NEAT EXPLANATION OF PROOF TREES

Agneta Eriksson and Anna-Lena Johansson

UPMAIL, Uppsala University
P.O. Box 2050, S - 750 02 Uppsala
Sweden

ABSTRACT

It is essential that the reasoning of an expert system can be explained and justified. In the paper simple ideas for providing comprehensive explanations of deductions are presented. The method is to transform the proof tree to a more comprehensive proof tree from which a neat explanation directly can be extracted.

The transformation of proof trees is a way to view the problem of explaining the reasoning of an expert system. The ideas presented in this paper are easy to implement. Furthermore, the transformation for reducing chains of specifications of a concept seems to be applicable in general. The transformation for introducing a new name for a compilation of concepts depends on the problem domain.

I INTRODUCTION

A fundamental part of an expert system design is its ability to explain and justify a line of reasoning (Parsaye, 1983), (Swartout, 1981), (Walker, 1983), (Wallis and Shortliffe, 1984). In this paper we outline some principles for providing comprehensible explanations of deductions, i.e. why a fact is derived. The interesting problem of explaining why a fact cannot be derived is not the subject of this paper.

We will focus on expert systems designed and implemented in a logic programming environment. The domain knowledge is expressed as an axiom set in the form of Horn-clauses. A program clause either describes a sentence that is unconditionally true, such as "The color of grass is green"

Color(Grass,Green) <-

or a conditional sentence, "x and y are playmates if both x and y are children and live at the same place"

Playmate(x,y) ← Child(x), Child(y), Place(z), Lives(x,z), Lives(y,z)

In a logic programming language such as Prolog (Pereira *et a*/, 1977), (Carlsson and Kahn, 1984) answers to questions are deduced from the set of axioms by resolution (Robinson, 1979).

II PROOF TREES

We illustrate the ideas with easy and comprehensible relations.

Two persons x and y are related if they have an ancestor in common.

Related(x,y) ← Ancestor(x,z), Ancestor(y,z) (1)

The individual y is an ancestor of x if the parent of x is y or if y is an ancestor of the parent of x.

Ancestor(x,y) ← Parent(x,y) (2)
Ancestor(x,y) ← Parent(x,z), Ancestor(z,y)

This work is supported by the National Swedish Board for Technical Development (STU)

Anyone who is either a father or a mother is a parent.

Parent(x,y) ← Father(x,y) (4)
Parent(x,y) ← Mother(x,y) (5)

In order to draw conclusions about family-relations concerning specific individuals we need some basic facts about the fathers and mothers. We state that: the father of Ada is John, ... , the mother of Tim is Pia.

Father(Ada,John) ← (6)
Father(Per,John) ← (7)
Father(Pia,Per) ← (8)
Father(Eva,Per) ← (9)
Mother(Tom,Eva) ← (10)
Mother(Tim,Pia) ← (11)

Let us study an AND-tree constructed for the proof of the formula Rdated(Ada,Tom). The root of the proof tree is Related(Ada,Tom). When both Ancestor(Ada,z) and Ancestor(Tom,z) are deduced we can apply rule (1) to obtain Related(Ada,Tom). In Figure 1 below, the complete proof tree is shown.

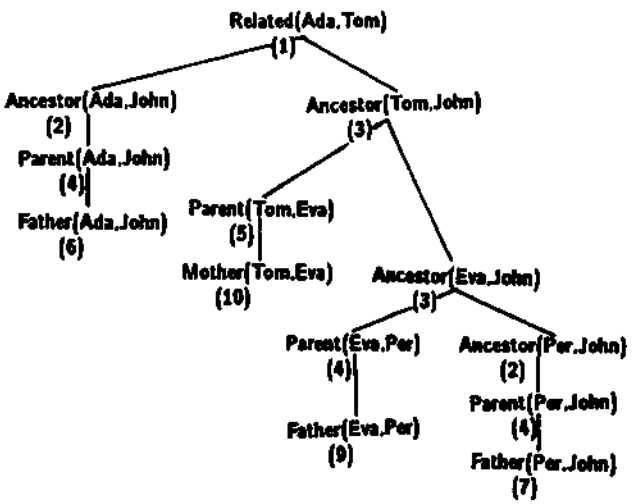

Figure 1. A proof tree for the formula Related(Ada,Tom)

III TRANSFORMATION OF PROOF TREES

Let us try to formulate rules for transforming proof trees.

