APRVR: A PRIORITY-ORDERED AGENDA THEOREM PROVER

by

WILLIAM MABRY TYSON

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 1981

APRVR: A PRIORITY-ORDERED AGENDA THEOREM PROVER

APPROVED BY SUPERVISORY COMMITTEE:

TO

DODY, KATHY,

AND MY PARENTS

ACKNOWLEDGMENTS

I am deeply indebted to Woody Bledsoe, my advisor, for all his guidance and help. He has been a source of ideas and inspiration and an example of excellence as both a researcher and a person.

I would also like to express my thanks and appreciation to the rest of my committee: Dan Chester, Don Good, Bob Simmons, John Alexander, and Jay Misra.

Clive Dawson has given me great technical support but even that is secondary to his friendship. Rich Cohen has also given his support and friendship.

There is really no way to express my thanks to my parents who have provided me with a good base and have continued to support me throughout.

Although she is too young to understand, I must acknowledge the patience of my daughter Kathy while she shared her Daddy so he could write his "book".

My most sincere gratitude and love go to my wife Dody. She has given her love and gentle, yet strong, support throughout this work and, meanwhile, given us a very happy life together.

This work was supported in part by National Science Foundation grant MCS 80-11417.

APRVR: A PRIORITY-ORDERED AGENDA THEOREM PROVER

Publication No.
William Mabry Tyson, Ph.D.
The University of Texas at Austin, 1981

Supervising Professor: W. W. Bledsoe

This dissertation is concerned with research in automatic theorem proving by computers. Discussed herein are issues involved with designing a natural deduction theorem proving system whose search is guided by the priorities assigned to the formulas it attempts to prove and the methods used to prove them. A list of the tasks to be done is kept in an agenda ordered by these priorities. A particular implementation of a priority-ordered agenda theorem prover, named APRVR, is described but the main purpose of the research is to explore the advantages and problems of using this search strategy for theorem proving.

The first chapter introduces theorem proving in general, and natural deduction theorem proving in particular. The need for a new combination of search strategy and control structure is motivated and the agenda-based system is shown to fill this need. The objective of the research for this dissertation is set.

Providing the groundwork for the later chapters, the second chapter describes the proof methods of this natural deduction theorem prover, including subgoaling, splitting, and chaining. The soundness of the basic deduction rules is proved in an appendix and involves a generalization of ordinary substitutions.

The agenda mechanism is described in the third chapter. The agenda consists of a list of goals, each of which is a data structure including the formula to be proved, the method to be used, and two separate components of the priority assigned to the goal. Some considerations are presented for assigning the initial priority to the goal and the later adjusting of this priority as the proof proceeds. The agenda-based system allows for improvements to two existing proof methods. More detail is presented in Chapter 4 of how APRVR implements this agenda mechanism.

The examples given to APRVR are described in Chapter 5 and include some problems from symbolic logic, set theory, and group theory. The final example is a linear real inequality that has never before been proved by machine. APRVR's performance on these examples is discussed. Example proofs are included in an appendix.

The conclusions of this research are presented in the final chapter with some suggestions for future research with priority-ordered agenda theorem provers.

TABLE OF CONTENTS

CHAPTER 1.

INTRODUCTION

1.1 Theorem Proving

Since the time of Thales of Miletus twenty-five centuries ago, man has been using deduction to prove theorems. Frege formalized this into the predicate calculus 100 years ago. Fifty years ago, Herbrand and Gentzen developed methodologies for mechanical theorem proving. For approximately twenty-five years, researchers have been making slow progress toward developing computer based, predicate calculus theorem provers. The early efforts demonstrated that theoretical procedures, while logically beautiful, are not necessarily practical. The Herbrand procedures ran into combinatorial explosions: simple theorems were provable but the proofs of only slightly more complex theorems were far beyond what could be computed on existing machines.

Predicate calculus is not decidable; no algorithm can exist to determine whether a formula is valid or not. As a result, combinatorial explosions are inevitable. Even decidable subsets of predicate calculus often run into this. For example, Presburger arithmetic has a time complexity on the order of

$$2^{2^{2^n}}$$

in the length of the formula [Oppen-78].

The development of the unification and resolution theorems [Robinson-65] brought a flurry of activity in computer theorem proving. The application of the basic resolution principle was again plagued by combinatorial explosion, although it was not as bad as before. Numerous refinements of resolution have been developed improving its performance. Research is actively being done on resolution based theorem provers; applications include deductions on data bases and program verification.

Resolution was not the only type of theorem proving being done. Other types of theorem proving included those that tried to do proofs in a manner more closely related to the way people have normally proved theorems for centuries. The earliest of these included the Logic Theorist of Newell, Simon, and Shaw [Newell-57] and the Geometry-Theorem Proving Machine of Gelernter [Gelernter-59]. Research is continuing with these natural deduction type systems by Bledsoe [Bledsoe-71, Bledsoe-78], Boyer and Moore [Boyer-75, Boyer-79], and others. Bledsoe's natural deduction system is in active use as part of a program verification system [Good-75].

1.2 Motivation

APRVR was based upon earlier work by W. W. Bledsoe [Bledsoe-71, Bledsoe-72, Bledsoe-78] on his interactive prover (to which I shall refer by the name of its main routine, **IMPLY**). Both provers are natural deduction systems in first order logic which use the concepts of subgoaling, backward chaining, and forward chaining. Bledsoe's IMPLY and its offshoots has been used successfully in a number of areas: set theory [Bledsoe-71], topology [Bledsoe-74], program verification [Good-75], limit theorems [Bledsoe-72, Ballantyne-75], and linear algebra [Bruell-79]. While many (but not all) of the proof methods used in APRVR are similar or identical to those in IMPLY, the control structures of the two provers are radically different. APRVR uses an agenda-based control structure as opposed to the hierarchical control structure of IMPLY.

One of IMPLY's strongest features is its man-machine interface which allows the user a number of capabilities to monitor and affect the progress of a proof. The user would typically watch the progress of the proof and interact with the prover when it needed help. If IMPLY came to a goal which was unprovable without some additional lemmas that were not supplied in the original proof, the user could add them. If IMPLY was spending too much effort on some subgoal that the user knew was not provable, the user could cause IMPLY to reject that subgoal. Commands existed that would allow the user to direct IMPLY to attempt some rarely used method in order to find the proof. Such a method might be one that was usually a waste of resources and time but was useful on rare occasions.

During my extensive usage of IMPLY, I found that the man-machine interaction was most often used to direct the prover toward or away from particular alternative attempts at proving the goal under consideration. The user would realize that the prover, left to itself, would try sequentially its arsenal of methods (possibly sequentially on some subparts of the hypothesis or conclusion) and so spend a considerable amount of time going down dead-end paths before finding the correct path. So he would alter the normal sequence of events so as to cause IMPLY to find the proof more quickly.

An alternative method of proceeding towards the goal became apparent. If the theorem prover could try several possible paths to a proof for a short distance, it would have more information about which path is best. In fact, one of the paths may have already found a proof. If not, it should be better able to choose on which one of the paths would be best to continue. After proceeding down this path further, the theorem prover may decide this was not as promising as first expected. It might determine that another of the alternative paths is now more likely to return a proof. The best action at this point would be for the theorem prover to switch over to that other path on which it had already done some work. Even later, it might be decided that the first alternative chosen has again become the most promising and so would be chosen to be processed further. If no progress had been made by some much later time, the system could decide that it was time to try some strategies that, while rarely worth doing early in the proof, might lead to a proof when the more commonly successful strategies appeared to be failing.

What I have just described is an **agenda** system where the items on the agenda are the paths attempting to prove the goal. This is not a new idea. For many years operating systems have had queues that operate according to the same concept. More recently, Doug Lenat [Lenat-76] has demonstrated the utility of an agenda system with his AM system which discovered such mathematical concepts as primes, Diophantine equations, the prime factorization theorem, and Goldbach's conjecture. Peter Bruell [Bruell-79] developed a version of IMPLY that used a somewhat different concept of an agenda. He only allowed tasks on the agenda that were alternative methods of proving the same goal. When one task on the agenda had been processed, it could not be retried. Thus there was no way to alternate between different attempts at a proof.

1.3 Dissertation

As the potentials of an agenda system became obvious so did the problems. The agenda is a list of ordered tasks. How does one calculate the priorities that should be assigned to the various tasks and what factors contribute to them? Indeed, can there even be a single priority for a task or does it depend upon the state of the proof? How can the switching between tasks be accomplished? Would an agenda system add any power that is not available to the hierarchical IMPLY? I decided to implement a theorem prover using an agenda-based control structure and to experiment on it to gain knowledge about these problems. Hopefully, such knowledge would be transferable to other provers regardless of their control structures.

For computational simplicity, I assumed that the priority of a task (goal) would be a single number. This allows a single ordered list of goals. Originally this number was atomic; it stood by itself and had no subparts. After some early experiments, I found that the priority could be readily updated if this single number was divided into two separate numbers (which could then be arithmetically combined into a single number which is the priority of the goal).

The implementation of the capability to switch back and forth amongst the paths of a proof was not immediately apparent. The more obvious solution would have been to use the same hierarchical IMPLY but to use coroutines (such as Interlisp's spaghetti stack [Teitelman-78]) to record the program's status when a path is abandoned so it could be resumed later. No such method was available to me at the time[1] and so it would have been necessary to implement it. Even if Interlisp's spaghetti stack had been available, it may have been prohibitively expensive to use.

An alternative to recording the execution state of the process when it suspends a path is to suspend the path only at specific points from which it may be easily be continued. This places the burden upon the routines used to follow the paths to suspend themselves after recording what information is necessary for their continuation. Interestingly, the recursive nature of IMPLY made this fairly simple. Each time any routine wished to call the routine IMPLY to prove a subgoal, rather than directly calling IMPLY on the subgoal, the routine would add a task to the agenda to solve the subgoal and then would suspend itself pending the return of a value from the newly scheduled goal. The hierarchical, recursive IMPLY is now flattened out into an agenda-based theorem prover operating on a list of goals. Thus the procedure used is to have a controlling routine that picks the most promising task from the agenda, pass control to the routine that is to

[1]APRVR was written in the University of Texas's version of UCI-LISP [Meehan-79] without using any features unique to that language.

process that task, wait for that routine to finish (which should be fairly quickly), and then loop back to pick the best task from the agenda again.

From the beginning I felt that the power of the agenda-based mechanism would be in its ability to proceed a short way down an attempted proof and then suspend it in favor of another, now more promising path. I realized this meant that sometimes the prover might leave the correct path for other superficially more attractive paths but I felt that these cases would be outweighed by the cases where it leaves an incorrect path for the correct one.

Although I had expected this to be the sole source of power, I discovered another method of finding the proof of a goal that was impossible under the recursive IMPLY. Because all the goals on the agenda are linked (by PARENT and SUBGOAL links), it is possible to examine the structure of the attempted proof. This allows a goal not only to know its parent(s) and subgoals but also to examine the whole proof tree. I discovered that when a goal succeeds partially (on one or more conjuncts of its conclusion) but fails on the others, then this information may be useful much earlier in the proof tree. In particular, if the goal finds a disjunct in the hypothesis and can prove the goal if it assumes one of the disjuncts but can not prove it assuming the other disjunct, then it may be reasonable to use that hypothesis (suitably instantiated) to be the basis for a case split much earlier in the proof tree.

This case splitting mechanism is somewhat similar to the case splitting done in [Bledsoe-77] but is more general. The older method is limited to inequalities and has a special mechanism built in to pass back the inequalities that constitute the remaining cases after some are satisfied. My method does not restrict case splitting to cases about inequalities. When the decision is made to try proof by cases, the task making that decision schedules the task to prove the cases. No other tasks are burdened with extra machinery because of this capability. Furthermore, because of the nature of the agenda, APRVR does not have to commit itself totally to the proof by cases as IMPLY does. It can still continue attempting to find a proof without doing a case split or it may also generate several candidate case splits.

In my planning of the work to be done on agendas, I decided that it would be best if I were able to experiment on a domain of theorems on that was not subject to being better solved using domain-specific knowledge. The purpose of this was two-fold. First, I was interested in experimenting with the agenda mechanism without any interfering special purpose mechanisms for the particular domain. By refusing to add such special purpose mechanisms, my effort could be concentrated solely on the problems of working with an agenda rather than trying to work on the special purpose procedures and their interactions with the agenda. Furthermore, whatever power I developed in my system could credited to the agenda rather than being passed off as a result of some powerful special purpose mechanism I might have employed. Secondly, this theorem prover, while based partly in concept on IMPLY, was all new code. By starting out fresh, I was a number of man-years behind other existing theorem provers in the development of the basic theorem prover and any special purpose mechanisms. I did not want to have to compare my system to others that had been much more extensively developed.

Considering all this, I decided to develop my system using as a domain some theorems from *Principia Mathematica* [Whitehead-70]. These theorems are purely logical and do not include any functions that might be attacked by special mechanisms. Unfortunately, for the theorems I did, my system was too powerful for these simple logical concepts. It employs logical reasonings that were not yet proved valid by the propositions preceding the proposition being proved. Therefore it was necessary to treat the logical symbols of the propositions differently than the normal logical symbols operated upon. Proof techniques such as ANDSPLIT, backward chaining, and forward chaining could not be used (directly) on these theorems; but I was not interested in proving difficult theorems, only in exploring priorities and agendas.

After developing APRVR on the rather barren propositions from *Principia Mathematica*, I attempted and proved some more traditional theorems. The first set were some basic theorems about sets and their complements, intersections, and unions. The most complex of the theorems was DeMorgan's Law that the complement of the intersection of two sets is the union of the complements of the sets. Following this, I proved a standard theorem for resolution theorem provers: If the square of every element in a group is the identity element, then the group is Abelian.

