] Samarins Diagnostic Web Site [online], available at http://www.samarins.com/diagnose/index.html.

[11] S.R. Alpert, M.K. Singley, P.G. Fairweather, Porting a standalone intelligent tutoring system to the web, in: C. Peylo (Ed.), Proceedings of the International Workshop on Adaptive and Intelligent Web-based Educational Systems, Montreal, Canada, 2000, pp. 1–11.

[12] W.D. Potter, X. Deng, J. Li, M. Xu, Y. Wei, I. Lappas, A Web-based expert system for gypsy moth risk assessment, Computers and Electronics in Agriculture, 2001, http://webster.cs.uga.edu/~potter/dendrite/iufro-gypsy.PDF.