SRI International

A PRODUCTION SYSTEM FOR AUTOMATIC DEDUCTION

Technical Note 148

July 1977

By: Nils J. Nilsson
Artificial Intelligence Center

This work was supported jointly by SRI International Project 6171, Office of Naval Research Contract No. NR049-405; and by the Stanford University Heuristic Programming Project, ARPA Order No. 2494.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

A new predicate calculus deduction system based on production rules is proposed. The system combines several developments in Artificial Intelligence and Automatic Theorem Proving research including the use of domain-specific inference rules and separate mechanisms for forward and backward reasoning. It has a clean separation between the data base, the production rules, and the control system. Goals and subgoals are maintained in an AND/OR tree structure. We introduce here a structure that is the dual of the AND/OR tree to represent assertions. The production rules modify these structures until they "connect" in a fashion that proves the goal theorem. Unlike some previous systems that used production rules, ours is not limited to rules in Horn Clause form. Unlike previous PLANNER-like systems, ours can handle the full range of predicate calculus expressions including those with quantified variables, disjunctions and negations.

CONTENTS

ILLUSTRATIONS

I Background

Logical deduction is a basic activity in many artificial intelligence (AI) systems. Specific applications in which deduction plays a major role include question-answering, program verification, mathematical theorem proving, and reasoning about both mundane and esoteric domains.

Of the several different approaches to deduction pursued by AI research, we might mention two extremes. In one (see for example, Hewitt, 1971), deduction procedures are based on more or less intuitive, ad hoc, and informal considerations. Such an approach derives its main advantage, namely efficiency, from the specialized, domain-dependent heuristics that can be tightly encoded in the system. The approach sometimes suffers, however, from excessive rigidity that frustrates the evolutionary development of systems. Most examples of designs based on this approach also exhibit deficient logical competence. (See Moore, 1975, for a discussion of these deficiencies and some remedial suggestions.)

At the other extreme, deduction is based on some formal logical system such as the predicate calculus. (See for example, Chang and Lee, 1973.) This approach confers the power of a well-developed logical formalism and is compatible with the evolutionary development of systems. When deductions are based on uniform (i.e., domain-independent) inference rules, however, the resulting systems are often too inefficient to be useful.

In this paper we shall propose a deduction system that enjoys most of the logical power of the formal systems without embracing their inefficient uniformity. It uses specialized, domain-dependent inference rules that are encoded as productions. As with most production systems,

it can easily be modified and extended by adding new production rules or by modifying old ones. The system is based on a synthesis of several ideas from various authors in artificial intelligence and automatic theorem proving. (The most immediate intellectual debts are to Bledsoe, 1977; Fikes and Hendrix, 1977; Hewitt, 1971; Kowalski, 1974a,b; Moore, 1975; and Sickel, 1976. Related work has been done by Nevins, 1975; Reiter, 1976; and Wilkins, 1974.)

Before describing the system in detail, we shall briefly mention some of the factors affecting its design. First we would like, in particular, to avoid the inefficiencies of resolution-based theorem proving systems. As has been observed by several authors, the "clause form" used by resolution theorem provers contributes to inefficiencies in two major ways: common sub-expressions in goals or axioms are "multiplied-out" into several different clauses each provoking its own separate but possibly redundant proof attempts; and conversion to clause form destroys possibly valuable heuristic information carried by the form of implicational statements among the axioms.

Second, we prefer a system in which the basic deduction steps have "common-sense" intuitive appeal. The process of resolution is, for some, difficult to relate to more familiar reasoning processes. This feature is especially important in those systems whose reasoning must be easily understood by users. Ease of understanding is also advantageous during system design and debugging. The processes of "natural deduction" more closely realize this goal than does resolution.

We want to be able to incorporate domain-specific knowledge into the system. This knowledge might consist of special inference rules and how to use them. In this regard, it is sometimes especially important, for efficiency, whether a deduction step proceeds forward (from the assertions toward the goal) or backward (from the goal toward the assertions). The domain expert, who participates in the design of the system, can often indicate the most efficient direction for each inference.

We are sufficiently impressed with the advantages of production systems (Davis and King, 1977) that we would like to model our design on that paradigm. Previous production system designs for deduction systems, however, had somewhat limited logical power. (An example is the restriction to Horn clauses in Kowalski, 1974b.) We want our system to be able to employ the full expressive power of the first-order predicate calculus, including the ability to reason with disjunctive assertions, negations, and quantification of variables. Certainly our system should be sound (i.e., it should not prove invalid expressions). With regard to completeness (i.e., being able to prove any theorem), we are less doctrinaire. We insist only that it behave reasonably according to criteria specific to the domain of application. Any incompletenesses that cannot be tolerated must be repairable by evolutionary changes to the system.

We also note that the production system paradigm permits a convenient separation between the "logical knowledge" embodied in the assertions and in the production rules and the use of this knowledge by a control system. Changes can be made to each component separately, depending on whether the logic or its control is to be changed. In particular we envision a more domain-specific control system than the simple, uniform interpreter used by most resolution systems.

We want the methods used by our system to be easily extendible to representations that are "richer" than the usual implementations of predicate calculus data bases. We have in mind, specifically, semantic networks (Fikes and Hendrix, 1977) and "structured-object" representations (Bobrow and Winograd, 1977) with various built-in features for indexing, taxonomic reasoning, and sorting of arguments according to type.

Lastly, we attach great importance to the "esthetic appeal" of the system. It should have a clear design, and it should itself be a clear statement of a useful synthesis of some of the best ideas in automatic theorem proving. We will gladly trade some efficiency for enhanced clarity.

II Overview of the System

The classical model of theorem proving in the predicate calculus involves three major components. First, there is a set of axioms or assertions that express information about the domain of application. For geometry, for example, these would be the fundamental postulates plus whatever other theorems we want to start with. (It is neither necessary nor desirable to limit the assertions to some primitive or minimal set.) Second, there are domain-independent, uniform rules of inference (such as resolution, modus ponens) that can be used to derive new assertions from existing ones. Finally, there is a conjectured theorem, or goal, to be proved. A proof consists of a sequence of inference rule applications ending with one that produces the goal.

AI research has produced an important deviation from this approach. The assertions are divided into two distinct sets: facts and rules. Facts are specific statements about the particular problem at hand. For example, "Triangle ABC is a right triangle" would be expressed as a fact. Rules are general statements, usually involving implications or quantified variables. For example, "The base angles of an isoceles triangle are equal" would be expressed as a rule. Rules are used in combination with facts to produce derived facts. One could think of them as specialized, domain-dependent inference rules.

This distinction can be further explained by a simple example. In the classical approach, from the two assertions A and A=>B we could derive the assertion B by modus ponens. In the AI approach, from the fact A we could derive the fact B by using the special rule A=>B. The distinction between facts and rules is an important part of our deduction system.

The rules will be used as production rules. They will be invoked by a pattern matching process. Some will be used only in a forward direction for converting facts to derived facts; others will be used only in a backward direction for converting goals to subgoals. The developing sets of facts and goals will be represented by separate tree structures. Goals will be represented in an AND/OR goal tree, and facts will be represented in a newly proposed structure that we shall call a fact tree. Rules are employed until the fact tree joins the goal tree in an appropriate manner. The entire process will be under the supervision of a control strategy that decides which applicable rule should be employed at any stage. We shall not propose any specific control strategies in this paper but shall merely point out that the designer has the freedom to use any domain-specific information whatsoever in the control system.

Several designs of this general sort have been proposed (see, for example, Kowalski, 1974b), but most of them have had restrictions on the kinds of logical expressions that could be accommodated. Although AND/OR goal trees have been used before, the notion of a fact tree, dual to the goal tree, allows some interesting correspondences, such as that between "reasoning by cases" and dealing with conjunctive goals, for example.

We shall first explain the system using the propositional calculus and then indicate how we deal with quantification.