When we interpret the rules as "definitional" in the sense that the precondition is a restatement of the conclusion, then in a rule of the form

P ← Q (12)

Q is a specification of the concept P; Q is more precise than P. If we find

an application of a specification rule in the proof tree, we can compact the tree by eliminating the consequence of the rule. Formally we can justify the transformation by the following reasoning: the two rules

$$A \leftarrow P_1, \ldots, P_{i-1}, P_i, P_{i+1}, \ldots, P_n \quad (15)$$
$$P_i \leftarrow Q$$

implies the rule

$$A \leftarrow P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n \quad (15)$$

An application of the two former rules (13) and (14) can be replaced by an application of the new rule (15). The new rule is redundant, it does not add to the derivable set of theorems but the rule provides us with a neater explanation of the proof tree. However, we do not want to increase the number of eligible rules. The rules derived from the original axiom set are used only as rewrite rules of the proof trees. All nodes in a proof tree except the root and the leaves can be eliminated by deriving and applying deduced rules. Rules of the type (12) give an example of nodes that can be eliminated without loss as long as the chain has a limited length. Since the depth of an hierarchy of definitions tends to be shallow (Amsler, 1080) the transformation seems to be useful. However, it is essential that nodes important to the understanding of the solution are kept in the proof tree. If the rules are of the type "cause-effect", where the conclusion follows from the preconditions by some mechanism, the chain of reasoning can be much longer. In order to distinguish rules illustrating a principal idea in the domain, rules could be augmented with a measure of importance in the manner suggested by (Wallis and Shortliffe, 1984). The use of importance metrics will then inhibit the elimination of concepts Pj if the importence measure exceeds a certain limit.

Using the described technique we can derive a redundant rule

$$\text{Ancestor}(x,y) \leftarrow \text{Father}(x,y) \quad (16)$$

from the two rules (2) and (4). Consequently, in a proof tree, all subtrees to the left in the figure below can be replaced by the subtree to the right.

Let us try to make the proof tree presented in Figure 1 easier to grasp by deducing further redundant rules. The rule

Related(x,y) - Father(x,z), Ancestor(y,z) (17)

can be derived from rules (1) and (16). The rules

Ancestor(x,y)<- Father (x,z), Ancestor(z,y) (18)
Ancestor(x,y) - M other (x,z), Ancestor(z,y) (19)

can be derived from rale (3) together with (4) and (5) respectively. From (18) and (16) we can derive

Ance$tor(x,y)<- Father(x,z), Father(z,y) (20)

The proof tree for a proof that John is an ancestor of Ada is a subtree of the tree in Figure 1. Applying the derived rules, (16) - (20) we get the following tree

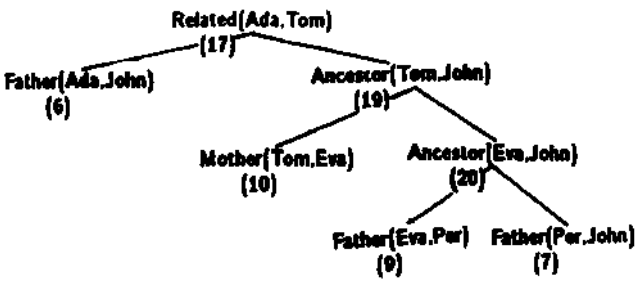

Figure 2. A transformed proof tree for the formula Related(Ada,Tom)

Another way to transform a proof tree is to introduce a new concept that is a specification of an earlier concept and a compilation of other concepts already present. In our example we can introduce the rewrite rules

$$\text{Ancestor}(x,y) \leftarrow \text{Grandfather}(x,y) \quad (21)$$
$$\text{Grandfather}(x,y) \leftarrow \text{Parent}(x,z), \text{Father}(z,y) \quad (22)$$

Grandfather is a specification of a certain type of ancestorship, we can break up the recursive chain of ancestors by introducing grandparents. By adding the rule (21) we are able to eliminate the ancestor node in situations where the rule (22) is applicable.