Finally, I attempted and proved a very hard problem that had never been proved before by machine. This particular problem, called AM8, is due to my advisor, W. W. Bledsoe, from his work in proofs involving linear real inequalities. Specifically it comes from a proof of the theorem that a continuous function attains its minimum over a closed region. It is quite difficult even for people and I challenge the reader to attempt this theorem before proceeding. Note that although the problem depends heavily upon inequalities, no special purpose machinery was built in for that (although such machinery very well might have sped up the solution of the problem).

AM8:

$$\begin{aligned} &\forall t\,[L < t \Rightarrow F(L) \le F(t)] \\ &\wedge \forall x\,(x > L \Rightarrow \exists t\,[t \le x \wedge F(x) > F(t)]) \\ &\wedge \forall w\,\exists g\,[F(g) \le F(w) \wedge \forall x'\,(F(x') \le F(w) \Rightarrow g \le x')] \\ \Rightarrow &\exists u\,\forall t'\; F(u) \le F(t') \end{aligned}$$

CHAPTER 2.

PROOF METHODS

2.1 Introduction

Before getting into the details of the agenda mechanism, it is necessary to describe the nature of the tasks that will be put onto the agenda. In order to do this, I shall describe the proof methods used.

Theorem proving is not an easy task for people; yet, people are much more proficient at it than machines. Researchers in automatic theorem proving are trying to understand the many methods (logical and extralogical) used by human theorem provers to find proofs and to incorporate them into their computer theorem provers. I feel that until such time as we understand these human techniques better, it is best to use theorem provers, described as **natural deduction** theorem provers, that generate proofs that are somewhat similar to those generated by people. Hopefully, because of this similarity, a natural deduction theorem prover will lend itself to the adoption of the methods used by people as we discover how to do them on a computer. Unfortunately, a natural deduction theorem prover can not be encoded as efficiently as, say, a Resolution [Robinson-65] theorem prover, meaning that the natural deduction system may not perform as well on the simpler problems. But the real question of the power of a research theorem prover is not its efficiency on any particular theorem (although in application areas such as program verification or data bases this is critical) but instead is what theorems it can prove that other theorem provers have not before been able to prove.

2.2 Subgoaling

APRVR is a natural deduction theorem prover. It is originally given a particular goal (the "theorem") to attempt to prove and tries to prove it by reducing that problem to possibly simpler subgoals. The prover then tries to prove these subgoals in exactly the same manner that it tries to prove the original goal.

APRVR works by developing and searching an AND/OR tree [Nilsson-80] of goals. One particular goal (which may just be some deep subgoal of the original problem) is being concentrated on at any point in time. In trying to solve this goal, APRVR may generate several subgoals of that problem. Some of these may be in an OR relation to the others; that is, if one of these subgoals is solved then the goal is solved. I shall refer to this as an **ORSPLIT** (because splitting an OR in the conclusion of a goal is a common example that generates this structure). Other subgoals may be in an AND relation to other subgoals (which may not yet exist physically), all of which must be solved in order for the original goal to be solved. This splitting into subgoals is termed an **ANDSPLIT** because it usually arises from splitting on an AND in the conclusion of the goal. Usually only the first goal of an AND branch exists and the others will be created as needed.

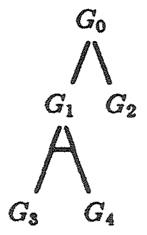


Figure 1

An example of an AND/OR tree is in Figure 1. The node labelled G_0 has below it two subgoals, G_1 and G_2, that are in an OR relation to each other. The goal G_1 has two subgoals, G_3 and G_4, below it that are in an AND relation to each other. Such a situation might occur if goal G_0 represents an attempt at proving

$$H \Rightarrow ((C_1 \wedge C_2) \vee C_3).$$

If either of the goals

$$H \Rightarrow (C_1 \wedge C_2) \qquad \text{(Goal } G_1\text{)}$$

or

$$H \Rightarrow C_3 \qquad \text{(Goal } G_2\text{)}$$

is proved, then G_0 would be proved. This is an example of an **ORSPLIT**, a split of a goal into subgoals because of an OR in the conclusion of the goal. ORSPLITs may also occur when there are multiple methods of proving a goal (for example, backward chaining on either of two different hypotheses). An example of an **ANDSPLIT** would be the goal G_1 which would be proved if both

$$H \Rightarrow C_1 \qquad \text{(Goal } G_3\text{)}$$

and

$$H \Rightarrow C_2 \qquad \text{(Goal } G_4\text{)}$$

are proved[1].

One of the major differences between APRVR and IMPLY is that IMPLY can keep track of only one path to the proof at once. If an ORSPLIT exists for a goal, then IMPLY must choose which subgoal to work on. In order to try a different branch of the ORSPLIT, IMPLY must terminate (and forget) any work that has been done on the first subgoal. IMPLY does not have any capability of remembering its previous work and continuing that work at a later point. APRVR does have that capability to suspend and resume working on any subgoal. This is one important aspect of the agenda mechanism.

2.3 Skolemization

Theorems (and non-theorems) that are given to APRVR are closed logical formulas possibly containing quantifiers. APRVR immediately removes these quantifiers by the process of Skolemization [Robinson-79, Chang-73] to generate an open formula containing variables which is then given as the original goal to the deductive routines.

Skolemization is traditionally defined as the process of transforming the closed formula into the logically equivalent prenex form (with all the quantifiers in front of the remaining quantifier free form) and then applying certain rules to remove the quantifiers. A formula of the form

$$\exists v\, \forall w\, \exists x\, \exists y\, \forall z\, P(v, w, x, y, z)$$

will be transformed into

$$P(v, S_1(v), x, y, S_2(v, x, y))$$

[1]This is a simplification of what happens. The goal G_4 may depend upon how goal G_3 is proved. In particular, variables that exist in C_2 may be partially instantiated according to the substitution returned from the proof of goal G_3. This will be covered in Chapter 2.4.

where S_1 and S_2 are Skolem functions[2]. This can be understood by realizing that the prover is trying to prove the validity of the quantified formula by finding some instantiation (or some disjunction of instantiations) of the open formula. Thus the existentially quantified variables become Skolem variables for which the prover is free to choose instantiations. The universally quantified variables are replaced by functions because, if the formula in the scope of the quantifier is to be true for every instance of the variable, then it must be proved true for an arbitrary, unspecified value, which is represented by the Skolem function. This Skolem function has as arguments the Skolem variables generated by the existential quantifiers containing the universal quantifier in order to prevent those variables from being instantiated in terms of the Skolem function. In the example, v is not allowed to be instantiated in terms of either of the Skolem functions S_1 or S_2.

If we can find some substitution, such as $\{a/v, b/x, c/y\}$ that generates a valid expression

$$P(a, S_1(a), b, c, S_2(a, b, c))$$

when the substitution is applied to the Skolemized form then we have proved the original form. We have also proved the original form if we find some set of substitutions, such as

$$\{\{a/v, b/x, c/y\}, \{d/v, e/x, f/y\}, \ldots\}$$

such that the disjunction of the results of applying the substitutions to the Skolemized form,

$$\begin{array}{l} P(a, S_1(a), b, c, S_2(a, b, c)) \\ \vee P(d, S_1(d), e, f, S_2(d, e, f)) \\ \vee \ldots, \end{array}$$

is valid[3]. This is a result of existential generalization:

$$(P(A) \vee P(B) \vee \ldots) \Rightarrow \exists x\, P(x)$$

APRVR actually does something somewhat more complex than this simple Skolemization. In order to preclude any unnecessary dependencies of the Skolem functions on the Skolem variables,

[2]Note that the Skolemization done in Resolution theorem proving is the opposite of this. The reason for this apparent inconsistency is that APRVR is trying to prove the validity of the formula rather than the inconsistency of its negation. If the two Skolemization techniques are given negatively opposite formulas, the results are identical except for the negation symbol.

[3]For more information, see Appendix A.

the formula is not put into prenex form before the quantifiers are removed. Thus a formula of the form

$$\forall w\, \exists x\, P(w, x) \Rightarrow \exists y\, \forall z\, Q(y, z) \tag{1}$$

is Skolemized into

$$P(w, S_1(w)) \Rightarrow Q(y, S_2(y)) \tag{2}$$

rather than into one of

$$P(w, S_1(w)) \Rightarrow Q(y, S_2(w, y))$$

$$P(w, S_1(w, y)) \Rightarrow Q(y, S_2(w))$$

$$P(w, S_1(w, y)) \Rightarrow Q(y, S_2(w, y))$$

depending upon which prenex form of the formula is chosen.

All goals in APRVR are open formulas. When APRVR proves a goal, it returns a substitution as the value of that proof. This indicates that if the designated substitutions are made for the variables in the open formula, then the resulting formula can be proved, treating any other variables in that formula as constants. If APRVR returns a substitution of

$$\{a/w, b/y\}$$

for the formula (2) above, then APRVR has proved that

$$P(a, S_1(a)) \Rightarrow Q(b, S_2(b))$$

is valid. Thus the original formula (1) is true.

Since the substitution returned is only applicable for the open formula it proves, any variables in the domain of the substitution that are not in the open formula are removed. This might occur if the proof involved using lemmas (not expressed in the formula) whose Skolemized form contained variables.

The substitutions necessary to prove a particular open formula are not always simple. Suppose we have the open formula

$$P(x) \Rightarrow (P(a) \wedge P(b)).$$

There is not a single substitution that will make this valid. The problem is that both substitutions $\{a/x\}$ and $\{b/x\}$ are needed. This is overcome by using the **generalized substitution**

$$(\{a/x\} \vee \{b/x\}).$$

By the definition of generalized substitutions[4], the result of applying this substitution to the open formula is

$$[P(a) \Rightarrow (P(a) \wedge P(b))] \vee [P(b) \Rightarrow (P(a) \wedge P(b))]$$

which is a valid formula. Such generalized substitutions may be necessary when the substitutions proving two subgoals must be combined.

APRVR is a little more clever than the above example might indicate. If the $P(x)$ in the hypothesis had been a hypothesis $\forall x\, P(x)$ in the original theorem given to APRVR, APRVR will treat the $P(x)$ like a lemma. Thus APRVR is, in effect, simply proving the open formula

$$P(a) \wedge P(b)$$

but is allowed to use the "lemma" $\forall x\, P(x)$. Whenever $P(x)$ would be used, a distinct variable is used for x. So APRVR may prove

$$P(x_{100}) \Rightarrow P(a)$$

and

$$P(x_{101}) \Rightarrow P(b)$$

and would never need to generate the generalized substitution described above.

2.4 ANDSPLITs

The dominant theorem proving concept in APRVR is the transformation of a problem into an equivalent problem or into component subproblems. A goal of the form

$$[H \wedge (A \vee B)] \Rightarrow C$$

[4]Generalized substitutions are defined and explained in detail in Appendix A.

is transformed into

$$H \Rightarrow [(A \Rightarrow C) \wedge (B \Rightarrow C)].$$

Although this is logically equivalent to the original form, it will be termed a subgoal of the original goal. This goal is then split into two subgoals:

$$H \Rightarrow (A \Rightarrow C)$$

and

$$H \Rightarrow (B \Rightarrow C)$$

which must be proved. The first of these is then transformed into

$$(H \wedge A) \Rightarrow C.$$

It is this concept of subgoaling that allows the agenda mechanism to be useful.

The most interesting usage of subgoaling comes in doing ANDSPLITs. When an AND is the main logical connective of the conclusion of the goal, APRVR splits the goal into two subgoals. First, a subgoal that tries to prove one of the conjuncts is generated. For example, if the goal is to prove

$$H \Rightarrow (A \wedge B)$$

then a subgoal

$$H \Rightarrow A$$

is generated. When this subgoal is proved and returns a substitution θ, then another subgoal

$$H \Rightarrow B\theta'$$

is generated. (See Appendix A for an explanation of the notation θ'. Normally θ' is identical to θ.) If this subgoal is then proved with a substitution λ, then the two substitutions are combined (usually by composition but occasionally into a disjunctive generalized substitution) and returned as the substitution proving this goal.

Although this ANDSPLIT looks simple, there are some subtle problems that occur because of it. The most common problem is trapping. If the first subgoal returns only one of the valid substitutions proving that subgoal and it is not one that will lead to a provable second subgoal (while another substitution would have), this is termed **trapping**. Some of the methods that have been tried to avoid this include attempting the various possible first subgoals or reproving the first subgoal but preventing it from returning the same substitution. APRVR avoids this problem by allowing subgoals to return multiple values. If several substitutions are found simultaneously as would be the case for a subgoal of the form

$$P(a) \land P(b) \Rightarrow P(x)$$

then they are all returned as a set[5]. Furthermore, because of the agenda system, the goal may not need to be discarded after some proving substitutions are returned. It is simply rescheduled for a later time in case its parent goal is not proved. If necessary, it will be continued and will hopefully discover the proof necessary to work with the other branches of the overall proof.

Another problem that an ANDSPLIT can create is one in which conflicting substitutions are returned. This was the case in the example above where both substitutions $\{a/x\}$ and $\{b/x\}$ were needed. In that ANDSPLIT, the substitution proving the first subgoal was $\{a/x\}$ while the substitution proving the second subgoal was $\{b/x\}$. Combining these conflicting substitutions was a problem in the earlier IMPLY system but APRVR uses generalized substitutions to return both substitutions in this case.

A final, more subtle problem occurs in the selection of which of the possible first subgoals to use. If the wrong choice is made, the first subgoal may have too great a degree of freedom and so may return several proofs which are incompatible with the remaining subgoals. If a different choice had been made, the proof may have been made easier by the bindings made to the variables by the alternate first subgoal. APRVR combats this problem by allowing all the possible first subgoals to have a chance at generating a quick proof (as explained later). If any do, the corresponding secondary subgoals are generated and the proof continues. If none do, APRVR reverts to selecting one of the subgoals to work on with more patience while cancelling the other subgoals it started.