III Goal Trees and Fact Trees

A. Conversion of Facts and Goals to Standard Form

In this section we shall introduce the tree structures used to represent collections of facts and goals. Facts and goals can be any expressions of the predicate calculus (propositional calculus for this section). We do convert them, though, into a standard form. Implications are changed to disjunctions by using the equivalence between (A=>B) and (~A & B). Negations are "moved in" by using the equivalences between ~(A & B) and (~A v ~B) and between ~(A v B) and (~A & ~B). Repeated negations are eliminated by using the equivalence between ~~A and A. Once a goal or fact expression has been converted to this standard form, it will consist of a conjunctive/disjunctive combination of literals. For example, the expression ~H => [G & ~(F & ~B)] would be converted to H v [G & (~F v B)].

Ordinarily the domain expert, who is providing us with facts and rules, would not give us any facts containing implications. These would be given as rules. Also, goal statements would not ordinarily contain implications. (The "hypotheses" of a theorem to be proved would ordinarily be represented as facts, the conclusion as a goal.) We may have disjunctive facts, however. The distinction between ~A v B as a fact and A=>B as a rule is simply this: as a fact, the domain expert is simply saying that either ~A or B is true and he doesn't know which. As a rule, the domain expert is saying that A is useful for proving B. The system makes quite different use of the two forms.

Also note that our conversion of facts and goals to standard form is not the same as conversion to clause form in resolution. In general, clause form involves more expressions. Our standard form is very close to the form of the original expressions.

B. AND/OR Goal Trees

For a goal of the form (A1 & ... & An) we must prove all of the goals A1 and ... and An. For a goal of the form (A1 v ... v An), it suffices to prove one of the goals A1 or ... or An. Structures called AND/OR goal trees (Nilsson, 1971) are used in many AI systems to represent collections of subgoals and their relation to the main goal.

Any goal expression that has been converted to our standard form can be represented by an AND/OR goal tree having single literals at its tips. For example, the expression H v [G & (~F v B)] would be represented by the AND/OR tree shown in Figure 1.

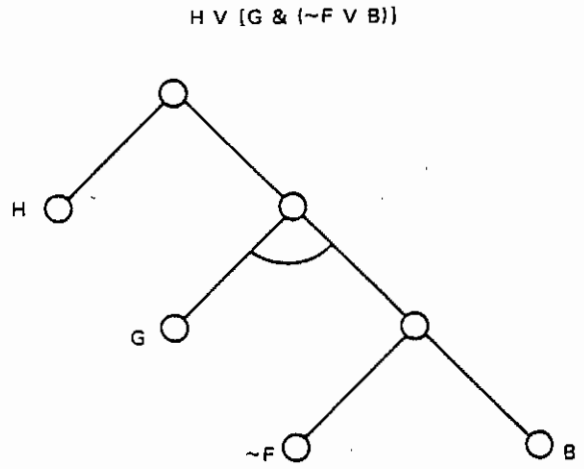
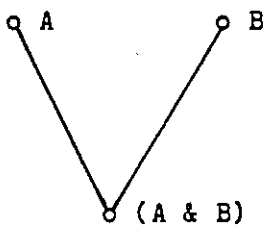


Figure 1. An AND/OR Tree

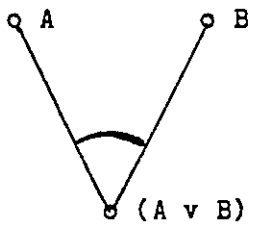
In AND/OR goal trees, nodes (such as node G in Figure 1) whose incoming branches are connected together by an arc are called AND nodes. If their incoming edges are not so connected, the nodes are called OR nodes.

C. OR/AND Fact Trees

It is convenient to represent the facts to be used in a deduction by a structure that is the dual of the AND/OR goal tree. We shall call this dual structure an OR/AND fact tree. The notational conventions for the fact tree are the reverse of those for the goal tree. We shall represent conjunctive facts by a structure consisting of AND-nodes, thus:



Disjunctive facts will be represented by a structure consisting of OR-nodes, thus:



Note that for fact trees the arc connecting the branches is used with disjunctions rather than with conjunctions. Also, fact trees are drawn "upside down" compared with goal trees.

Any fact expression that has been converted to our standard form can be represented by an OR/AND fact tree having single literals at its tips. For example, the expression A & [B v (C & E)] & D would be represented by the OR/AND tree shown in Figure 2.

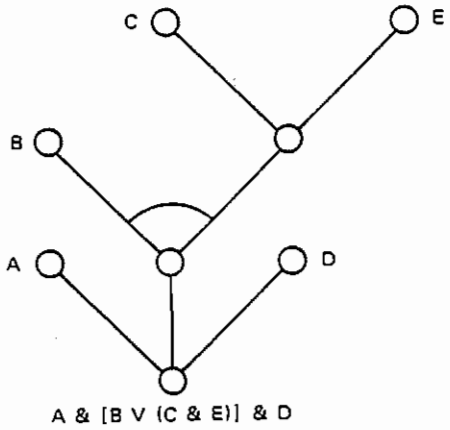


Figure 2. An OR/AND Tree

The reason that we use opposite conventions to denote disjunctions and conjunctions in fact and in goal trees has to do with the nature of their duality. We shall see later that these opposite conventions will simplify some definitions.

(For both goals and facts we will represent repeated instances of the same literal by different nodes. This practice allows us to use trees instead of graphs.)

D. Connecting Fact and Goal Trees: Proof Termination

The problem of making a deduction is to "connect" the goal tree to the fact tree. This will be done mainly by using rules to extend the trees. We will also admit a process that allows a type of tree pruning. But before moving on to discuss these subjects, let us first define precisely what is meant by "connecting" a goal tree to a fact tree.

The connections between fact and goal trees are at nodes labeled by the same literal. In the original trees, such nodes must be tip nodes. After the trees are extended by rule applications, the connections might occur at any node labeled by a single literal. We shall call nodes labeled by a single literal literal nodes. After all such connections are made, we still have the problem of determining whether or not the expression at the root of the goal tree logically follows from the expression at the root of the fact tree. Our proof procedure will terminate when this determination can be made (or when we can conclude that it can never be made). The termination condition is a simple generalization of the condition for determining whether the root node of an AND/OR tree is "solved" (Nilsson, 1971, p.89.) The termination condition is based on a simple symmetric relationship, called CANCEL, between a fact node and a goal node. In the definition of CANCEL we use the phrase arced nodes to refer both to AND nodes in goal trees and to OR nodes in fact trees. If CANCEL holds for two nodes n and m, we shall say that n and m CANCEL each other. CANCEL is defined recursively as follows:

Two nodes n and m CANCEL each other [that is, CANCEL(n, m) holds] if one of (n, m) is a fact node and the other a goal node, and

0) if n and m are labeled by the same literal,

or 1) if n has arced successors, $\{s_i\}$, such that CANCEL(s_i, m) holds for all of them,

or 2) if n has unarced successors, $\{s_i\}$, such that CANCEL(s_i, m) holds for at least one of them.

Our definition of CANCEL supports a simple termination checking process that starts at nodes labeled by the same literal and propagates the CANCEL relation toward the roots. The proof procedure terminates successfully whenever we can show that the root of the fact tree and the root of the goal tree CANCEL each other.

Note, in particular, that our proof procedure treats conjunctive goal nodes correctly. Each conjunct must be proved before the parent is proved. Disjunctive fact nodes are treated in a dual manner. In order to use a disjunct in a proof, we must be able to prove the same result using each of the other disjuncts in turn. This process is sometimes called "reasoning by cases."

The reader might like to establish termination for the goal-fact tree pairs of Figure 3.

E. Transferring Between Fact and Goal Trees: Checking for Contradictory Facts and Tautological Goals

Being cancelled by the fact tree is only one of the ways that a goal can be satisfied. We can also show that a goal is true by reducing it to a tautology. Recognizing some tautologies in goals can be accomplished by a simple extension of the termination process just described. We shall introduce our discussion of this extension by describing how nodes could be transferred between the goal and fact trees.