We can derive the rules

Ancestor(x,y) <- Mother(x,z), Grandfather) z,y) (23)
Grandfather(x,y) <- Father(x,z), Father(z,y) (24)

The proof tree in Figure 2 can be reduced to the more comprehensible tree where the occurrence of node Ancestor(Eva,John) is replaced by the node Grandfather!Eva,John).

The resulting proof tree depends on the rewrite rules that are provided. Each user can in these rules express their own view. When there are competing rewrite rules a selection mechanism has to be used.

IV PRESENTATION OF PROOF TREES

A. Stepwise presentation

When we have a non trivial proof tree it is essential to make the explanation sufficiently abstract. We make use of the tree structure for a stepwise presention of the proof tree. The first step of the explanation includes the root and its sons. The next step focus on some of the sons. This also offers the user possibilities to direct the presentation according to the her interests.

We look at a transformed tree for the proof of Related(Pia,Tom). In this tree both a new rule and a new fact appear

$$\text{Sister}(x,y) \leftarrow \text{Parent}(x,z), \text{Parent}(y,z), \text{Woman}(y) \quad (25)$$
$$\text{Woman}(\text{Eva}) \leftarrow \quad (26)$$

The tree is shown in Figure 3.

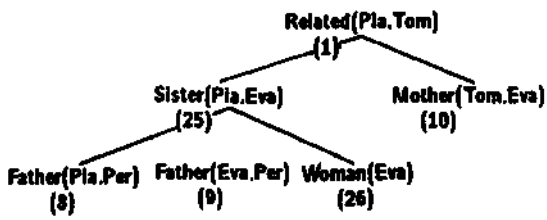

Figure 3. A proof tree for the formula Related(PlaTom)

In the presentation we start to explain why Pia and Tom are related in terms of the information in the nodes at the underlying level in the tree.

Rdated(Pia, Tom) <- Slster(Pia,Eva), Mother(Tom,Eva)

We can continue the presentation with explaining why Tom's mother is a sister of Pia.

Sister(Pia,Eva) <- Father(Pla,Per), Father (Eva, Per), Woman(Eva)

In order to make the explanation easier to read we present a clause using expressionforms connected to the predicates. For example the predicate Father(x,y) has the form "The father of x is y". In a corresponding way we write Ancestor(x,y) and Related(x,y) as "An ancestor of x is y" and "x and y are related".

AN instance of a rule, i. e. a node with its directly underlying nodes, we express "A since B" if the rule is A <- B. For example,

Ancestor(Ada,John) ← Father(Ada,John)

we express as "An ancestor of Ada is John since the father of Ada is John".

B. Filtering uninteresting values

In a proof tree, the arguments of the predicates have been in* stantiated to values that satisfy the formula in the root node. In an explanation we are not interested in all of the values. To present all of them is inconvenient.

A conjunction where two parts share values can be expressed in a form where the shared values are eliminated. To explain why Ada and Tom are related, see Figure 2, by naming every person in the tree is more unclear than if we rewrite the expression omitting some names. The presentation "Ada and Tom are related since the father of Ada is John and a grandfather of Eva is John and the mother of Tom is Eva" is less clear than "Ada and Tom are related since the father of Ada is a grandfather of the mother of Tom".

We can eliminate "uninteresting" values by reformulating predicates as functions and use these as values. For example, a conjunction of two predicates sharing a value can be reformulated as one of the predicates taking a representing function of the other as value. Analogously for conjunctions of three or more predicates. We illuminate it by an example

Father(Ada,x), Grandfather(Eva,x)

"The father of Ada is a grandfather of Eva". In this explanation, "a grandfather of Eva" is a representing function for the grandfather relation and is substituted for x in the father relation.