[5]Note that this is different than Nevins [Nevins-74] where he returns all solutions available under certain heuristics; he does not find some and later go back for more.

In this way, APRVR relieves some of the risk in chosing only one first subgoal without unduly increasing the branching factor of the proof tree.

2.5 Backward Chaining

A common proof technique used in APRVR is **backward chaining**. This is a form of an ANDSPLIT that occurs when an implication in the hypothesis is used to try to prove the conclusion. The first step is to prove the conclusion of the goal, given the conclusion of the implication. Usually this is achieved by a simple matching without doing extensive search to find this proof. The second subgoal is generated by trying to establish the hypothesis of the implication (suitably instantiated) assuming the hypothesis of the goal.

If the goal is

$$[H \wedge (A \Rightarrow B)] \Rightarrow C, \tag{3}$$

the first subgoal to establish is[6]

$$B \Rightarrow C.$$

If this is proved with some substitution θ, then the second subgoal is

$$H \Rightarrow A\theta'.$$

A similar technique, **detachment**, is used in the *Principia Mathematica* examples. If the goal is simply C and a lemma

$$L_H \Rightarrow L_C$$

exists such that C matches L_C by a substitution θ, then a subgoal

$$L_H \theta$$

will finish the proof. Chapter 5.2 explains why this was needed for reasons of implementation.

Backward chaining is a useful technique. Because the first step is to match with the conclusion of the goal, backward chaining is more likely to remain on the right path to the proof. It is also less likely to generate problems of infinite recursion than the related method, forward chaining.

[6]The first subgoal could be $(H \wedge B) \Rightarrow C$, but that might be diverted into effectively trying to prove $H \Rightarrow C$.

2.6 Forward Chaining

Forward chaining can be attempted in the same situation where backward chaining can be. The difference is that while backward chaining works backward across the implication, forward chaining works forward. That is, it tries to establish the validity of the antecedent of the implication as the first subgoal. If that succeeds, the second subgoal is similar to the original goal except that there is now an additional hypothesis. Both the new hypothesis (originally the conclusion of the implication chained on) and the conclusion are instantiated by the substitution proving the first subgoal.

If the formula to prove is (3), the same goal as in backward chaining, then the first subgoal to be proved is

$$H \Rightarrow A.$$

If this is proved with the value θ, then the second subgoal becomes

$$[H \wedge (A \Rightarrow B) \wedge B\theta] \Rightarrow C\theta.$$

Forward chaining is difficult to deal with. Doing forward chainings early in a problem creates extra hypotheses that may clutter the proof and obscure the important hypotheses. The irrelevant hypotheses generated by forward chaining increase the branching factor and so the cost of doing the proof. The alternative of not doing forward chaining early is to risk missing a proof later by not having a needed fact or to try forward chaining later, perhaps repeatedly, whenever it *might* be needed. In doing forward chaining, one must be careful to prevent infinite loops generating new hypotheses that in turn will match the antecedent of the hypothesis being chained on. (Both forward and backward chaining can generate infinite loops. This is only a real problem where it can loop unchecked.)

2.7 Case Splitting

Occasionally there may not be a simple, straightforward proof of a theorem; several different proofs may be required according to different situations or cases. One attempted proof of a theorem may hinge upon a premise that is not always true. In such situations, it may be advantageous to accept this proof for the cases where the premise is true and then try to find a different proof of the theorem for all the other cases. This strategy is called case splitting or proof by cases. If only the two cases, A or B, are possible, then the proof by cases of

$$H \Rightarrow C$$

involves proving both

$$(A \wedge H) \Rightarrow C$$

and

$$(B \wedge H) \Rightarrow C.$$

2.8 Other methods

The methods listed so far constitute the majority of the techniques used in proving a theorem. There are other methods that are used but not as frequently. Proof by contradiction is one such method. In this method the conclusion is negated and added to the hypotheses. One of the hypotheses is negated and used as the new conclusion. Most often this comes into use when a forward chaining has generated a hypothesis that is inconsistent with another hypothesis. The contradiction is noted and the proof succeeds.

Backward chaining and forward chaining can be reversed by using the contrapositive of the implication chained on. Reverse backward chaining in

$$[H \wedge (A \Rightarrow B)] \Rightarrow C$$

would be equivalent to backward chaining in

$$[H \wedge (\sim B \Rightarrow \sim A)] \Rightarrow C$$

so that the first subgoal generated would be

$$\sim A \Rightarrow C.$$

Reverse forward chaining is similarly defined. These two types of chaining are done rather than the regular backward and forward chainings if both the hypothesis and conclusion of the first subgoal will have the same sign, either both negated or neither negated.

When the conclusion of a goal is an implication, its hypothesis is promoted into the hypothesis of the goal. If the goal is

$$H \Rightarrow (A \Rightarrow B),$$

the result of promoting the implication is

$$(H \wedge A) \Rightarrow B.$$

Many different truth-preserving transformations are made on goals. Conjuncts and disjuncts may be reordered. Negations may be distributed (or cancelled) across other logical operators. True conjuncts and false disjuncts are removed. Terms may be collected so that a goal

$$H \Rightarrow [(A \wedge B) \vee (A \wedge C)]$$

will be reduced to

$$H \Rightarrow [A \wedge (B \vee C)].$$

Some of these are applied by a rewriting routine that simplifies new goals and others are applied when they become necessary.

CHAPTER 3.

THE AGENDA MECHANISM

3.1 Introduction

As has been stated, APRVR works by reducing a goal to (presumably) simpler subgoals. As these goals are created they are added to an ordered list called the agenda, according to the priority assigned to them. When APRVR is ready to select a new task, it chooses the highest priority goal on the agenda. To attempt to prove the chosen goal, APRVR calls the appropriate routine which then does limited processing trying to find proofs. If it is successful, it will usually immediately return the proof after rescheduling the goal (at a much lower priority) in the event that a different proof (substitution) is the one needed. More often, while processing the goal, a routine will spin off some subgoals and suspend its processing (by exiting after removing the goal from the agenda) and wait for one of the subgoals to complete a proof of its formula and to reschedule this goal. A routine may also reschedule the goal (at a lower priority) along with some information indicating what is left to be done. If the goal later reaches the head of the agenda, the routine will be called again and processing will resume where it left off. Of course, the goal may never be recalled if it is not necessary to the proof.

The agenda mechanism is basically similar to the heuristic graph search procedure described in [Nilsson-80] except that nodes (goals) selected are often left on the open list (agenda). The priority of a goal is determined by the depth of the goal (via an overhead cost) and the heuristic "estimate" of the magnitude of work left on a goal. This "estimate" actually has little bearing on what the actual amount of work left might be but is a numeric result of combining a number of heuristic factors indicating how likely the goal is part of a proof. APRVR is interested in finding some proof (if one exists) without caring whether it is the shortest.

3.2 Goal Structures

The actual objects stored on the agenda are data structures of which one part is the formula to be proved. Other information stored in the goal data structure includes the goal's priority, its parents and subgoals, its proofs already generated (if any), a flag (the **type** of the goal) indicating what routine is to be called to process this goal, and a slot used by that routine for information local to it.

When a proof routine, such as an ANDSPLIT, needs to have a formula (a subgoal) proved, it generates a goal data structure containing a canonical form of this formula with a goal-type specified as "GOAL"[1]. If a goal with the same formula already exists, a new goal structure will not be created but the old one will be used. Furthermore, if the new formula is identical, up to the renaming of variables, to one already in a GOAL-type goal, a special RENAME-type goal structure is generated (and returned) which will return the appropriate values whenever the similar GOAL-type goal returns values. By these provisions, duplication of work is kept to a minimum. Thus there is a one-to-one correspondence (up to renaming of variables) between every formula generated as a goal at the logical level and the GOAL-type goal structures.

When a routine generates a subgoal of one of these goal structures the subgoal may be one of two types: methodological or logical. A methodological subgoal is a goal structure in which the formula to be proved has not changed, but the goal-type has changed to indicate a different, more specific method is to be tried to prove the formula. A logical subgoal is one in which the formula has changed and is always a GOAL-type goal or a RENAME-type goal.

The routine to process the goals with a type of THMOPS is SOLVETHMOPS which, as its name implies, processes the goal by looking at the logical operators in the formula. This routine can generate both methodological subgoals (generating ANDSPLIT, ORSPLIT, and HAND type goals) and logical subgoals. If SOLVETHMOPS is given a goal whose formula is

$$H \Rightarrow (A \land B)$$

it will generate a goal structure whose formula is the same but whose goal-type is "ANDSPLIT". So SOLVETHMOPS has passed the responsibility of proving this formula through to a routine specific for doing ANDSPLITs. A new goal structure was generated rather than directly calling the routine so that the ANDSPLIT routine could manipulate and reschedule its own goal without

[1]This type goal structure will be referred to as a GOAL-type goal.

interfering with the THMOPS goal.

However, if SOLVETHMOPS is given a formula of the form

$$H \Rightarrow (A \Rightarrow B),$$

the transformation necessary for the logical subgoal is simple enough that SOLVETHMOPS does it. A new GOAL-type goal structure, a logical subgoal, is created with a formula

$$(H \wedge A) \Rightarrow B.$$

HAND-type goals are generated for the purpose of manipulating the hypotheses in order to prove the goal. The routine to process these type goals, HAND, is an example of a procedure that will reschedule the goal which it is processing for a later time. Typically, HAND may cause a number of logical subgoals resulting from backward chaining to be created and then will reschedule its goal (by assigning it a lower priority) for a later time. If the HAND-type goal reaches the head of the agenda again, HAND will do some more work on it, possibly generating some subgoals for forward chaining. But if in the meantime, one of the backward chaining subgoals is proved, the HAND-type goal is immediately woken up to respond to that.

3.3 Considerations for Priority Schemes

The priorities assigned to goals determine which goal gets worked on when. Because of this, it would seem that the assignment of priorities is all important. If the "right" goals (those on the path to the proof) get the higher priorities, the proof will be found easily. Of course, there is no general way to determine what these goals will be without already knowing the proof. Some "wrong" goals (not on a path to the proof) are almost surely going to be selected while searching for a proof no matter how the priorities are chosen. It is almost a definition that hard problems are those in which the number of "wrong" goals selected far outweigh the "right" goals.

Although the priorities are important, a significant amount of freedom can be allowed in the assignment of priorities without undue influence on the outcome of the proof. APRVR's priority system is still quite rough and untuned but it has performed reasonably well[2]. It is not too

[2]Because of this lack of tuning I will try to avoid referring to specific numeric values for priority manipulations and will instead discuss the strategies in assigning them. The reader is referred to the program if more specific information is desired.

difficult to understand why this is so. It should be obvious that the priorities are just a ranking of the candidate goals and that the scale of the priorities is unimportant. Under certain conditions described later, the actual ordering of the goals by the priorities is not too important. In a reasonable agenda system, the difference in the ordering of two goals whose priorities are close in value should not drastically affect the outcome of a proof. In practice, the problem turns out not that some "wrong" goals get too attractive a priority but that some "right" goals get too low a priority.

Obviously, the main objective in setting up a system of priorities is to give high priorities to those goals most probable to be on a path to a proof. However, since wrong choices are almost always made, another aim should be to minimize the cost of making a wrong choice. One simple way of doing this is giving some preference to a goal which will be solved or rejected quickly. Such a goal may be one in which there are few or no variables or one in which the number of logical connectives is minimal (the positions and types of which can also be an influence). Unfortunately, the proof attempts for such goals are not always guaranteed to terminate quickly because of interactions with the lemmas. Many other considerations like these may be added if more domain-specific knowledge is used.

Another way to minimize the resources spent on a possibly "wrong" goal is to limit the amount of work done on it. By including a **depth factor** in assigning priorities, the deeper a goal is in the proof tree, the lower its priority will tend to be. While this hurts the descendents of all goals whether they are off or on a path to a proof, the descendents of a "right" goal will probably make some progress towards a proof and so may overcome the depth factor. If they do not totally overcome it, they may at least partially reduce it. This is also one way to prevent the prover from following down an infinite, recursive path.

This depth factor can not be allowed to be too large. If it is dominant over the other factors contributing to the priority, the search for a proof will effectively be breadth first. In practice I have found that for difficult problems the depth factor may become significant at times, forcing a number of goals and their descendents to be searched in a breadth first fashion. Eventually, a breakthrough occurs and one goal becomes significantly better than the others.

The search for a proof tends to become breadth first when some goals are created that are similar in structure and have no features that are distinguished by the priority calculations. These goals and their descendants all get assigned priorities that aggregate together. If there are four separate but similar groups of goals, there are four proof trees being developed a step at a time in succession. This can be very frustrating to an observer watching the prover in action. In addition, it may even be that the four separate goals are four different intermediate steps in the proof all leading to the same result.

To improve this situation somewhat, an **inertia factor** is taken into consideration when choosing the next goal to be worked on. If the highest priority goal of the subgoals of the goal just worked on (or the goal itself) is sufficiently close to the overall highest priority goal, that goal would be chosen, thus keeping the prover proceeding in the same context rather than jumping to a different context. Following the prover as it works is easier when the context of what is being worked on does not change too frequently.

This can also be useful in getting a proof over a little hill where the priority of one goal on the path is not very attractive. If the hill is not too big or the inertia is strong enough, the proof will sail right over it. Unfortunately, there is nothing magical about the inertia; it behaves like inertia does in physics. If APRVR gets going in the wrong direction, the inertia is a disadvantage. If a hill on a correct path was too big, it is going to take extra work to get over it. Examples run through APRVR have shown both kinds of behavior but I have kept an inertia factor because of its improvement in an observer's perception of the flow of the proof.