Suppose from a given set, F, of facts, we must prove a disjunctive expression of the form G1 v G2, where G1 and G2 can be any expressions. In logical notation we can represent this problem by the expression:

$$F \vdash G1 \vee G2$$

(The expression "A ⊢ B" means "B logically follows from A".) Now we can invoke what we shall call here the law of transfer to convert this problem into either of the following ones:

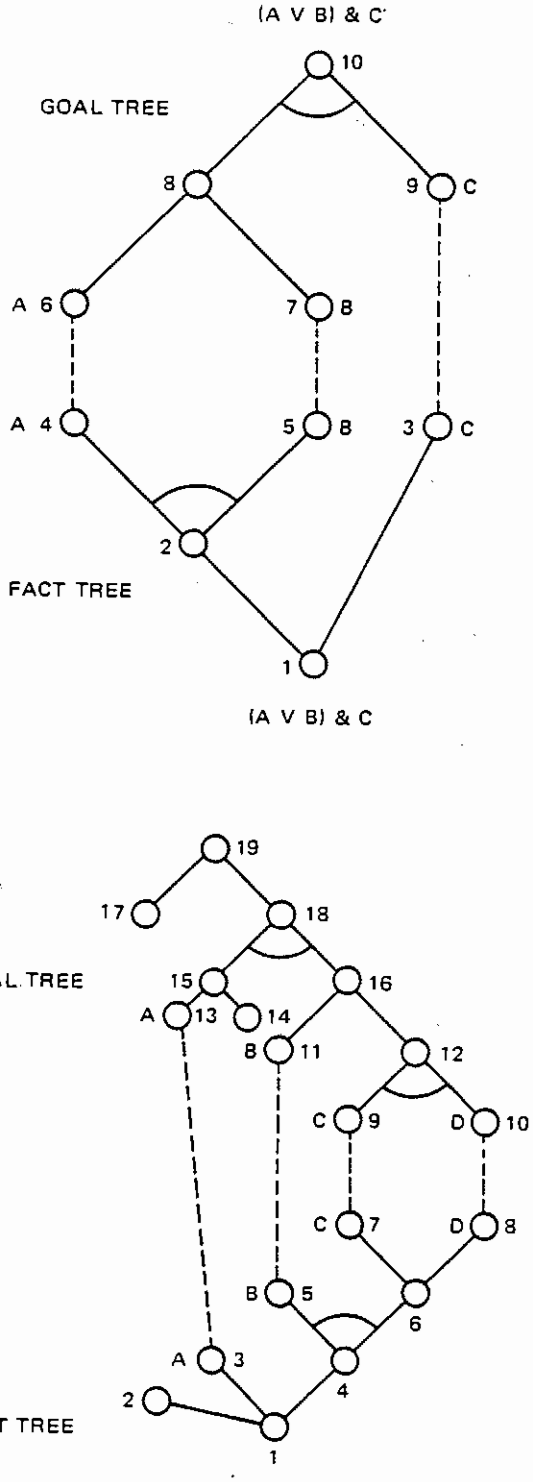


Figure 3. Example Goal-Fact Tree Pairs

$$(F \;\&\; \neg G1) \vdash G2$$

or

$$(F \;\&\; \neg G2) \vdash G1$$

That is, one of the goal disjuncts can be negated and transferred to the fact tree, where it is conjunctively associated with the other facts. For example, the tautological goal A v ¬A can be represented as a goal A and a fact A. The termination check now reveals that the two root nodes CANCEL, so we have a proof.

In a dual fashion, we could recognize contradictory facts by transferring one of the conjuncts of a fact conjunction over to the goal tree. To do so, we negate the fact tree to be transferred and disjunctively append it to the goal tree. In either case, when a tree is negated prior to transfer we need only negate the literal nodes; the reversed conventions about arced and unarced nodes automatically provide the correct interpretations when the tree is transferred.

But we really do not have to perform these transfer operations explicitly in order to deal with tautological goals and contradictory facts. Instead, we can allow oppositely signed literals of the same tree to CANCEL each other and then use the rest of the definition of CANCEL to propagate CANCELled nodes toward the roots. In this manner, the definition of CANCEL is extended to apply to nodes in the same tree. In applying the definition, we need to label the root of the goal tree and the root of the fact tree with the same identifier. Termination can now occur if the root of one tree CANCELs either itself or the root of the other.

Note that proof strategies based on proof by contradiction (refutation) involve transferring the entire negated goal tree to a conjunctive branch of the fact tree. Also in some theorem proving systems (e.g., Fikes and Hendrix, 1977), disjunctive goals are split into alternative subproblems in which the negation of the sibling goals can be <u>locally</u> added to the fact base for each subproblem. This

strategy corresponds to a local transfer process. For our purposes, with our extended definition of CANCEL, it doesn't really matter whether we leave the fact and goal trees as originally given or whether we perform explicit transfer operations.

The reader will note that computing CANCEL relations within the same tree corresponds to a type of resolution process. General resolution of facts (or goals) is not so simply accomplished, however.

The transfer operation is one way of transforming a given problem into a set of equivalent ones. Another type of transformation is used in some systems (such as that of Fikes and Hendrix, 1977) for dealing with disjunctive facts. Suppose our problem is to prove the expression G from the expression F & (F1 v F2), where G, F, F1, and F2 can be any expressions. We can convert this problem into either of the following pairs of problems:

$$F \vdash \neg F1$$

and

$$F \,\&\, F2 \vdash G$$

or

$$F \vdash \neg F2$$

and

$$F \,\&\, F1 \vdash G$$

That is, we first prove one disjunct false and then use the other to prove G. But the subproblem $F \vdash \neg F1$ corresponds to a transfer operation that really does not need to be performed by our system with its extended definition of CANCEL. The other subproblem, $F \,\&\, F2 \vdash G$, evolves naturally in our system as a result of the recursive definition of CANCEL. There is a dual explanation that can be given for dealing with conjunctive goals.

Now that we are well equipped to recognize when our proof process can be terminated, we can begin discussing how rules are used to extend the fact and goal trees. We first discuss the form of the rules.

IV Rules

A. Rule Forms

We allow two basic types of rules. One, called an OPERATOR, is used to extend the fact tree. OPERATORs permit the system to reason in a forward direction. The other type, called a REDUCER, is used to extend the goal tree. REDUCERs permit the system to reason in a backward direction. OPERATORs and REDUCERs are roughly analogous to the antecedent and consequent theorems, respectively, used in the PLANNER language (Hewitt, 1971). As in PLANNER, OPERATORs and REDUCERs are invoked by a pattern matching process. Each has a distinguished literal, called the pattern, that is used to match a corresponding literal in the fact or goal tree.

The basic form of an OPERATOR is

$$\underline{A} \Rightarrow EXP$$

where EXP is any predicate calculus expression, and where an underline beneath a literal indicates that this literal is the pattern. Thus OPERATORs are always implications whose antecedent consists of a single literal that is the pattern. If this pattern matches a literal in the fact tree, then the fact tree can be extended at this node by sprouting a descendant OR/AND tree representation of EXP.

The basic form of a REDUCER is

$$EXP \Rightarrow \underline{A}$$

where EXP can be any predicate calculus expression. Again, the pattern is underlined. REDUCERs are always implications whose consequent consists of a single literal that is the pattern. If this pattern

matches a literal in the goal tree, then the goal tree can be extended at this node by sprouting a descendant AND/OR tree representation of EXP.

It is only for reasons of simplicity that we constrain our rules to have single-literal patterns. Useful variants of our system can be devised in which tree structures more complex than a literal node are used as patterns. Of course the matching process for these more complex structures would be correspondingly more tedious. Also, later we shall discuss a technique for achieving the effect of more complex OPERATOR antecedents by allowing OPERATOR consequents to contain rules.

It has been argued by Moore (1975) that the contrapositive of a REDUCER should be expressed as an OPERATOR and vice versa. Thus if A => EXP is useful as an OPERATOR, its contrapositive form, namely ~(EXP) => ~A, would also be useful as a REDUCER. Our system will automatically add these contrapositive forms for every rule entered into the system. (Note that after negating an expression, we must move the negation in.)