Our approach is to avoid including instantiations that don't appear in the current root. We want to explain the conclusion without including values appearing only in the precondition of a rule. Our example in previous section would become

Rdated(Pia,Tom) <- Sister(Pia,Eva), Mother(Tom,Eva)

"Pia and Tom are related since a sister of Pia is the mother of Tom", thus suppressing the value "Eva".

Reordering of conjunctions with common values can be necessary. A rule with a conjunction like

Cousin(Tom,Tim) ← Mother(Tom,Eva), Mother(Tim,Pia), Sister(Pia,Eva)

has to be reordered according to the common values. This rule becomes

Cousin(Tom,Tim) ← Mother(Tom,Eva), Sister(Pia,Eva), Mother(Tim,Pia)

The explanation is "Tom and Tim are cousins since the mother of Tom is a sister of the mother of Tim".

C. Same values at corresponding places

If there is a conjunction of instantiations of the same predicate and the common uninteresting values are corresponding arguments, the expression will be clearer if we point out this fact. We express

Father(Ada,x), Father(Per,x)

as "the father of Ada is the same as the father of Per".

D. Customised explanations

An important issue for obtaining clarity of explanations is to avoid presenting facts that already are known to the user.

General knowledge in onr domain could for example be what names are names of women and what are names of men. If we are going to present the following rule

Sister(Eva,Pia) ← Father(Eva,Per), Father(Pia,Per), Woman(Eva)

we can assume that the tact that Eva is a woman is general knowledge and suppress it. The explanation vill be "Eva is a sister of Pia since the father of Eva is the same as the father of Pia".

Specific knowledge possessed by the user would also be taken care of. It is possible that the user is not totally naive, some facts may be well known to her and it would only be negative to repeat them.

We assume that the user is a near friend of parts of our example family and knows that Eva is the mother of Tom. Then we present

Related(Pia, Tom) - Sister(Pia,Eva), Mother(Tom,Eva)

as "Pia and Tom are related since a sister of Pia is Eva".

V CONCLUSIONS

The transformation of proof trees is a way to view the problem of explaining the reasoning of an expert system. The principal ideas presented in this paper are fairly easy to implement. Furthermore the transformation for reducing chains of specifications of concepts seems to be generally applicable. The transformation for introducing a new name for a compilation of concepts depends on the problem domain.

The solution to the problem of presenting the proof trees described in this paper is simple. Further work to improve upon the naturalness of the presentations lead into the area of natural language processing.

REFERENCES

[1] Arasler, R. A.: *The Structure of the Merriam- Webtter Pocket Dictionary,* Technical Report TR-164, University of Texas, Austin, 1980.

[2] Carlsson, M. and K. M. Kahn: *LM-Prolog User Manual,* Upmail Technical Report 24, Department of Computing Science, Uppsala University, 1984.

[3| Parsaye, K.: *Database Management, Knowledge Ba$e Management and Expert System Development in PROLOG,* Proc. IJCAI-83, Karlsruhe, DDR, 1983.

[4] Pereira, F., L. Pereira, and D. Warren: *DEC-10 PROLOG users Guide,* Department of Artificial Intelligence, University of Edinburgh, 1977.

[5] Robinson, J. A.: *Logic: Form and Function,* Edinburgh University Press, Edinburgh, 1979.

|6] Swart out, W. R.: *Explaining and Juttifying Expert Consulting Program,* Proc. IJCAI-81, Toronto, Canada, 1981.

|7] Walker, A.: *PROLOG/EX1, An Inference Engine which Explain Both Yet and No Answers* Proc. IJCAI-83, Karlsruhe, DDR, 1983.

|8] Wallis, J. W. and H. Shortliffe: *Cuttomised Explanation Using Canal Knowledge,* in Buchanan and Shortliffe, "Rule-Based Expert Systems", Addison-Wesley, 1984.