3.4 The Components of a Priority

When a new goal is created it is given a priority that is a combination of an estimated cost of proving that goal (static estimation) and an estimation of how much work is left to be done once the goal is proved. These estimations are rather arbitrary and in no particular units and are really only used for the ranking of the goals. Thus an estimated cost for a goal of 100 does not mean that it is five times more difficult than a goal with an estimated cost of 20.

The estimation of work left to be done (the **TODO cost**) is based upon the parent's TODO cost plus an estimation of the cost of any goals to be proved by that parent when this goal is proved plus an overhead cost. It is this overhead cost that partially implements the depth factor discussed above. The deeper a goal is, the more overhead cost is included in the estimated work to be done.

Originally, APRVR kept a goal's priority as a single number but it became clear that this was too little information. When a goal was created only one number was stored in it as the priority. No record was kept of what portion of that was due to the static estimation of the goal and how much was passed down from its parent (the work left to be done). After a goal is created, some work may be done on it and then it is rescheduled, at a lower priority, to try some methods later that are less likely to be successful. This rescheduling masked any information contained in the original priority. APRVR now keeps the estimated TODO cost for a goal separate from the other factors entering into the priority but when the total priority of a goal is needed the TODO cost is combined with them to return a single value.

The static estimation for goals has also evolved. Originally it was an intentionally simplistic measure of the size of the formula to be proved as I was more interested in the dynamics of the priorities and was working with simple theorems. As harder theorems were attacked, the static estimation became more involved but can not be described as sophisticated. It now includes checking whether the formula is ground, the correspondence between function symbols in the hypothesis and conclusion, some checking for simple proofs, and, still, the sizes of the hypothesis and conclusion. Since the static estimation is done only once for each goal created, a thorough examination of the goal could be done. If domain specific knowledge (eg., models) is added, a more powerful static estimation routine can be built upon the solely syntactic one I am using.

3.5 Manipulations of Priorities

The priority given to a goal does not stay static once it is worked on. It is the dynamics of the priorities that makes the agenda system successful. The estimated cost (and therefore the priority) ascribed to a goal changes as more information is learned about it.

When a proof is found for a goal, its priority changes drastically. If the proof is a ground proof, that is, no variables need be substituted into the goal's formula, the goal is completely removed from consideration. No better proof can be found. Otherwise its estimated cost is multiplied by a factor to push the goal far down the agenda. This prevents the goal from being worked on further unless the total proof goes nowhere, in which case the goal will be called upon again in hopes of finding another proof. While this theoretically relieves the problem of trapping (and it does do that for simple cases), for any difficult problem, the chance of the goal being called again is remote. If APRVR had better knowledge of when a proof had the wrong substitutions then it could be wiser about rescheduling the goal. In some cases APRVR can determine that a proof is inconsistent with other goals and so can retry the goal to find a different proof.

When two or more goals generate the same subgoal, that subgoal's priority is adjusted to reflect that fact. If a particular subgoal is important on several different possible paths to the proof, its importance is increased because the multiplicity of the paths improves the chances that at least one will be successful. But the paths to the proof probably join at some point above this goal and so the number of parents of a goal may not correspond to completely independent proof attempts. It would take a complete analysis of the entire proof tree to sort out the duplication in the various paths. This is not cost effective as it would be expensive and the priorities are only rough estimates anyway. Clearly the priority of the goal should be no worse than that computed by the considering the path with the minimum estimated TODO cost and probably should be somewhat better. APRVR takes all this into account by computing the goal's priority by using an artificial TODO cost of two-thirds the minimum TODO cost passed down from the parents of the goal.

The priority of a goal changes as work is done on the goal. When created, a goal's priority is just a combination of its estimated TODO cost and its initial static estimation. As more processing is required for the goal, this estimated cost is increased and the priority lowered. Typically the routine called for a particular goal will try some methods of proving the goal and then reschedule itself at a lower priority (by increasing the goal's estimated cost) to try some more methods later.

HAND-type goals undergo this refiguring to the estimated cost. When a HAND-type goal is first processed, some of the more attractive backward chainings (if any) are done and the goal is rescheduled at a lower priority. The next time it is at the head of the agenda, any disjunctions in the hypothesis will be processed and the goal is rescheduled. Then come the forward chainings, the rest of the backward chainings, tries at contradictions, reverse backward chainings, and finally reverse forward chainings. As each of those steps is tried and the next is scheduled, the estimated cost of the goal is increased according to a factor associated with the likelihood that the next step will lead to a proof. Since the next step obviously has not yet been tried, this factor is only based on the general performance of the next method rather than anything specific for this goal. Once that next method is tried, any subgoals generated will better reflect whether the method may be successful for this goal.

Two types of adjustments are made to priorities. The most common one is to adjust the priority by multiplying the estimated cost of the goal by a constant factor. This linear adjustment has the property that the absolute value of the priorities does not matter. If a factor increases the estimate ten percent, it does not matter if that estimate had been 100 or 2000. As a proof proceeds the estimated costs of the active goals tends to increase. This does not affect these percentage adjustments to priorities.

Other adjustments are absolute adjustments. The overhead cost is an example of this. The overhead charged for creating a subgoal is always constant. As the proof attempt gets more involved, these factors decrease in importance relative to the percentage factors.

The values of these factors used to adjust the priorities are stored in variables in APRVR. I have always been concerned when others have said that their provers have a number of parameters that can be tuned, because these could be adjusted for each particular problem to get the best performance on that one problem. The parameters for adjusting priorities in APRVR were put in because I had very little a *priori* idea what they should be and I wanted to be able to modify them as it became clear that the present values could be improved. Usually I adjusted a parameter a number of times as I saw how the examples performed with it. After a few runs, the parameter had adequately converged to some rough value and it was not changed again unless some new factor was included which changed the relative performance due to this factor. Fine tuning (adjusting a factor by less than 25 be necessary. All the problems were run with the same parameter values.

3.6 Proof Methods Specific to Agendas

A hierarchical theorem prover like IMPLY has a conceptual elegance in that to prove a subgoal, it simply recalls itself and waits for a proof (or failure). There are few considerations necessary in manipulating subgoals; most of the details are hidden in the use of recursion. Since IMPLY is interactive, if it fails to prove a needed subgoal while using the default limits of efforts, the user can come to the rescue and help direct the proof.

APRVR gives up some of the advantages of IMPLY. The same conceptual elegance is still somewhat there in that subgoals are just added to the agenda, but there are many more details to be handled. APRVR was not designed to be highly interactive and so is designed to avoid cutting itself off from finding a proof, again adding to the complexity of the prover.

The agenda theorem prover has a number of advantages that outweigh these complications. The most obvious one is the ability to process the subgoals according to a priority ordering rather than working on them in a fixed ordering. Associated with this is the ability to suspend processing on a subgoal whenever the subgoal is no longer promising. Other advantages of using the agenda in APRVR include a more flexible ANDSPLIT and a non-local case split. When I first wrote the ANDSPLIT routine, it tried all the conjuncts as "first" conjuncts to generate the "first" subgoal. The idea was that by trying all orders, the best order would be included and so the proof would finish quickly. I was depending on the priority ordering to direct the search down the best path. Early experimentation showed that the potential combinatorial explosion from this method was not adequately limited by the narrowing effect of ordering by priorities. The ANDSPLIT mechanism was reverted back to the method in IMPLY; only one first subgoal (derived from the lexically first conjunct) is generated. This method is reasonable in that it considerably reduces the branching factor in the search and any proof that could be found starting from any of the other conjuncts can be found by starting with this conjunct.

The problem with looking at only one first subgoal is that others may lead to proofs more quickly. The difference is due to substitutions returned as proofs. One conjunct may restrict the binding of a variable to a particular variable while another may simply check that a certain property holds for that variable (or its binding). For example consider the conjunction

$$x*a = a*x \land x*a = e$$

where e is the identity element in group theory. The first conjunct may possibly have many proving substitutions, just one of which would be the only proving substitution of the second conjunct. If APRVR only looks at the subgoal generated from the first conjunct, it might first return the substitution $\{e/x\}$ which would cause the second subgoal to fail. APRVR could eventually find the right solution but it may involve unnecessary delays. If APRVR were smart enough to try the second subgoal first, finding that x should be the inverse of a, the temporary trapping could be averted.

The new feature added to APRVR to prevent this temporary trapping was to try all of the conjuncts as first conjuncts, as before. The difference now is that after a short while all the "first" subgoals (and their subgoals) that have not been proved are terminated. If none have been proved, the subgoal from the lexically first conjunct is not terminated. Those that have been proved continue and generate their associated second subgoals. (This may have the additional advantage of finding multiple alternative values for the substitution proving the conjunction, each starting from a different conjunct.) The effective branching factor of the search has not been increased much (since most of the branches were quickly cut) and there may be a considerable increase in efficiency in finding a proof.

A very interesting use of the agenda is the non-local generation of case splits. The motivation for this comes from a problem in which there is a very promising path for a proof which is blocked at the end by the lack of a specific fact. The proof would succeed if that fact were true but, unfortunately, this is only sometimes the case. This fact might trigger a person solving this problem to do a case split at the appropriate point in the proof, somewhere above the point at which the attempted proof failed.

Two different types of case splitting can be identified. Although these two are logically equivalent, they are different in terms of the motivations for doing the case split. The first is the more general: A proof may be split upon whether a proposition, A, or its negation, $\sim A$, is true. The difficulty with using this type splitting is in knowing what A should be. Every goal $H \Rightarrow C$ is a candidate trigger for this split by just noting that the goal is true if C (or some $C\theta$) were true. Then a case split could be done on some ancestor goal (parent, grandparent, etc.) of the goal in which one case is where C is true and the other case is where C is false. Of course, doing this type of case split often would lead to much extra work.

The second type of case splitting is more useful. If there is some fact known by hypothesis or lemma that is of the form

$$A \vee B \vee \ldots$$

then any goal whose conclusion, C, matches (by unification using a substitution θ) one of the disjuncts can trigger a case split at some ancestor of the goal on the cases $A\theta$, $B\theta$, ... (one of which would be just the case $C\theta$).

If there were a lemma or hypothesis that was just the tautology

$$X \vee \sim X,$$

this second type of case splitting would include the first type. Hopefully the available lemmas would not include such simplistic ones but instead would include those with more semantic importance.

Whenever APRVR is going to try a proof based upon the disjuncts of an OR in the hypothesis it generates an "ORSPLIT-H" goal. ORSPLIT-H accepts a formula of the form

$$((A \vee B) \wedge H) \Rightarrow C$$

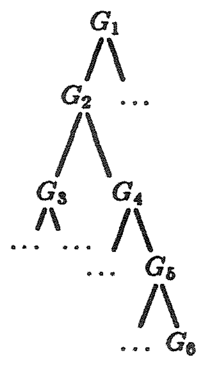
and generates a goal whose formula is

$$H \Rightarrow ((A \Rightarrow C) \wedge (B \Rightarrow C)).$$

ORSPLIT-H then attaches itself as a "daemon" watching the goal it created. This means that whenever that goal is worked on, APRVR wakes up the ORSPLIT-H goal to check on what has been accomplished. Specifically, ORSPLIT-H attaches itself as a daemon to subsequent subgoals until an ANDSPLIT is generated. When each of the subgoals corresponding to the conjuncts (which correspond to the disjuncts in the OR hypothesis) are created, ORSPLIT-H attaches itself as a parent to the subgoal. Then, whenever one of these subgoals is proved, ORSPLIT-H allows the normal proof attempt to continue but schedules itself for some point in the future. If the proof of the ANDSPLIT and, therefore, the proof of the ORSPLIT-H succeeds, everything proceeds normally. But if the ANDSPLIT has not been proved when ORSPLIT-H eventually wakens, APRVR will generate the cases based upon the disjunction in the hypothesis (after application of the substitution proving the subgoal of the ANDSPLIT) if those cases are ground (contain no variables).

Where in the proof tree should the case split be tried? It could be tried at the goal immediately above the triggering goal but the chances that the other cases could be proved there are slim. It could be tried at the original goal (the "theorem" being proved) but that probably would lead to a large amount of duplication of work.

APRVR chooses to back the case split as far up the proof tree as possible until a goal is reached at which backing up any further would reach a goal that would not necessarily be helped by doing the cases. The goal stopped at is provable, given the right case, by the path already taken. The parent goal would not necessarily be proved given that same case.



In the figure above, all branches shown are AND branches and all goals to the left of the branch upward from the present goal (G_6) are already proved while those to the right are not proved. If G_6 is the goal that triggers a case split, the cases would be applied at G_2. The case that proves G_6 would finish the proof of G_2 but not G_1. If G_2 could be proved for the remaining cases, the proof could continue at the right hand branch under G_1.

How much extra work is done if this is not the right place to do the case split? If the case split need not have gone back as far as G_2 but could have been done at G_4, extra work would have been done, but not much. Since G_4 was the point at which the cases had to be done, the proof of G_3 was acceptable without regard to the cases and did not need to be redone (which is not true if G_2 was the correct place to do the case split). So, in fact, very little extra work was necessary.

If, on the other hand, G_1 was the proper point for the case split, APRVR has not wasted its effort. G_2 would be proved by doing the split there. Later, when it was found that G_1 needed the case split, the G_3 branch would be already proved for each case.

There are surely other methods available to a prover with an agenda mechanism like APRVR's. More non-local methods might be found where information from one part of the proof might affect what is being done elsewhere. Perhaps if a goal was found to be false, this could be traced up the tree to the point at which a false ancestor of this goal was generated. Other similar goals might also be purged. The idea of a daemon goal watching other goals may prove fruitful. Also, since the entire proof tree is kept, if one goal is found similar (analogous) to another, the successful proof of one could be followed, correcting it wherever the similarity broke down.