The existence of the contrapositive forms of rules means that it does not make any difference to our system whether goals and facts are kept on their own side of the line or transferred. If a goal invokes a given REDUCER, then the fact resulting from transferring that goal would invoke the corresponding OPERATOR. Thus, it is really unimportant whether we maintain goals and facts as given or whether we negate all of the goals, for example, add them to the fact base, and look for a refutation. We shall adopt the convention of maintaining goals and facts as given, mainly to ease the process of explaining the behavior of the system to the user.

B. Use of Rules

The basic cycle of operation of our deduction system can be informally described by the following steps:

(1) Initialize the goal and fact trees to the given expressions.

(2) If the termination check succeeds, exit.

(3) Use the domain-specific control strategy to select one of the literal nodes and an OPERATOR or REDUCER whose pattern matches this literal node.

(4) Apply the selected rule, extend the goal or fact tree, and go to (2).

Rule application is thus a pattern-directed process having effects on data bases (fact and goal trees). The system design can thus reasonably be described as a "production system" in the sense in which that term is generally used in AI research. In the next section we shall show how the system might work on some propositional calculus examples.

V Propositional Calculus Examples

As a first example of how the system works, suppose we want to prove {H v [G & (B v ~F)]} from the expression {A & [B v (C & E)] & D}. We are given the REDUCERs

R1: C & E => ~F

and

R2: D => G

From these, we construct the corresponding OPERATORs

O1: F => ~C v ~E

and

O2: ~G => ~D

The fact and goal expressions are already in standard form. We show their tree representations in Figure 4.

In Figure 4, we use capital letters next to the tip nodes for literals, and we use numerals to label the nodes themselves for later reference. We connect matching nodes by dashed lines. From Figure 4 we see that nodes 9 and 5 CANCEL. Using the definition of CANCEL, we note that nodes 12 and 5 CANCEL. It will be helpful to keep a list of the CANCELled pairs. This list is also shown in Figure 4.

None of the OPERATORs is applicable, but both of the REDUCERs are. Suppose we apply R2 first adding node 15 to the goal tree. After this cycle, the situation is as depicted in Figure 5. We have updated the list of CANCELled pairs. At this stage we have essentially reasoned only about one case of the disjunct B v (C & E).

Now we apply the only remaining applicable rule, R1. The resulting trees are shown in Figure 6. The list of CANCELled pairs includes (1, 1), so we terminate successfully.

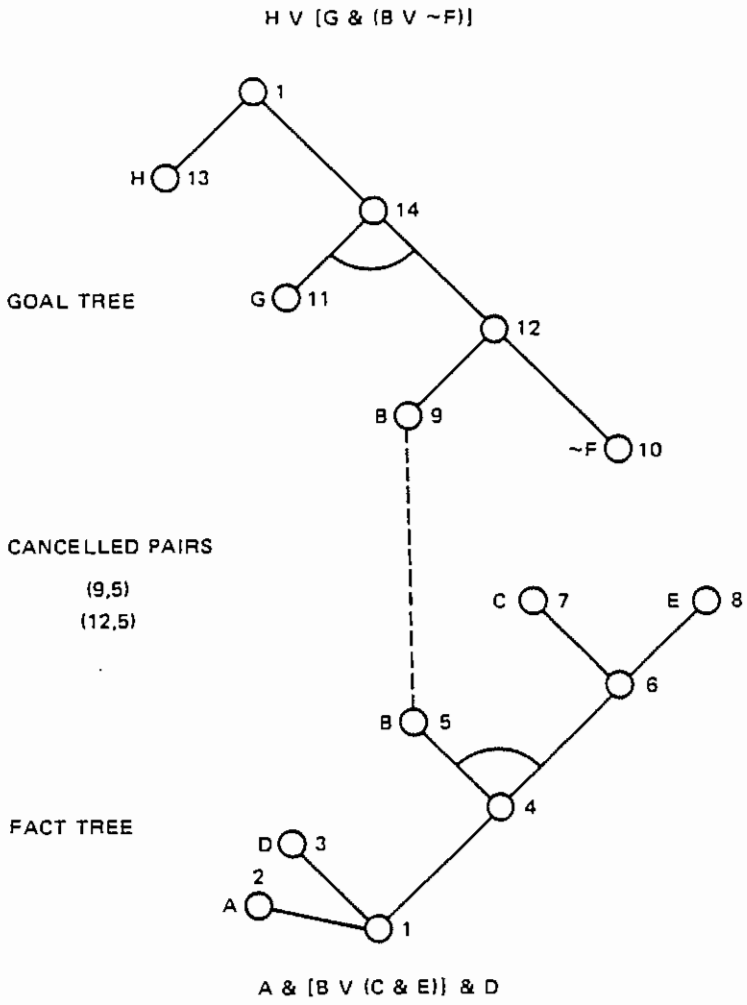


Figure 4. Example Fact and Goal Trees

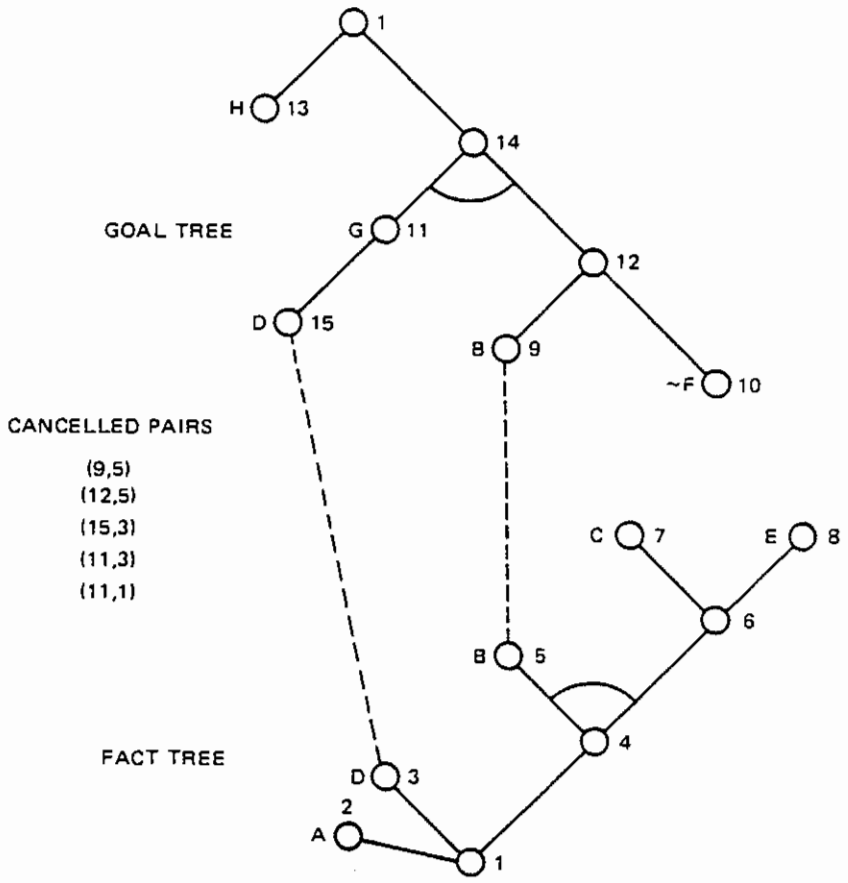


Figure 5. An Intermediate Stage of a Proof

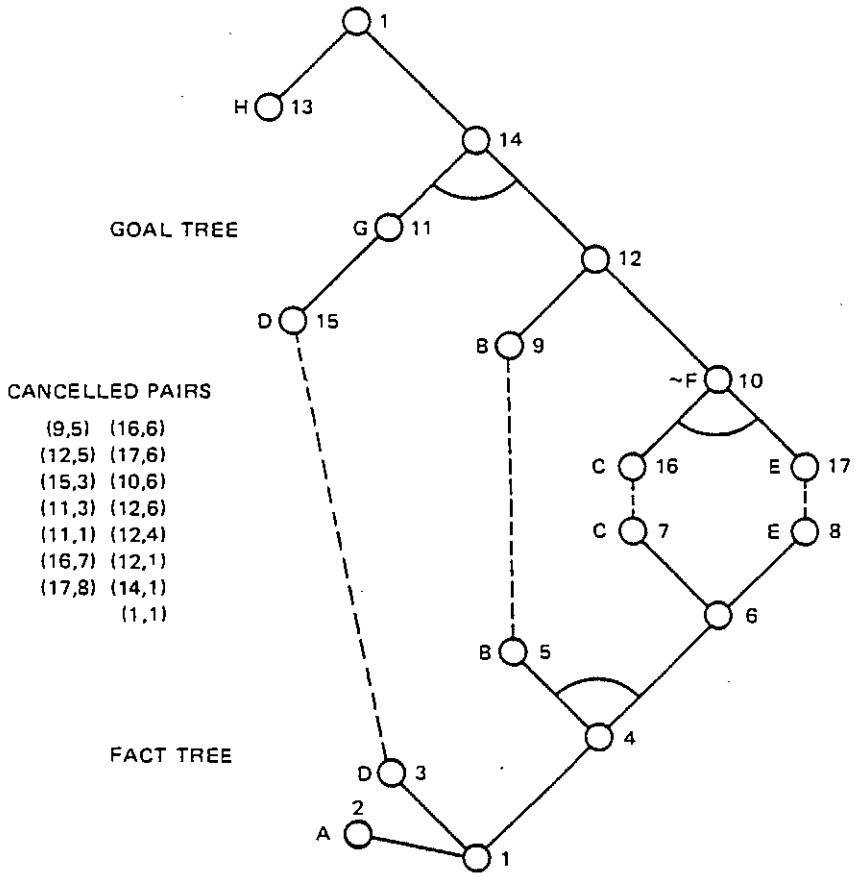


Figure 6. The Final Stage of a Proof

For our second example, we will illustrate how CANCELling nodes in the same tree can be used to obtain a proof. Suppose our goal is to prove B v C and we are given the following OPERATORS:

O1: ~C => D

and

O2: D => B

We will assume we have no facts. The problem would be straightforward if we were to split B v C into two disjunctive subgoals such that while working on subgoal B we could assume (locally) the fact ~C. This fact would combine with OPERATOR O1 to produce fact D, which in turn would combine with O2 to produce B.

Our system does not make these local assumptions, but the use of contrapositive rules and CANCELling nodes within a tree accomplishes the same thing. The contrapositives of our OPERATORs are the REDUCERs

R1: ~D => C

R2: ~B => ~D

Now R1 can be used on subgoal C to produce subgoal ~D. REDUCER R2 can be used on this subgoal to produce subgoal ~B. This subgoal CANCELs the earlier subgoal B, with the ultimate result that the root CANCELs itself. (The reader may want to verify this with the help of a diagram.)

A case of special interest occurs when a rule application produces a literal node that CANCELs one of its own literal node ancestors. This corresponds to a special case of ancestor resolution. Propagation of the CANCEL relation may then ultimately result in a node CANCELling itself. For purposes of CANCEL propagation, any node that CANCELs itself can be regarded as being CANCELled by the root node of the opposite tree. (Self-CANCELling goal nodes correspond to tautologies, and self-CANCELling fact nodes correspond to contradictions.)

Another interesting case occurs when a pair of sibling nodes CANCEL each other. If the siblings are unarced, then obviously their parent CANCELs itself, and we have the previous case. If the siblings are arced, the parent node cannot possibly appear in a proof, so it is eliminated from the tree. If this parent node is itself arced, its parent is eliminated, and so on. If a root node is ever eliminated, the entire proof attempt fails.

There are some problems (even in propositional calculus) that our system cannot solve. Our tolerant attitude towards this sort of incompetence is explained as follows. We are relying on the domain expert to provide us guidance about which rules are useful and in which direction they should be used. Hopefully, his expertise enhances the efficiency of the system. But dependence on the expert carries a price: gaps in his expertise decrease the competence of the system. There are some simple examples that illustrate this point.

Suppose the goal is B v C and the OPERATORs are $\underline{A}$ => B and ~$\underline{A}$=> C. Unfortunately these rules work in the wrong direction; if they were REDUCERs instead, the goal would be easy to prove. (The contrapositive REDUCERs of the given OPERATORs are of no help.) One way around this difficulty is to use the implicit fact A v ~A as if it were explicitly in the fact tree. The given OPERATORs could then be used to obtain a proof. Obviously this strategy of assuming all tautologies to be explicit facts would defeat our attempts at efficient operation, because it would allow every OPERATOR to be used in every problem. Another possible approach to this problem would be to analyze the OPERATORs to look for pairs having oppositely signed patterns. The disjunction of their consequents could then be added to the fact tree. (A dual approach could be used with REDUCERs.) But this catches only first-level difficulties. The main point is that to increase efficiency we are using the rules only in a given direction, and we are not allowing the rules to interact among themselves; therefore the domain expert must pose the problem in such a way that the system can still find a solution even with these restrictions.

Another troublemaker involves the goal A => C (that is, ~A v C) and the facts A => B (that is, ~A v B) and B => C (that is, ~B v C). Suppose there are no OPERATORs and no REDUCERs. Since facts cannot interact among themselves, we cannot produce a proof. Again the domain expert has failed us in not structuring the problem correctly.

Our attitude toward these problems is to avoid the easy but inefficient approach of allowing intrafact and intrarule inferences. That is precisely what our system is trying to escape. Instead, we will exploit the inherent modularity of the system to correct inadequacies in the rule and fact base as they are discovered.

VI Extension to Quantification

A. Overview

The system we have described for propositional calculus can be easily modified to deal with quantified variables in expressions. The modifications involve: (1) replacing certain variables by Skolem functions, (2) using unification during CANCEL operations, and (3) associating a substitution with each CANCEL relation. In this section, we shall discuss these modifications and present some examples.

B. Skolemization

Fact expressions receive the same initial preparation as for the propositional calculus case; implications are eliminated, and negations are moved in. We use the equivalences between

~(EXISTS X) F(x) and (FORALL x)[~F(x)]

and between

~(FORALL x) F(x) and (EXISTS x)[~F(x)]

to move negations in through quantifiers. Next we replace all instances of existentially quantified variables by Skolem functions of those universally quantified variables in whose scopes they reside. Next we drop all quantifiers and henceforward adopt the convention (for facts) that all variables are universally quantified. When a fact expression is in this form, it can be represented as an OR/AND fact tree. The literals at the tip nodes may contain variables, of course.

Goal expressions also receive the same initial treatment. Skolemization, however, is different. In goal expressions, we replace all instances of universally quantified variables by Skolem functions of

those existentially quantified variables in whose scopes they reside. (Recall that goals can be regarded as negated facts and that negated existential quantifiers are equivalent to universal ones. Thus, it shouldn't be surprising that Skolemization of goal expressions uses conventions dual to those of Skolemization of facts. If we are able to prove some expression F(a) where a is a constant different from those used in the facts and rules--that is, it is a Skolem constant--then we can deduce (FORALL x) F(x) by universal generalization. Skolemization of universally quantified variables in goals can thus be regarded as using the rule of universal generalization in reverse.)

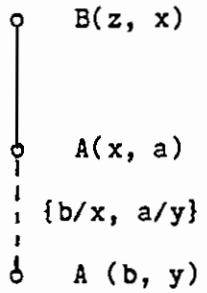
After elimination of the universally quantified variables, we can drop all of the quantifiers and adopt the convention that all variables (in goals) are existentially quantified. When a goal expression has been thus prepared, it can be represented as an AND/OR tree.

Skolemization of variables in rules is just slightly more complicated. (We allow rules of the same general form as in the propositional calculus case, however, they can have arbitrary quantification.) Quantifier scopes in rules can be of three types: the scope can be the entire implication or limited to either the antecedent or the consequent. We Skolemize any existential whose scope is either the entire implication or its consequent. We Skolemize any universal whose scope is limited to the antecedent of the implication.