CHAPTER 4.

IMPLEMENTATION

4.1 Introduction

APRVR is a fairly large system consisting of over 400 LISP functions and 100 flags and parameters. Most of these are support functions, such as the simple data accessing functions, the functions to manipulate the agenda, and user interaction functions. Other large groups include those for Skolemization, unification, and for the printing of proofs. Only about 70 functions implement the proof methods used.

To handle this large system a number of support systems are necessary. Listings of the program are cross referenced (by page of reference) and a separate LISP cross reference of the calling structure and global references are generated. A help system references this second cross reference to provide easy on-line documentation. It also displays the top level comments for the functions.

Only the compiled versions of most of the functions are in core, but simple commands bring in the interpreted versions, edit, or "pretty-print" them. Functions modified are noted and a simple command saves them all to a disk file.

Although LISP may not be as efficient as other languages in execution speed, its advantages for program development made it ideal for the development of APRVR. Its interactive capabilities allowed debugging in the middle of a proof attempt. I was able to do arbitrary computations and corrections on the code and to continue or restart (not necessarily at the beginning) a proof.

This chapter deals with the implementation of APRVR - the details that, while they are not major points, are important to the operation of agenda based theorem provers in general, and APRVR in particular.

4.2 Implementation of the Proof Methods

A number of the proof methods are described in Chapter 2 and will not be covered here. This will cover some of the details omitted in that chapter.

4.2.1 Goal Types

Goal structures have a field to specify the TYPE of the goal. This, in turn, specifies what routine is used to process the goal when it reaches the head of the agenda. In order to have APRVR prove a logical formula, a **GOAL**-type goal needs to be created for it. A number of things happen when a GOAL-type goal is to be created.

The formula is logically reduced and normalized into a canonical form. The formula and all its subparts are hashed into an array so that duplicate expressions originally occupying different memory locations become references to the same memory. This reduces the amount of memory used while speeding up tests of equality and membership applied to logical expressions.

If the formula to be proved is identical to that already in some other GOAL-type goal, the previous goal is used. If the formula is identical to a previous one up to a renaming of variables, a **RENAME**-type goal is generated, made a parent of the previous GOAL-type goal, and returned. Its only purpose is to rename the variables in any substitutions returned for the earlier goal.

In either of these two cases, if the previous goal had proofs a **DUMMYWAKEUP**-type goal is generated and put at the head of the agenda. When the next goal is picked from the agenda, it will be this one. This goal then "wakes up" (puts at the head of the agenda) the goal that just requested the creation of that GOAL-type goal. This indirection of using an intermediate goal is necessary to prevent the waking-up of the goal from interfering with the requestor's possible rescheduling of itself.

If the goal is new, further processing is done on it. Under certain restrictions, all possible forward chainings are done to it and added to the hypotheses. These restrictions demand that no forward chainings require substitutions for variables other than those in lemmas (and lemma-like hypotheses). Furthermore, no hypothesis or lemma may ever use an implication (or disjunction) to forward chain on if that implication had been used before in creating this hypothesis (or one of its ancestors if forward chaining had been used several times to create the hypothesis). All successful forward chainings are recorded in two lists: one records what was generated, indexed by each hypothesis used, and the other records what hypotheses were used, indexed by what was generated. This allows most forward chainings to be done rather quickly. Only new combinations of hypotheses need to be tested for the generation of new forward chains.

This forward chain processing (done by the routine **QKFC**) has the useful feature that it reaches a closure. Later attempts at forward chaining on its result would not generate anything new. By testing whether this hypothesis exists elsewhere, APRVR can use this feature to reduce greatly the running time to generate forward chainings on subsequent goals.

The advantage of doing all these simple forward chainings and updating the hypothesis to include them at this point is to eliminate the expense for doing them one by one later if they *might* be needed. If the forward chainings are added later, the problem exists of what should be done with all the work already done on this goal. By not having the additional hypotheses, some critical deduction may have been missed. But redoing all the deductions (and creating nearly identical goals along the way) is quite expensive. By doing the simpler forward chainings now, these problems are avoided. The more complex forward chains still need to be done later.

After the goal structure is created but before control is passed back to the routine requesting it, a quick attempt is made at proving the goal by a routine called **QKIMPLY**[1]. QKIMPLY does many of the manipulations that can be done on goals but only those that can be guaranteed to terminate quickly: ANDSPLIT, ORSPLIT, limited backward chaining, promotion of an implication in the conclusion, and the matching of the conclusion against the hypotheses. This is actually the only method to recognize that the conclusion matches one of the hypotheses. QKIMPLY is used to prove a simple goal when it is created rather than waiting until it reaches the head of the agenda. Substitutions for all the proofs that QKIMPLY can find are returned as a set.

When a GOAL-type goal is chosen to be worked on, SOLVEGOAL is called. SOLVEGOAL does no processing itself but checks its property list for the types of subgoals to generate and what priorities to give them. Normally it has only one type to generate, a **THMOPS**-type goal. The intention of SOLVEGOAL was to allow for room for later expansion of other methods to work on a goal. As an experiment, a specialized method called P->*, was recently added for the problems from *Principia Mathematica* to look for the existence of a lemma that would help prove the goal's formula. It was a trivial matter to get SOLVEGOAL to generate **P->***-type goals by placing P->* into a property list.

[1]This is essentially identical to the same routine in IMPLY.

While normally only the routine responsible for processing a particular goal type can add subgoals to that type goal, the routine that generates non-local case splits may add a subgoal to a GOAL-type goal. SOLVEGOAL never notices this except when that subgoal is proved.

As with all the routines to process goals, SOLVEGOAL must provide for its goal being selected after being woken up when one of its subgoals was proved. SOLVEGOAL, like most of the other routines, simply passes the proof up to its parents unchanged except for the addition of some information for later printing the proof.

A THMOPS-type goal is generated to try to find a proof by performing some manipulations based on the logical operators in the goal's formula. SOLVETHMOPS, the routine to do this, may spin off **ANDSPLIT** goals to handle an AND in the conclusion, **ORSPLIT** goals to handle an OR in the conclusion, or a **HAND** goal to work on the operators in the conjunctions of the hypothesis. SOLVETHMOPS may also generate GOAL-type goals for the subgoals of the simpler processing it does itself.

ORSPLIT-type goals are simple. The routine to process them simply generates a different GOAL-type goal for each of the disjuncts in the conclusion.

The processing of ANDSPLIT-type goals has already been discussed. It generates GOAL-type goals for its "first" subgoals and either GOAL-type or ANDSPLIT-type goals for its "second" subgoals depending upon whether there were two or more conjuncts.

The HAND routine to process HAND-type goals may attempt many different methods on the hypotheses to prove the goal. These methods do not appear as the type of goals; instead, the HAND-type goal handles all of them, keeping the name of the currently active method in the goal itself. The normal sequence of these is: **BC** (backward chaining), **OR** (looks for ORs in the hypothesis and generates an ORSPLIT-H-type goal), **FC** (forward chaining), **SBC** (secondary backward chaining), **CONTR** (looks for contradictions in the hypotheses), **RBC** (reverse backward chaining), and **RFC** (reverse forward chaining). HAND picks its methods from a list stored in the goal, which allows an alternative ordering on some goals to be CONTR, FC, RFC, BC, OR, SBC, and RBC when it appears that the goal would be more suited to finding inconsistencies in the hypotheses.

As each new method is chosen a subset of the hypotheses may be selected by a start-up function associated with the method. Thus BC is only done on those hypotheses returned from BC-STARTUP, allowing the prover not only to be selective about which hypotheses are worked on (with the others done by SBC) but also in the order in which they are done. BC-STARTUP sorts the candidate hypotheses into an order that roughly reflects the degree of matching between the conclusion of the hypotheses and the conclusion of the goal. The more non-variables that match, the closer the matching. The use of closer matching hypotheses increases the probability that the backward chaining will be productive.

When all the eligible hypotheses are chosen, the method is applied to each of them in sequence. If the method applies to a particular hypothesis and a subgoal is generated, the HAND routine suspends itself (with the goal getting a slightly lower priority) in case the new goal has a better priority. If not, the HAND routine processes the next hypothesis. When all the hypotheses are processed, the HAND-type goal is given a somewhat worse priority, according to the next method to be tried, and is rescheduled. When all the methods have been tried, the HAND-type goal is removed from the list of active goals, pending the proof of a subgoal.

As described earlier, when an ORSPLIT-H-type goal is generated, it will generate subgoals to do an ANDSPLIT. Under certain conditions it may also generate a CASESPLIT-type goal and add it as a subgoal to some GOAL-type goal.

Because the proof methods were different for the problems from *Principia Mathematica*, different methods were used in HAND. These are: BC*, BCX*, FC*, SBCX*, SBC* and are similar to the normal methods of BC, FC, and SBC. See Chapter 5.2 for more details.

4.2.2 Priority Details

Priorities in APRVR are usually numeric, normally ranging from 1 upward to INFINITY. The priority of a goal is associated with its estimated cost, so a goal whose numeric priority (cost) is 1 has the highest priority while a goal whose priority is the special symbol "INFINITY" is the lowest priority. A goal with a priority of infinity is said to be "asleep". Nothing will be done on it unless one of its subgoals gets proved and "wakes it up". Whenever a goal is proved it wakes up its parents by negating their priorities. This puts the parent goal at the front of the agenda ahead of all the normal goals, insuring that it will be worked on quickly.

Once a goal is proved, the likelihood of its generating other, more useful, proving substitutions drops. APRVR "slows down" the goal by multiplying its priority by a factor. This lowering of priority is filtered downward through all the descendants of the goal, taking care that no goal is slowed down more than once.

If the proving substitution for a goal is the empty substitution, then no other solution would be better. In this case, the goal is "killed" rather than being slowed down. If a goal is flagged as killed, no work will ever get done on it. Furthermore, it tries to "drug" its sons.

This involves removing its support for the attempted proof of the subgoal. If no other support exists for that subgoal, it is "drugged" and tries to drug its subgoals. A drugged goal is just a goal for which no active goal is requesting its proof and so is removed from active consideration. If another goal later generates the drugged goal as a subgoal, the goal and its drugged descendants are revived. The ANDSPLIT routine drugs its subgoals that it wants to terminate.

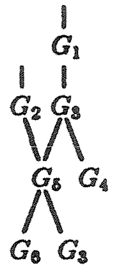
4.3 Multiple Parentage

One of the features of APRVR is the uniqueness of each GOAL-type goal. Each of these is essentially a theorem to be proved by itself; during proofs, almost no consideration is made of what the ancestry of a goal is or whether it is the top goal or not. A second request for the proof of a particular subgoal by a second goal results in the reuse of the original subgoal (which records that it has the two parents to which to return its values).

The scheduling of a goal getting an additional parent has to be adjusted. It appears to be a critical subgoal; if it could be proved, more than one possible path to the total proof could be advanced. After analyzing the situation and trying several different strategies, I decided on the strategy discussed earlier.

The TODO cost of the subgoal is set to two-thirds of the minimum of the TODO costs passed down from its active parents.

A more serious problem arises when the parent subgoal graph has loops in it. Note that G_3 occurs twice in the following subgraph of a proof tree:



If G_3 is proved via G_4, it will pass its values up to both G_1 and G_5. G_5 passes the value up to G_2 but can not pass that value back up to G_3. The goals used to generate a proof are explicitly recorded in the proof to allow APRVR to check for this type of recursion. An alternative solution would be to prohibit loops from ever being generated but if G_3 had not been allowed to become a subgoal of G_5, a proof of G_2 via G_4, G_3, and G_5 would not have been possible.

The possibility of loops causes problems in a number of places. As stated, APRVR must refuse to pass a proof up from a goal to its parent if that parent was involved in the proof of the goal. A similar problem occurs when the goals below a proved goal have their priorities adjusted (or are drugged) because of that proof. When an ANDSPLIT terminates and drugs its "first" subgoals that have not been successful, it encounters the same difficulties and must avoid them. When ORSPLIT-H looks back up the proof tree to do a CASESPLIT, it must also be aware of the problem.

4.4 Human Interaction

One of the strongest points of IMPLY is its man-machine interface. IMPLY is not really designed to be run without a person controlling it. By using some of the many powerful commands available to him, the user can guide the prover towards a successful proof.

Although I wanted a strong human interface in APRVR, I found that an agenda theorem prover was not as convenient for this as a recursive theorem prover. While there was an obvious point where interaction could be added, it was not clear when APRVR should allow (force) the user to interact. The obvious point was in the routine LOOPER that just loops picking the best goal off the agenda and applying the proper routine to it. APRVR could stop to interact with the user just before it chooses the next goal to be worked on, but this occurs too often to be done every time.

Rather than APRVR deciding when the user might want to interact, APRVR allows the user to interrupt it. Every time a goal is chosen, LOOPER checks to see if the user has typed something. If not, APRVR continues on its way without disturbing the user. If so, APRVR interrupts what it is doing and calls the LISP interpreter to give control to the user. When the user finishes, control is passed back to LOOPER which continues as though nothing had happened unless the user has changed the data base. In that case, depending on what the user did, LOOPER may pick a different goal to work on.

Before the user can know when to interact with the system, he needs to know what it is doing. A trace of the proof attempts is printed with explanatory comments as the proof proceeds. Each goal creation is displayed, as is every goal that is chosen from the agenda to be worked on. Whenever APRVR changes context by choosing a goal not closely related to the previously chosen goal, the first few items on the agenda are displayed with their priorities. Logical expressions such as the conclusions of the goals are displayed in infix notation by a "pretty printer" that can limit the detail shown to a particular depth. Example output of a proof trace can be found in Appendix B.