After Skolemization, we can drop the quantifiers, and the variables will "behave correctly." That is, when an OPERATOR is used, those variables occurring in the new fact nodes will have assumed universal quantification, and similarly for REDUCERS.

C. Use of Rules

When a rule is used to extend the fact or goal tree, its pattern must be unifiable with the literal at the node from which it extends. It will be convenient to represent this matching process by an explicit edge of the tree and associate the most general unifier (mgu) with this edge. Thus, when the OPERATOR A(x, a) => B(z, x) is used to extend the fact node A(b, y), we produce the following structure:



We represent "match edges" in trees by dashed lines and label them by the mgu obtained in unification. (Note that we do not apply to the rule consequent the substitution obtained by unifying with the antecedent. The equivalent of this operation will be incorporated into our new definition of CANCEL.) When a match edge is added to the tree, the node associated with the rule pattern is always an "unarced" node.

The variables that occur in the fact and goal trees should be kept standardized apart. This means that any variables that are common across goal disjuncts or fact conjuncts can be given different names. For example, the goal expression A(x) v B(x) can be changed to A(x) v B(y). The fact expression C(x) & D(x) can be changed to C(x) & D(y).

D. Extending the Definition of CANCEL

We must extend the definition of CANCEL so that it takes into account the substitutions obtained during matching. For example, in propagating CANCEL relations involving arced nodes to the parent of the arced nodes, we must make sure that the substitutions for variables at these nodes are "consistent." The necessary elaboration involves associating a substitution with each CANCEL relation and modifying the definition of CANCEL to check for substitution consistency.

In the definition, we use the concept of a unifying composition (uc). The unifying composition of two substitutions, u_1 and u_2, is a most general substitution, u, satisfying

$(Lu_1)u = (Lu)u_1 = Lu = (Lu_2)u = (Lu)u_2$

for an arbitrary literal L. (The expression Lu denotes the result of applying substitution u to literal L.) If no such u exists, then the uc is undefined. The uc of a set of substitutions $\{u_1, \ldots, u_n\}$ is the uc of any member, u_1, of the set with the uc of the rest of the set $\{u_2, \ldots, u_n\}$. The substitutions in a set are inconsistent if the set has no uc.

The following are examples of unifying compositions (Sickel, 1976):

u_1	u_2	u
{a/x}	{b/x}	undefined
{x/y}	{y/z}	{x/y, x/z}
{f(z)/x}	{f(a)/x}	{f(a)/x, a/z}
{x/y, x/z}	{a/z}	{a/x, a/y, a/z}
{s}	{ }	{s}

The new definition of CANCEL is that nodes n and m CANCEL

(1.1) If n and m are literal nodes of different trees and if the corresponding literals are unifiable. In this case, we associate the mgu with CANCEL(n, m),

or (1.2) If n and m are literal nodes of the same tree and if one of the corresponding literals unifies with the negation of the other. In this case, we associate the mgu with CANCEL(n, m),

or (2) If n has arced successors, $\{s_i\}$, such that CANCEL(s_i, m) holds for all of them, and the unifying composition (uc) of the set of substitutions associated with the individual CANCELs exists. In this case, we associate the uc with CANCEL(n, m),

or (3) If n has unarced successors, $\{s_i\}$, such that CANCEL holds for at least one of them and the uc of the edge substitution and the substitution associated with the individual CANCEL exists. In this case, we associate the uc with CANCEL(n, m).

The consistency requirement on the individual substitutions in part 2 of our definition for CANCEL ensures proper propagation of CANCEL through arced nodes. The consistency requirement in part 3 of our definition ensures that the proper instances of matched rules are used to extend the trees. (In using part 3 of the CANCEL definition, we assume that the empty substitution is associated with nonmatch edges.)

E. An Example

Several important mechanisms are implicit in our definition of CANCEL. These can best be understood by detailed examination of an example. The example is illustrated in graphical form in Figure 7. The fact expression is shown at the bottom of the figure in OR/AND tree form; the variable "s" is assumed to have universal quantification. The goal expression is shown at the top in AND/OR tree form; the variable "x" is assumed to have existential quantification. The rules are simply shown as unconnected pieces of graph near the tip nodes where they ultimately will be used. All nodes in the graph are given a number. (In this example, it happens that the rules will be used at most once, so we prenumber their "nodes" for convenience.) Rule patterns are indicated by the usual convention. Lower-case letters near the beginning of the alphabet (for example, a, b, c, ...) denote constants,

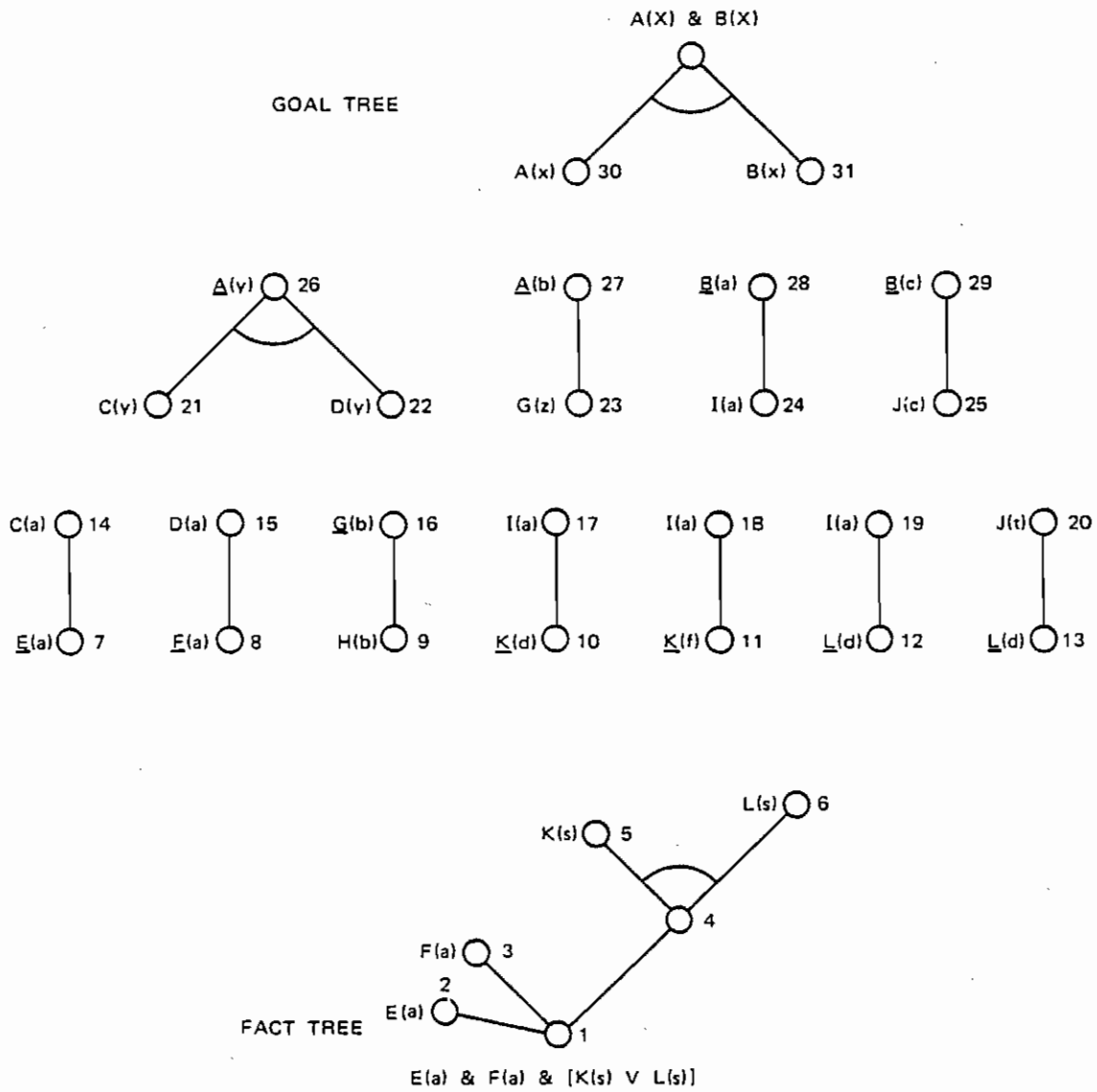


Figure 7. An Example with Variables

and lower-case letters near the end of the alphabet (for example, ... x, y, z) denote variables. All variables have been standardized apart. We have not shown the contrapositive forms of the given rules since they won't be used in this example.

At the outset we notice that there are several applicable rules. Since we have not yet advocated any particular control strategy, we shall trace through this example in an order that best illustrates the points we wish to make.

First, let us match node 30 with the REDUCER node 26. The mgu is {y/x}. (When a variable is substituted for another variable, we adopt the convention of substituting the variable about to be added to the tree for the one already in the tree.) The goal tree that results after this match is shown in Figure 8. No CANCEL relations are established yet, but we do associate {y/x} with the match edge between nodes 26 and 30. Let's next match goal nodes against REDUCER nodes 27, 28, and 29. The goal tree will now be as shown in Figure 9.

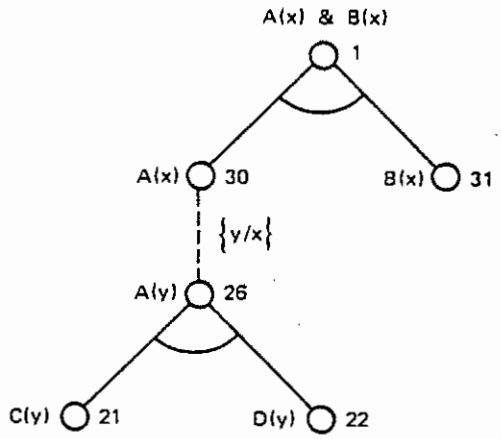


Figure 8. The Goal Tree After Applying a REDUCER

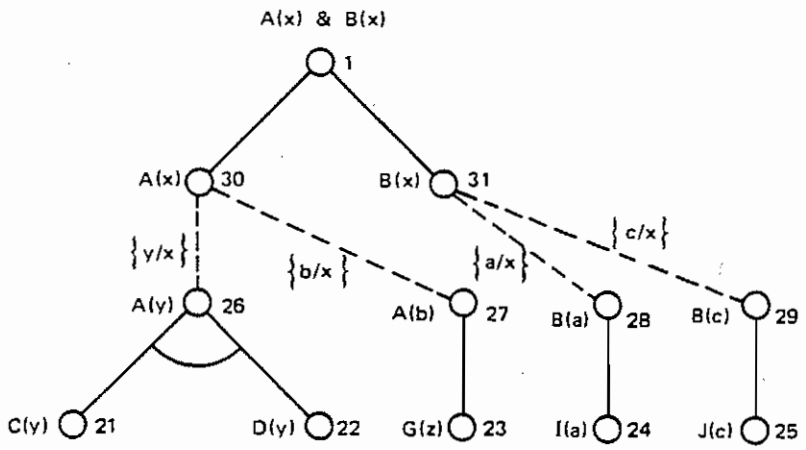


Figure 9. The Goal Tree After Applying Four REDUCERs

We could continue to apply REDUCERs or OPERATORs until some nodes could be CANCELled, but at this stage it is possible to predict that certain later attempts at CANCELling will fail. Notice in Figure 9 that any attempt to propagate a CANCEL relation up through node 27 to node 30 will involve the substitution {b/x}. But this substitution is inconsistent with all of the substitutions shown below node 31. If we have exhausted all possible matches to node 31, then we know that the substitution {b/x} at node 30 can never occur in a proof because only a or c can be substituted for x. Such an occurrence would correspond to a violation of horizontal consistency (Sickel, 1976). Thus, there can be no unifying composition of a CANCEL relation propagated up through node 27 with a substitution for any CANCEL relation in which node 31 participates. At this stage, we can prune node 27 (and, with it, node 23) from the goal tree and save ourselves the effort of attempts to prove G(z).

Quite analogous considerations would allow us to prune node 11 (and, with it, node 18) from the fact tree after we have matched against the OPERATOR nodes 10, 11, 12, and 13. After all of this, the fact tree is as shown in Figure 10.

Now, we can do some CANCELling between the literal nodes of the fact and goal trees. The following CANCELled pairs can be established:

(17, 24) { }
(19, 24) { }

and

(20, 25) {c/t}

By using the CANCEL definition we can for example, determine next the following CANCELled pairs:

(29, 20) {c/t}
(31, 20) {c/x, c/t}
(31, 13) {c/x, c/t}
(31, 6) {c/x, c/t, d/s}

The associated substitutions are merely unifying compositions between edge substitutions and previous CANCEL substitutions. If a uc did not exist for a proposed CANCEL relation, then we could not establish this relation. Such an occurrence corresponds to a violation of vertical consistency (Sickel, 1976).

We can also obtain another CANCEL relation between nodes (31, 6) by a different route and thus with a different substitution, namely {d/s, a/x}. We represent both of these substitutions by repeated instances of CANCEL(31,6).

The other CANCEL relation of interest that can be established at this stage is between the node pair (5, 31) with associated substitution {d/s, a/x}. To summarize, the CANCEL relations of interest (i.e., those between nodes closest to the roots of the trees) are now:

(6, 31) {c/x, c/t, d/s}
(6, 31) {d/s, a/x}
(5, 31) {d/s, a/x}

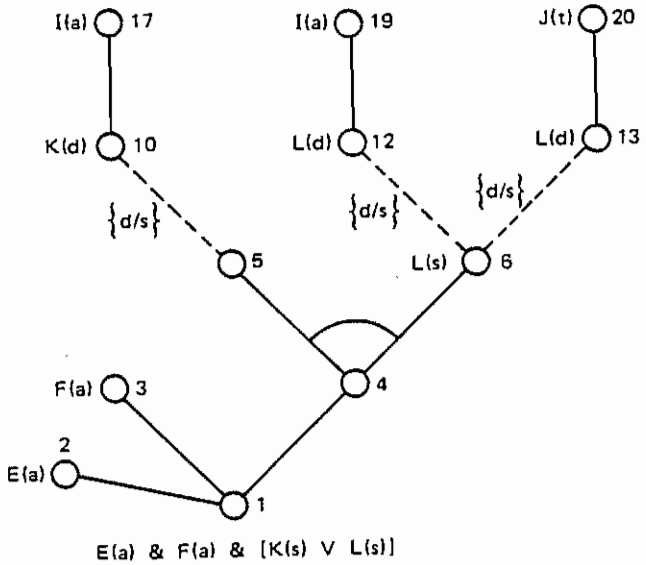


Figure 10. The Fact Tree After Applying Four OPERATORs

We note that the last two CANCEL relations can be combined to yield CANCEL(4, 31) with associated substitution {d/s, a/x}. This in turn yields CANCEL(1, 31), {d/s, a/x}.

In a straightforward manner, we can next match against the OPERATOR nodes 7 and 8 and perform matches between literal nodes to obtain:

CANCEL(2, 21) {a/y}
CANCEL(3, 22) {a/y}

These relations produce the sequence:

CANCEL(1, 21) {a/y}
CANCEL(1, 22) {a/y}
CANCEL(1, 26) {a/y}

Proceeding through the match edge between nodes 26 and 30, we obtain:

CANCEL(1, 30) {a/x}

Combining this relation with CANCEL(1, 31), {d/s, a/x}, we obtain finally:

CANCEL(1, 1) {d/s, a/x}

Thus the goal is proved from the given facts. The relevant instance of Fact ⊢ Goal, useful for many information retrieval applications, can be simply obtained by applying the substitution associated with CANCEL(1, 1) to both the fact and goal expressions. This operation yields:

E(a) & F(a) & [K(d) v L(d)] ⊢ [A(a) & B(a)]

VII Some Additional Extensions

A. Embedding New Rules in Operators

One way of relaxing the single-literal restriction on rule patterns is to allow rules to be embedded in the consequents of OPERATORs. Since (A & B) => C is equivalent to A => (B => C), we can get the effect of the conjunctive pattern by adding the new OPERATOR B => C when A appears in the fact tree. One cannot simply add the new rule to the global rule base, however. Suppose we have the OPERATOR A => (B => C) and the fact A v D. When B => C is added as a new OPERATOR, we must be careful not to use it on the disjunct D. The rule B => C can only be used "in the context" of A.