Normally a user will watch the flow of the proof without interrupting it until some interesting event happens. At that point he may wish to back up to before the event so he can watch the proof more closely or check on what the state of the proof was before that event happened.

Since IMPLY is recursive, it is very easy to back up to an earlier point in a proof by popping up recursive levels to where the proof could be continued from just before the point desired. This is not possible in the agenda based control structure of APRVR. Interaction points always occur at the same level; no recursion saves the state information. All the changes since the previous interaction point have already been made to the data base.

Because the ability to back up and retrace the steps in a proof are very important to developing the prover, I implemented an UNDO facility in APRVR. The functions which modify the global data structures used in APRVR were changed to use structure modification functions that record the information that would be lost by doing the modification so that recreation of the previous state is possible. This information is then recorded onto a list. APRVR keeps an index into the list where the interaction points were, indexed by the goal that was next created after that point. If I had a question related to the performance on a particular goal, I could just tell APRVR to back up to just before the creation of that goal (or one of its subgoals, if that was where the problem was). APRVR would traverse the list of saved information, undoing all that had been done since that point. Afterwards, APRVR would still be at the user interaction point but the data base would be reverted back to its earlier state. The proof could be continued and would proceed as it did before. LISP breakpoints could be set to provide a closer view of what was happening within APRVR.

Obviously the retention of the information to UNDO earlier data modifications requires memory, but because of the memory sharing of the formulas, the memory requirement was not as much as might be expected. The amount of memory required for each interaction point is on the order of 100 words. This allows a fairly long memory, depending on how many words the user is willing to devote to it. I found that most of the time I wanted to back up only a few interaction points.

Now that the user can interrupt the prover and back it up, what can he do during his interaction? He may wish to examine the state of the proof. One routine, FRONTIER, prints out the proof tree. By using keywords, he may specify that he wants more or less information displayed. Other routines are available for printing more detailed information about any of the goals. Although the proof printing routine was designed for printing the complete proof of a theorem, the user can invoke it on any proved goal. This provides more detail on how a proof was achieved than is available from watching the trace of the proof as it develops. Example outputs of the proof printer are in Appendix B.

In order to affect the flow of the proof, the user must modify the data base. Although it is simple to move a goal to the front of the agenda so it gets processed next, APRVR is likely to create its subgoals with priorities approximately the same as the goal had before it was moved to the head of the agenda. Therefore, they would not be at the front of the agenda as desired. So, unless the goal will be proved when it is next processed, this user interaction may not help much.

A more useful manipulation is to **concentrate** the search. By reducing the active goal list to only the descendants of a particular goal or just to a single goal, the search for a proof can be reduced considerably. Similarly, all the descendants of an attractive, but untrue, goal can be drugged so as to stop the attempts at proving it.

APRVR is now powerful enough to prove the examples in Chapter 5 without any interaction.

However, during the development of the prover, the interaction facilities were used extensively to monitor the proof and also to bypass uninteresting dead ends to concentrate on whatever techniques were being developed. Harder problems in the future may require the use of the interaction facilities in order for any proof to be found.

CHAPTER 5.

PROBLEMS AND RESULTS

5.1 Introduction

My intention in doing this theorem prover was to develop an agenda-based natural deduction theorem prover and to experiment with it and the assignment of priorities. I did not enter into this to prove any specific theorems by any particular methods designed for them. I was attempting to do something more general; I was not planning on advancing the frontiers of the theorems proved by automatic theorem provers in the present system[1].

That could come later when more specialized heuristics could be added that would be comparable with what is available in other theorem provers. This system is just an initial testbed to determine the basic advantages and problems of agenda systems.

5.2 *Principia Mathematica*

I chose *Principia Mathematica* (PM) [Whitehead-70] as the domain for the early developmental work on APRVR. PM, the classic logical tome of Whitehead and Russell, had a number of advantages. It is a well laid out system with the axioms and proof rules explicit. The problems are presented as logical statements and generally proceed from simpler to harder. In the early theorems, in chapter *2, there are no function symbols, other than the logical ones, to require special handling.

PM was not without its disadvantages. The theorems are not impressive in appearance. Without adhering to the logical presentation in PM and letting the prover treat the propositions in *2 as it would normally, the theorems are immediately proved. This required rephrasing the theorems by replacing the logical connectives with different symbols that were not already built into APRVR. Then, some of the already existing deductive machinery had to be duplicated to provide deduction capabilities allowed in PM. While this went against my desires not to build

[1]However, see Chapter 5.5 on AMS.

special machinery in this version, the special deductive machinery was very similar to what already existed and so the results of experimenting could be adopted.

In PM, the distinction between implication and disjunction is notational only. This was reflected in the the matching routine in APRVR and caused no problems. The definitions in the next chapters, *3 and *4, of conjunction and equivalence appeared likely to present difficulties to my system. Conjunction and disjunction are handled quite differently in my prover for semantic reasons. It would be out of the spirit of natural deduction and would be a burden to transform back and forth between the equivalent forms of AND and OR all the time. Since I was only interested in working on agendas and not verifying PM by computer, I avoided these problems by only choosing examples in *2. I feel that translating between forms of AND and OR rarely occur in most theorems.

There were two major rules of inference, detachment and backward chaining (see Chapter 2.5). Forward chaining (for PM) was not added until later. APRVR normally keeps the hypothesis and the conclusion of a goal separately (with the implication implicit), but since the logical operators of PM had to be disguised from APRVR, the goals were kept intact in the slot normally containing just the conclusion. Because of this, the implementation of detachment is actually identical to APRVR's normal backward chaining. The backward chaining for the PM problems, although identical in concept to the normal backward chaining, had to be specially implemented. In the notation of PM, detachment is justified by *1.11 while backward and forward chaining are justified by *1.11 and the principle of the syllogism (*2.05 and *2.06).

I did not try to do a complete run of all the theorems in *2. By default (so as to fit in physical memory while on our DEC-10 computer), APRVR was limited to 150 goals (about 50-75 logical goals) during each problem. Some theorems could not be proved under this limitation, others required limiting the number of previously proved propositions available during a proof, while most were proved with all previous propositions available as lemmas. When I later added forward chaining, a number of the proofs became easier. APRVR proved 47 out of about 55 tried.

I was satisfied with the results. APRVR's success on these theorems was adequate and I had developed the initial structure of APRVR and was ready to try new fields. Meanwhile, using the same deductive techniques (originally without forward chaining) available to the Logic Theorist

[Newell-57], I had been approximately as successful in proving the theorems. Later, when I added forward chaining, I proved the most difficult problem for LT, *2.17, rather easily. As a test in adding domain specific techniques, I also recently added a procedure that looks for a "lemma" to be used in proving a goal in this domain. This showed the modularity of APRVR; it was quite easy to add such a procedure without modifying the other procedures. It was only necessary to write the one procedure to do the proposing and calling the other routines to do the proving, another one to print its part of the proof, and then modify a property list indicator to tell APRVR to include that procedure. This procedure allowed APRVR to find the proof for a theorem on which the Logic Theorist totally failed, *2.13. The proofs of *2.13 and *2.17 are in Appendix B.

5.3 Set Problems

The next set of problems presented to APRVR were a list of elementary set theory problems. At first APRVR did poorly on them. In analyzing why, I realized that these problems involved developing several ground facts from the hypotheses via the axioms. Forward chaining in APRVR had been designed with the idea that it was not often needed and that backward chaining could usually be used. When forward chaining was really needed, it would have to wait its turn behind the more commonly used backward chaining.

These problems pointed to a mistake in that logic. The problems needed not one, but several, simple forward chainings. In my system this would mean a delay until the first forward chaining was done and then a further delay until the next one was done. While this might be acceptable for some complex forward chaining, it was intolerable for these simple ones.

The solution to this was to do some kind of limited forward chaining early in the problem. Limitations had to be imposed to prevent forward chaining from continuing forever recursively. Recursive forward chainings would be prohibited and only ground chainings could be done. This is exactly the type of simple forward chainings that were desired but also prevents any complications, with the goal or the substitutions returned in its proof, arising from any substitutions for forward chaining. This "quick" forward chaining routine, QKFC, is described in Chapter 4.

The results on these problems look impressive but it should be realized that they do not reflect the amount of effort spent in QKFC, which was considerable. On the whole, QKFC, even though it might be expensive to use, is a worthwhile tool, saving many times its own expense.

The lemmas used in the set problems follow:

$$x \subseteq y \land y \subseteq x \Rightarrow x = y$$

$$\forall z(z \in x \Rightarrow z \in y) \Rightarrow x \subseteq y$$

$$x \in a \land x \in b \Rightarrow x \in (a \bigcap b)$$

$$x \in (a \bigcap b) \Rightarrow x \in a \land x \in b$$

$$x \in a \lor x \in b \Rightarrow x \in (a \bigcup b)$$

$$x \in (a \bigcup b) \Rightarrow x \in a \lor x \in b$$

$$\sim(x \in a^c) \Rightarrow x \in a$$

$$\sim(x \in a) \Rightarrow x \in a^c$$

$$x \in a \Rightarrow \sim(x \in a^c)$$

$$x \in a^c \Rightarrow \sim(x \in a)$$

$$\sim(x \in \phi)$$

Results

Theorem	Total Goals	GOAL-type Goals	Goals Proved
$\phi \bigcap a = \phi$	19	10	19
$a = a \bigcap a$	21	11	21
$a \bigcup b = b \bigcup a$	43	21	25
$a \bigcap b \subseteq a$	7	4	7
$a \bigcap b = b \bigcap a$	21	11	21
$\phi \bigcup a = a$	31	16	28
$b \subseteq (a \bigcup b)$	11	6	11
$a \bigcup a = a$	25	13	25
$(a^c)^c = a$	17	9	17
$(a \bigcap b)^c = (a^c) \bigcup (b^c)$	50	25	38

5.4 Group Theory Problem

One of the classical group theory problems in automatic theorem proving is the proof that a group is Abelian (commutative) if the square of every element is equal to the identity element. Because the associativity axiom is used three times on partially instantiated multiplications, this problem is a good test of the system's design in avoiding inefficiencies in finding substitutions for free variables.

APRVR did well on this problem. The only additional improvement to APRVR that came from trying this problem was in the way ANDSPLITs are handled. Originally, when faced with an ANDSPLIT, APRVR only tried proving the first of the conjuncts, leaving the others until that one was finished. This reduces the branching factor considerably as any proof that could be found starting from one of the conjuncts could be found by starting with any other conjunct. But when there is more than one substitution proving an ANDSPLIT, the conjunct chosen as the initial one to prove may make a difference in the efficiency of finding the required substitution. If that conjunct lends itself to an easy proof, but with a different substitution than the one needed, the prover may be temporarily trapped while it tries that first solution.

The new feature added to APRVR to prevent this temporary trapping was to try all of the conjuncts for a quick proof. For each conjunct proved, the remaining conjuncts have to be proved with that substitution. Thus while several proofs of the conjunction might be going on simultaneously, it will only be so if several conjuncts have been proved. If none of the conjuncts could be proved quickly, APRVR reverts back to trying to find a proof only of the first conjunct. The branching factor had not been increased much (because only quick proofs were tried) and a considerable increase in efficiency occurs when an ANDSPLIT returns multiple values.

Because equality is not implemented in APRVR, expressions of the form $X*Y = Z$ had to be encoded as predicates, $P(X,Y,Z)$. The associativity axiom then had to be expressed as a pair of implications. Expressed in this form, the properties of groups that are needed are:

$\forall xy\, \exists z\, P(x,y,z)$	Closure under multiplication
$\forall x\, P(x,E,x)$	Right identity
$\forall x\, P(E,x,x)$	Left identity

$\forall x\, P(x, I(x), E)$	Right inverse
$\forall x\, P(I(x), x, E)$	Left inverse
$\forall xyzv_1v_2w(P(x,y,v_1) \wedge P(y,z,v_2) \wedge P(v_1,z,w) \Rightarrow P(x,v_2,w))$	Associativity
$\forall xyzv_1v_2w(P(x,y,v_1) \wedge P(y,z,v_2) \wedge P(x,v_2,w) \Rightarrow P(v_1,z,w))$	Associativity

and the theorem is simply

$$\forall x\, P(x,x,E) \Rightarrow \forall xyz(P(x,y,z) \Rightarrow P(y,x,z)).$$

In order to prove this, APRVR requires 228 goals of which 105 are GOAL-type goals. The proof involves 35 of the goals.

5.5 AM8

A mathematician would have found the previous examples rather simple. The problems are indeed simple but I think he would feel that way partly because he is very familiar with the problem domains and can use methods (possibly extralogical) not available to the prover[2]. These methods allow him to reduce his search space by determining what axioms and theorems are likely to be used, what bindings will be used for the variables, and possibly some lemmas that will cause the search tree to be shallower.

The hardest of the examples for APRVR is a problem in linear real inequalities, called AM8. It is a derivative of the problem to prove that a continuous function attains its minimum over a closed region. Even knowing about inequalities and knowing what the problem is about, a human prover will not find the solution quickly.

APRVR is not knowledgeable about inequalities and certainly can not understand what the problem means. The lemmas for inequalities are simply uninterpreted predicates to APRVR. I believe a person given the problem in the form that APRVR gets would not be able to solve it. This form is:

[2]For example, finding a rough proof of the $x*x = E$ group theory problem is easy if cancellation is used. But cancellation is not immediately obvious from the axioms and lemmas given to the prover. If it were knowledgeable about groups and had specialized heuristics and procedures such as cancellation, then it should have little difficulty with that problem.