A simple generalization of our rule-based system supports the correct use of OPERATORs embedded in the consequents of OPERATORs. (Embedding REDUCERs in OPERATORs appears to be much more complex. Thus, we will not use REDUCER contrapositive forms of embedded OPERATORs.) The generalization involves associating each OPERATOR with a node of the fact tree. The initial set of conjunctive OPERATORs is associated with the root of the fact tree. An OPERATOR added at node n is associated with node n. The OPERATORs associated with node n can be used on facts associated with node n or its descendants.

This technique even generalizes nicely to permit "disjunctive" OPERATORs. Suppose we have an OPERATOR of the form A => [(B => C) v (D => E)]. Before such a rule disjunction is associated with the fact tree at literal node A, we split node A into the disjunction A v A and represent the disjunction by two OR node descendants of A. A different rule disjunct is then associated with each of the OR nodes. If the initial OPERATORs are in some complex logical relationship to each other, we represent this relationship by the appropriate OR/AND tree and

label each of the tip nodes of this tree by the initial fact expression. This fact expression is then put in OR/AND tree form at each of the tips.

If the embedded OPERATORs contain quantified variables, these can be Skolemized at the time the OPERATORs are associated with nodes in the fact tree. Care must be taken to ensure that the appropriate instance of an embedded OPERATOR is added.

B. High Complexity Proofs

Our system, as we have described it so far, is not able to find proofs for which any of the goal or fact expressions need to be rewritten with different variables and used a multiple number of times. (We can, of course, use the same node any number of times, but such usage does not rewrite any variables in the expression at the node. Also, we can use rules any number of times, each with different variables.) In analogy with a definition of proof complexity given by Sickel (1976), we shall say that the complexity level of a proof is precisely the number of times a fact or goal expression must be rewritten for multiple use. So far then, our system can produce only proofs of complexity level zero.

As examples of problems requiring complexity-level-one proofs, we have:

1) Goal: $A(x)$

   Fact: $A(a) \vee A(b)$

and its dual,

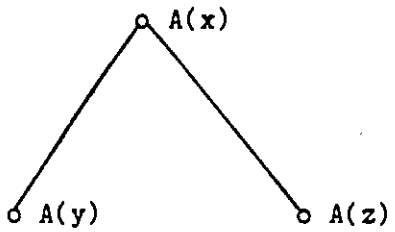
2) Goal: $B(a) \& B(b)$

   Fact: $B(x)$

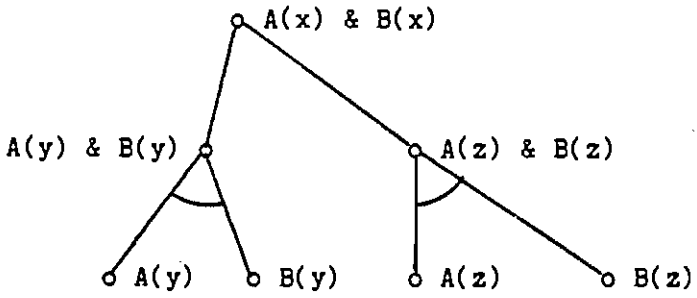
Straightforward attempts at proofs for these problems by our system are frustrated by horizontal consistency violations. However, if in problem 1, for example, we replace the goal by the equivalent one $A(x) \vee A(y)$, then a proof is easy to obtain.

Following Sickel, we might adopt the strategy of trying first to obtain a complexity-level-zero proof. If that attempt fails, we can look for higher complexity proofs in stages. The search for a complexity-level-one proof would involve selecting each of the goal and fact variables (in turn) and rewriting as a disjunction (for goals) or as a conjunction (for facts) the highest node in the goal or fact tree that contains that variable. Substitution consistency violations provide obvious clues about which variables should be rewritten.

To rewrite a goal node A(x), for example, we produce the following tree structure:



To rewrite a goal node A(x) & B(x), for example, we produce the following tree structure:



VIII Conclusions

We have presented a design for a general system that uses production rules and a data base of fact and goal trees to perform deductions. The system can be regarded as a synthesis of many current and some new ideas in automatic deduction. The major innovations presented in this paper are the OR/AND fact tree and the CANCEL operation. These ideas bring a simplifying symmetry to several of the standard techniques for reasoning about facts and goals.

Logical completeness of the general system has not been a design goal. Instead, we assign responsibility for acceptable performance of any specific system to the domain expert, who provides the rules, and to the designer of the specific system, who can repair any unacceptable deficiencies in performance by adding or modifying rules or facts.

An important topic that we have not yet addressed concerns the control strategy for the system. Specialized control strategies for different domains of application (for example, deductive retrieval, theorem proving, common sense reasoning) will probably be necessary in order to achieve high performance. The control system must ensure that the appropriate rule is used sufficiently often to prevent the usual combinatorial explosion. Separation of facts, OPERATORs, and REDUCERs should, we believe, help contain this explosion.

It is also hoped that the proposed system will serve as the beginning of a theoretical foundation for the various applications of "rule-based systems" now being developed by AI research. Many of these systems are fundamentally deduction systems ev n though some of them allow uncertain or probabilistic f..cts and rules. Extending the pres.nt system so that it could also deal with uncertain knowledge would be a valuable future project.

IX Acknowledgements

The author would like to thank the following people for providing helpful suggestions and criticisms and for their comments on early versions of this paper: Douglas Appelt, W. W. Bledsoe, Richard Fikes, Gary Hendrix, Robert C. Moore, and Earl Sacerdoti.

REFERENCES

Bledsoe, W. W. (1977), "Non-resolution Theorem Proving," Artificial Intelligence, in press.

Bobrow, D. and T. Winograd (1977), "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, Vol. 1, No. 1, January.

Chang, C., and R. C. Lee (1973), Symbolic Logic and Mechanical Theorem Proving, Academic Press, Inc., New York.

Davis, R., and J. King (1977), "An Overview of Production Systems," in E. Elcock and D. Michie (eds.), Machine Intelligence 8: Machine Representations of Knowledge, Wiley, New York.

Fikes, R., and G. Hendrix (1977), "A Network-Based Knowledge Representation and its Natural Deduction System," Proc. IJCAI-77.

Hewitt, C. (1971), "Procedural Embedding of Knowledge in PLANNER," Proc IJCAI-71, British Computer Society, London, England, pp 167-182.

Kowalski, R. (1974a), A Proof Procedure Using Connection Graphs, University of Edinburgh School of Artificial Intelligence Memo No. 74, February.

Kowalski, R. (1974b), Logic for Problem Solving, University of Edinburgh School of Artificial Intelligence Memo No. 75.

Moore, R. C. (1975), Reasoning from Incomplete Knowledge in a Procedural Deduction System, MIT Artificial Intelligence Laboratory Report No. AI-TR-347, December.

Nevins, A. J. (1975), "A Relaxation Approach to Splitting in an Automatic Theorem Prover," Artificial Intelligence, Vol. 6, No. 1, Spring, pp. 25-39.

Nilsson, N. J. (1971), Problem Solving Methods in Artificial Intelligence, McGraw Hill Book Co., New York.

Reiter, R. (1976), "A Semantically Guided Deductive System for Automatic Theorem Proving," IEEE Trans. on Computers, Vol. C-25, No. 4, April, pp. 328-334.

Sickel, S. (1976), "A Search Technique for Clause Interconnectivity Graphs," IEEE Trans. on Computers, Vol. C-25, No. 8, August.

Wilkins, D., (1974), "A Non-Clausal Theorem Proving System," Proc. AISB Summer Conference.