Lemmas:

$$\forall xy[(P(x,y) \wedge P(y,x)) \Rightarrow P(F(x),F(y))]$$

$$\forall x\, P(x,x)$$

$$\forall xy[P(x,y) \vee P(y,x)]$$

$$\forall xyz[(P(x,y) \wedge P(y,z)) \Rightarrow P(x,z)]$$

$$\forall xy[\sim P(x,y) \vee \sim P(y,x) \vee (P(x,y) \wedge P(y,x))]$$

Problem:

$$\begin{aligned} &\forall t\, [\sim P(t,L) \Rightarrow P(F(L),F(t))] \\ &\wedge \forall x\, [\sim P(L,x) \Rightarrow \exists t\, [P(t,x) \wedge \sim P(F(t),F(x))]] \\ &\wedge \forall w\, \exists g\, [P(F(g),F(w)) \wedge \forall x\, [P(F(x),F(w)) \Rightarrow P(g,x)]] \\ \Rightarrow\, &\exists u\, \exists t\, P(F(u),F(t)) \end{aligned}$$

Doing an undirected search for the proof of this problem would be hopeless. But, if the variables were instantiated with the correct values, the problem might become solvable.

When APRVR starts trying to prove the problem, it finds a promising path to follow. After a while, the problem is reduced to proving something that (although APRVR can not know it) is not always true. In trying to prove this, APRVR finds a disjunction in the lemmas for which two of the three disjuncts would finish the proof. APRVR then does a case split. It looks back up the proof tree for an appropriate spot, in this case, the original goal, and does the case split there.

By noticing it had a partial proof, APRVR was able to use the indicated substitutions and reduce the problem to three other problems each with its own additional hypothesis, two of which are easily solvable. If this case split is necessary for the proof (as it was on this problem), APRVR has effectively reduced the search space by binding some variables. Even if it has guessed wrong, the search space is increased only linearly. During the proof of AM8, two other case splits are tried besides the one leading to the proof.

The successful case split involves three separate cases: whether the Skolem constant $G(T'(L))$ is less than, greater than, or equal to L. The Skolem function G is from the third hypothesis while T' is the Skolem function for the variable t in the conclusion. Two of the cases are trivial to prove. The third is an impossible case. When APRVR tries that case, it finds that QKFC generates a contradiction in the hypothesis and so reports the goal is proved.

CHAPTER 6.

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Summary

Theorem proving is the search for a set of deductions that leads from the axioms, lemmas, and theorems to the problem under consideration. Since the British Museum Algorithm of applying the proof rules to every possible combination of already proved facts is very ineffective, some heuristics must be applied to try to guide the search in the right direction. Although many theorem provers may have these heuristics to order the search embedded in their code, this dissertation explores an **agenda** method for making this ordering explicit and free from restrictions imposed by the control structure of the theorem prover. But freedom implies responsibility. To be free to chose the order of search means the choice must be explicitly made.

APRVR is an agenda-based theorem prover. The problems of determining the **priorities** on which to base the choices as well as the advantages of having the freedom of a less restrictive control structure have been experienced during its development. From this different view of theorem proving have come some fresh ideas for finding proofs.

Although this theorem prover was only meant to be a vehicle to explore priorities and agendas, APRVR has performed quite well on the problems it has been given. Each new problem has provided more insight into the agenda system, resulting in a change that has added power to the prover. The system is not completed; future problems should provide more ideas on how to strengthen the system by better utilization of the agenda.

6.2 Conclusions

Designing and working with APRVR has lead me to a number of conclusions, some of which are general in nature and some specific to agenda systems.

1. The iterative, as opposed to recursive, choice of subgoals leads to new methods of finding proofs. The concept of trying to prove each of the conjuncts, but only for a short while, in order

to find the best conjunct of an ANDSPLIT to start with would be difficult and expensive without an agenda system. It would be impractical to try to use information developed in one part of the search tree to affect the search in a different part of the tree, especially if the area to be affected is above the part causing the effect. This can be accomplished in an agenda system with little difficulty.

2. The determination of priorities is difficult. There are many factors that can enter into a priority of a new goal: how likely is it to be provable, how difficult might it be to prove or disprove, how likely is it to be part of the total proof, how much work would remain after it is proved, does it have multiple proofs, would its single or multiple solutions increase later work, would it bind variables to restrict the search later. As work is done on the goal, the priority should be adjusted to reflect how its partial successes or failures affect the likelihood of its being proved and how the ways to prove the goal have decreased. All these factors must be somehow weighted and combined so the goals can be compared to determine the best goal.

3. Although the priorities are important to the efficiency of the system, they are not critical. If certain guidelines are followed in assigning priorities, the specific priorities assigned to goals are not overly important. A goal with a slightly worse priority will still be processed, only somewhat later. As a result, the computation and assignment of priorities can be done somewhat loosely and still be reasonable. The only significant differences will appear at the limits of the prover's capabilities.

4. Completeness is only a theory. In determining how to assign priorities, a theorem prover designer must realize that no matter how well designed his system is, it only has a finite amount of resources with which to search an infinite tree. By the assignment of priorities, he can and must choose where the incompleteness will occur.

5. The overhead in using an agenda system and assigning priorities is significant. This does reduce the actual number of deductions made but should also reduce the number necessary. On simple problems, the agenda prover may not be as fast as some other types of provers. On more complicated problems, the agenda mechanism may be the difference between finding a solution or not.

6. There are difficulties in using a system in which common logical subgoals are shared by

their parents. I feel that the simpler conceptualization of having unique goals outweighs the added bookkeeping.

7. The human interface, even in an almost totally automatic theorem prover, is important. It is important for developing the system, knowing it works correctly, and allowing the user to help when necessary.

8. The priority system, taken by itself, might cause the prover to jump from one branch of the proof tree to another rapidly. Overlaying the priorities with some system of giving inertia to a proof branch causes the prover to appear more stable and allows it to hurdle small barriers to the proof.

9. This system does not begin to exhaust the possibilities of agenda theorem proving. I feel that only the surface has been scratched and that future systems should provide significant advances in automatic theorem proving.

6.3 Future Directions

There are many possible future directions to be taken with APRVR and agenda theorem proving.

APRVR does all the limited forward chaining for a goal when it is generated and adds the results as hypotheses. I feel there ought to be some way to distinguish these from the original hypotheses and not use them for some of the later deductions. They should be available to be matched, but not necessarily used as chaining hypotheses. By making them second-class hypotheses, the branching factor of the search should decrease.

Although the parameters for the combining of the priority factors are not critical, the present parameters could provably be improved by running APRVR on a larger class of problems and seeing how it performs.

More importantly, if a particular domain is chosen, the syntactic judgements of the provability of a formula could be supplemented by semantic, domain specific knowledge. I believe this would drastically improve APRVR's performance and might also force a better understanding of how to determine the plausibility of a formula.

I feel for certain that there are many more non-local methods in which the exploration of one part of a proof tree provides insight into what should be tried in another part. An example of this might be where two goals may be proved similarly. Perhaps the proof of one could be a guide for the proof of the other or perhaps the two proofs could even be attempted as a single attempt, sharing common deductions and differing only where necessary.

There is a great deal of independence among the goals on the agenda which could be exploited on future parallel computers. Different CPUs would take different goals, process them, update the data base, and then pick the next goal off the agenda. This n-fold increase in speed might be somewhat decreased by conflicts in accessing the database but a significant speed-up might be possible. Unfortunately, theorem provers run into problems with combinatorial explosions. This is not resolved by a linear increase in speed.

The human interface in APRVR can be improved. There is too much detail being presented to the user, distracting him from the important points. A better display of the status of the proof could be made but it would need to have some mechanism for letting him view the overall structure while seeing the details of what he is interested in. He might be given better access to control the overall priority mechanism. Perhaps he could say which hypotheses are critical and which are not. He could also warn the computer of traps it might fall into.

I certainly hope that I will be able to continue working with APRVR or a descendant and I expect that others will begin to work with priority-ordered agenda theorem provers.

APPENDIX A.

GENERALIZED SUBSTITUTIONS AND SOUNDNESS[1]

A.1 Introduction

APRVR is designed to prove the validity of formulas in first order predicate calculus. This appendix will show that if the prover returns a substitution for some formula then that formula is valid.

When a closed formula, E, is given to the prover, it Skolemizes the formula into an open formula, S. If $v_1, v_2, \ldots, v_n$ are the free variables in S, then the original formula E is equivalent to

$$\exists v_1 \exists v_2 \ldots \exists v_n S. \tag{1}$$

If some substitution θ exists such that $S\theta$ is ground (contains no free variables) and $S\theta$ is true then (1) is true. Likewise if there is some set of substitutions $\theta_1, \theta_2, \ldots, \theta_m$ such that

$$S\theta_1 \vee S\theta_2 \vee \ldots \vee S\theta_m$$

is both ground and true then (1) is true.

It will be shown that if APRVR returns some non-NIL substitution θ as the result of proving a goal S_2[2] then for any substitution σ such that $S\theta\sigma$ is ground then $S\theta\sigma$ is true. From this it follows that (1) and the original input are true.

A formula, P, will be termed **ground-true** if for any substitution σ such that $P\sigma$ is ground, $P\sigma$ is true[3]. This will be indicated by the notation

$$\forall\sigma\, P\sigma.$$

[1]This appendix is an adaptation of Appendix 3 of [Bledsoe-78]. The concept of generalized substitutions is originally due to W. W. Bledsoe but APRVR is the first theorem prover to employ it. I developed and proved the theorems contained in this appendix. Similar material also appeared in [Tyson-79].

[2]In this appendix, I will speak of substitutions as being the value of a goal. In the implementation, information other than the substitution is also returned but it has no relation to the validity of the proof and so will be ignored here.

[3]Equivalently, P is ground-true if and only if $\forall v_1 \forall v_2 \ldots \forall v_n P$ is true where $v_1, v_2, \ldots, v_n$ are the free variables in P.

Lemma 1. Properties of Ground-Truth. If P and Q are any formulas and θ and λ are any substitutions, then

(1.1) $\forall\sigma\, P\sigma \Leftrightarrow P$ when P is ground.

(1.2) $\forall\sigma\, P\sigma \Rightarrow \forall\sigma\, P\theta\sigma$.

(1.3) $\forall\sigma\, P\sigma \Rightarrow \forall\sigma(P\theta \wedge P\lambda)\sigma$.

(1.4) $\forall\sigma\, P\sigma \Rightarrow \forall\sigma(P\theta \vee P\lambda)\sigma$.

(1.5) $\forall\sigma(P \wedge Q)\sigma \Leftrightarrow \forall\sigma\, P\sigma \wedge \forall\sigma\, Q\sigma$.

(1.6) $(\forall\sigma\, P\sigma \vee \forall\sigma\, Q\sigma) \Rightarrow \forall\sigma(P \vee Q)\sigma$.

(1.7) $(\forall\sigma\, P\sigma \wedge \forall\sigma(P \Rightarrow Q)\sigma) \Rightarrow \forall\sigma\, Q\sigma$.

Remember that to show ground-truth it need only be shown that $P\sigma$ is true for all σ such that $P\sigma$ does not contain free variables.

To show that the prover is sound it is necessary to show that the rules used in APRVR are sound. Induction will be used where the prover creates subgoals. Most of the rules used permit straight forward proofs. For instance, the proof method that handles splitting when a disjunct is found in the hypothesis requires the proof that

$$\forall\sigma((A \Rightarrow C) \wedge (B \Rightarrow C))\theta\sigma \Rightarrow \forall\sigma((A \vee B) \Rightarrow C)\theta\sigma. \tag{2}$$

The left hand side is the induction hypothesis, i.e., that the subgoal $((A \Rightarrow C) \wedge (B \Rightarrow C))$ is proved by APRVR with a substitution θ such that $((A \Rightarrow C) \wedge (B \Rightarrow C))\theta$ is ground-true. The conclusion of (2) is what must be shown: that the returned substitution θ must be such that the original form, $((A \vee B) \Rightarrow C)$, is ground-true after θ is applied. Lemma 1.7 and the fact that the two forms in (2) are equivalent proves (2).

The rules concerning ANDSPLIT, forward chaining, and backward chaining are more difficult to prove. In applying these methods, two subgoals are proved, the values of which may be combined into a generalized substitution, a generalization of the notion of a substitution.

Normally, APRVR can avoid the extra expense of returning (and later applying) generalized substitutions when it composes substitutions and can instead use ordinary substitutions. The next section will show the soundness of APRVR when it uses ordinary substitutions. The following sections will define generalized substitutions and show the soundness of APRVR for those cases in which they are needed.

A.2 Soundness Results for Ordinary Substitutions

When there are no complications arising from conflicts in the substitutions returned from subgoals, APRVR will return ordinary substitutions. In this section, APRVR's ANDSPLIT will be proved sound when ordinary substitutions can be returned. The equivalent theorems for backward chaining and forward chaining are similar.

When APRVR encounters a goal of the form

$$H \Rightarrow (A \wedge B),$$

it generates a first subgoal of

$$H \Rightarrow A.$$

If this goal returns an ordinary substitution θ, then a second subgoal

$$H \Rightarrow B\theta$$

is generated. If this goal returns the ordinary substitution λ and it does not conflict with θ, then APRVR returns the composition of the substitutions, $\theta\lambda$, as the value of the original goal.

Before proving this is sound, it is necessary to formalize the concept of substitutions.

Definition. A **substitution** θ is a set $\{a_i/x_i{:}1 \leq i \leq n\}$ where the x_i's are variables and the a_i's are terms and $a_i \neq x_i$ and $i \neq j \Rightarrow x_i \neq x_j$.

Definition. If A is an expression and $\theta = \{a_i/x_i{:}1 \leq i \leq n\}$ is a substitution then $A\theta$ is the expression obtained by replacing all the x_i's in A by the corresponding a_i's.

Definition. A **composititon**, $\theta\lambda$, of two substitutions $\theta = \{a_i/x_i{:}1 \leq i \leq n\}$ and $\lambda = \{b_i/y_i{:}1 \leq i \leq m\}$ is defined to be the set

$$\theta\lambda = \{a_i\lambda/x_i{:}1 \leq i \leq n\} \bigcup \{b_i/y_i{:}1 \leq i \leq m \wedge 1 \leq j \leq n \Rightarrow y_i \neq x_j\}$$

A composition is clearly a substitution. Composition is associative: $(A\theta)\lambda = A(\theta\lambda)$

Definition. The **domain** of a substitution $\theta = \{a_i/x_i{:}1 \leq i \leq n\}$ is the set $\{x_i{:}1 \leq i \leq n\}$. The **range** of θ is the set $\{a_i{:}1 \leq i \leq n\}$. We will say a variable occurs in the range of θ if it belongs to the range or if it occurs in one of the elements of the range.

Definition. A substitution θ is called **normal** if no element of its domain occurs in its range.

If θ is normal then $\theta\theta = \theta$. If θ is normal then $A\theta$ contains no element in the domain of θ.

Definition. Two substitutions θ and λ are said to **conflict** if their domains are not disjoint.

Lemma 2. If θ and λ are normal and non-conflicting then $\theta\lambda$ is normal if and only if no element in the domain of θ occurs in the range of λ .

Proof. Proof of $\Rightarrow$

Suppose x is in the domain of θ and occurs in b which occurs in the range of λ . But by the definition of composition, x is in the domain of $\theta\lambda$ and b occurs in the range of $\theta\lambda$. So $\theta\lambda$ is not normal.

Proof of $\Leftarrow$

Suppose $\theta\lambda$ is not normal. Then there is an x in the domain of $\theta\lambda$ that occurs in some c in the range of $\theta\lambda$. By the definition of composition, either x is in the domain of θ or in the domain of λ . Likewise either c is in the range of λ or there is some a in the range of θ such that $a\lambda = c$.

Suppose x is in the domain of λ . Since λ is normal, if c were in the range of λ then x can not occur in c. Likewise x can not occur in $a\lambda$. So x can not be in the domain of λ .

Therefore x must be in the domain of θ. Since θ is normal, x can not occur in any a in the range of θ. Thus x can occur in $a\lambda$ only if x occurs in the range of λ . So no matter where c comes from, x would occur in the range of λ .

QED.

Lemma 3. If θ, λ , and $\theta\lambda$ are normal and θ and λ do not conflict, then

$$\lambda\theta\lambda = \theta\lambda.$$

Proof. $\theta\lambda$ and $\lambda\theta\lambda$ only differ for elements in the domain of λ . Suppose v is some such variable. By the previous lemma, no element of the domain of θ occurs in the range of λ . Since $v\lambda$ is in the range of λ, $v\lambda\theta = v\lambda$. So $v\lambda\theta\lambda = v\lambda\lambda = v\lambda$ (since λ is normal $\lambda\lambda = \lambda$). Since v is in the domain of λ and since θ and λ do not conflict, v is not in the domain of θ. So $v\theta = v$ and $v\theta\lambda = v\lambda$. So $v\theta\lambda = v\lambda\theta\lambda$. So $\theta\lambda = \lambda\theta\lambda$.

Corollary. If θ, λ , and $\theta\lambda$ are normal and θ and λ do not conflict,

$$\forall\sigma(H\theta\lambda \Rightarrow H\lambda(\theta\lambda))\sigma$$

and

$$\theta\lambda = \theta\lambda(\theta\lambda).$$

Theorem 1. If ϕ is some substitution,

$$\begin{array}{rr} \forall\sigma(H \Rightarrow A)\theta\sigma & (3) \\ \wedge\,\forall\sigma(H \Rightarrow B\theta)\lambda\sigma & (4) \\ \wedge\,\forall\sigma(H\theta\lambda \Rightarrow H\lambda\phi)\sigma & (5) \\ \wedge\,\forall\sigma(B\theta\lambda\phi \Rightarrow B\theta\lambda)\sigma & (6) \\ \Rightarrow \forall\sigma(H \Rightarrow A \wedge B)\theta\lambda\sigma & \end{array}$$

Proof. We need to show that if γ is some substitution such that $(H \Rightarrow A \wedge B)\theta\lambda\gamma$ is ground then

$$H\theta\lambda\gamma \Rightarrow A\theta\lambda\gamma \wedge B\theta\lambda\gamma. \tag{7}$$

We will do so for an arbitrary γ satisfying that condition.

We begin by assuming

$$H\theta\lambda\gamma. \tag{8}$$

By the assumptions on γ, we know that (8) is ground. Combining (8) with (3), we then know

$$A\theta\lambda\gamma \tag{9}$$

is ground and true.

From (5) and the assumption of (8), we also know

$$\forall\sigma(H\lambda\phi\gamma)\sigma.$$

Combining this with (4), we know

$$\forall\sigma(B\theta\lambda\phi\gamma)\sigma.$$

Further combining this with (6) we get

$$B\theta\lambda\gamma \tag{10}$$

which is both ground and true.

Thus by assuming (8) we derived both (9) and (10), thereby proving (7).

QED.

By the previous corollary, it can be seen that the hypotheses about ϕ are satisfied by $\phi = \theta\lambda$ when θ, λ, and $\theta\lambda$ are all normal and θ and λ do not conflict. This is the usual case.

Theorem 2. If θ, λ , and $\theta\lambda$ are all normal and θ and λ do not conflict, then

$$\begin{gathered}\forall\sigma(H \Rightarrow A)\theta\sigma \\ \wedge \forall\sigma(H \Rightarrow B\theta)\lambda\sigma \\ \Rightarrow \forall\sigma(H \Rightarrow A \wedge B)\theta\lambda\sigma\end{gathered}$$

A.3 Generalized Substitutions

The previous theorem proves the soundness of APRVR if the substitutions returned are of a particular form. By employing generalized substitutions these restrictions may be removed.

Definition. θ is a **generalized substitution** if

(i) θ is an ordinary substitution, or

(ii) θ has the form of either

$$(\theta_1 \vee \theta_2)$$

or

$$(\theta_1 \wedge \theta_2)$$

where θ_1 and θ_2 are generalized substitutions.

Some examples are,

$$\theta_1, \quad \theta_1 \vee \theta_2, \quad ((\theta_1 \vee \theta_2) \wedge \theta_3),$$

where the θ_i are ordinary substitutions.

Definition. If θ is a generalized substitution, then we define θ' by

(i) $\theta' = \theta$ if θ is an ordinary substitution,

(ii) $(\theta_1 \vee \theta_2)' = (\theta'_1 \wedge \theta'_2)$,

(iii) $(\theta_1 \wedge \theta_2)' = (\theta'_1 \vee \theta'_2)$.

Definition. A generalized substitution is said to be a **pure disjunction (conjunction)** if it contains no $\wedge$ symbols ($\vee$ symbols).

Notice that this definition allows ordinary substitutions to be called pure disjunctions (and pure conjunctions).

Definition. If A is a formula and θ is a generalized substitution, then $A\theta$ is the formula obtained by applying θ from left to right, i.e.,

(i) $A\theta$ is the usual result if θ is an ordinary substitution,

(ii) $A(\theta_1 \vee \theta_2) = A\theta_1 \vee A\theta_2$,

(iii) $A(\theta_1 \wedge \theta_2) = A\theta_1 \wedge A\theta_2$.

An iterated generalized substitution such as $\theta\lambda$ can be converted into a generalized substitution by applying λ to θ. For example, if θ_i and λ_i are ordinary substitutions, then

$$\begin{aligned}(\theta_1 \wedge \theta_2)(\lambda_1 \vee \lambda_2) &= (\theta_1\lambda_1 \wedge \theta_2\lambda_1) \vee (\theta_1\lambda_2 \wedge \theta_2\lambda_2)\\ &= (\theta_1\lambda_1 \vee \theta_1\lambda_2) \wedge (\theta_1\lambda_1 \vee \theta_2\lambda_2) \wedge (\theta_2\lambda_1 \vee \theta_1\lambda_2) \wedge (\theta_2\lambda_1 \vee \theta_2\lambda_2).\end{aligned}$$

A.4 Properties of Generalized Substitutions

Lemma 4. If θ and λ are generalized substitutions, λ is a pure disjunction, and A and B are formulas then λ' is a pure conjunction and

4.1 $(\theta')' = \theta$

4.2 $\sim(A\theta) = \sim A\theta'$

4.3 $(A \vee B)\lambda = A\lambda \vee B\lambda$

4.4 $(A \wedge B)\lambda' = A\lambda' \wedge B\lambda'$

4.5 $(A \Rightarrow B)\lambda = (A\lambda' \Rightarrow B\lambda)$

Proof. 4.1 and 4.2 follow directly from the definition of θ' and the properties of $\sim$. 4.3 and 4.4 follow from the associativity of $\vee$ and of $\wedge$. Then 4.5 follows from 4.3, 4.2, and 4.1, as follows:

$$\begin{aligned}(A \Rightarrow B)\lambda &= (\sim A \vee B)\lambda \\ &= (\sim A\lambda \vee B\lambda) \\ &= (\sim(A\lambda') \vee B\lambda) \\ &= (A\lambda' \Rightarrow B\lambda).\end{aligned}$$

A.5 Generalized Substitutions in APRVR

Generalized substitutions can be generated when two subgoals are combined to prove a goal. This occurs during ANDSPLITs, forward chaining, and backward chaining. In order to prove these methods valid, it is necessary to specify what the first subgoal is, how its value is used in generating the second subgoal, and how the values of the two subgoals are combined into the (generalized) substitution that proves the goal.

In doing an ANDSPLIT on a goal of the form

$$H \Rightarrow (A \wedge B),$$

the first subgoal is

$$H \Rightarrow A.$$

If θ is the value returned, it is then used in generating the second subgoal

$$H \Rightarrow B\theta'.$$

If this subgoal returns a substitution of λ, then the generalized substitution

$$\theta\lambda \vee \lambda$$

is returned as the substitution proving the original goal.

Similarly, if a forward chaining is attempted on

$$(H \wedge (P \Rightarrow Q)) \Rightarrow C,$$

the first subgoal generated is

$$H \Rightarrow P.$$

If this goal is solved by the substitution θ, the second subgoal generated is

$$[H \wedge (P \Rightarrow Q) \wedge Q\theta] \Rightarrow C\theta.$$

If this is then proved with a substitution of λ, the substitution

$$\theta\lambda \vee \lambda$$

is returned as the value of the original goal.

Backward chaining is again similar. If the goal is

$$[H \wedge (P \Rightarrow Q)] \Rightarrow C,$$

the first subgoal is

$$Q \Rightarrow C.$$

If this returns the value θ, a second subgoal

$$H \Rightarrow P\theta'$$

is generated. If this is proved with a value λ , once again,

$$\theta\lambda \vee \lambda$$

is returned.

A.6 Soundness

We can now prove the soundness of APRVR when generalized substitutions are generated. As before, we will do so only for ANDSPLITs. Proofs for other rules are similar.

Note that all the substitutions returned as the value of a goal are either ordinary substitutions or, by the above rules, disjunctions of substitutions already returned from a subgoal; no conjunctions are ever returned. Therefore, the substitutions (whether ordinary or generalized) that APRVR returns are always pure disjunctions. Thus the values of the subgoals used as inductive hypotheses in the soundness theorems must be pure disjunctions.

If we are proving

$$H \Rightarrow A \wedge B,$$

and θ is returned for

$$H \Rightarrow A$$

and λ is returned for

$$H \Rightarrow B\theta'$$

then $\theta\lambda \vee \lambda$ is returned for

$$H \Rightarrow A \wedge B$$

Lemma 5. If θ and λ are pure disjunctions then

5.1 $(H \Rightarrow C)\theta\lambda \Leftrightarrow (H\theta'\lambda' \Rightarrow C\theta\lambda)$

5.2 $\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda' \Rightarrow C\theta\lambda)\sigma$

5.3 $\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda' \Rightarrow C\theta\lambda')\sigma$

5.4 $\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda \Rightarrow C\theta\lambda)\sigma$

Proof.

5.1

$$\begin{aligned}(H \Rightarrow C)\theta\lambda &\Leftrightarrow (H\theta' \Rightarrow C\theta)\lambda \\ &\Leftrightarrow (H\theta'\lambda' \Rightarrow C\theta\lambda)\end{aligned}$$

5.2

$$\begin{aligned}\forall\sigma(H \Rightarrow C)\theta\sigma &\Rightarrow \forall\sigma(H \Rightarrow C)\theta\lambda\sigma \\ &\Rightarrow \forall\sigma(H\theta'\lambda' \Rightarrow C\theta\lambda)\sigma\end{aligned}$$

5.3 Proof is by induction on the structure of λ.

Case 1. λ is ordinary.

$\lambda = \lambda$' so 5.2 applies.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$.

We use the induction hypotheses:

$$\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda_1' \Rightarrow C\theta\lambda_1')\sigma,$$
$$\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda_2' \Rightarrow C\theta\lambda_2')\sigma.$$

So we have

$$\begin{aligned}\forall\sigma(H \Rightarrow C)\theta\sigma &\Rightarrow (\forall\sigma(H\theta'\lambda_1' \Rightarrow C\theta\lambda_1')\sigma \wedge \forall\sigma(H\theta'\lambda_2' \Rightarrow C\theta\lambda_2')\sigma)\\ &\Rightarrow \forall\sigma[(H\theta'\lambda_1' \Rightarrow C\theta\lambda_1') \wedge (H\theta'\lambda_2' \Rightarrow C\theta\lambda_2')]\sigma\\ &\Rightarrow \forall\sigma[(H\theta'\lambda_1' \wedge H\theta'\lambda_2') \Rightarrow (C\theta\lambda_1' \wedge C\theta\lambda_2')]\sigma\\ &\Rightarrow \forall\sigma[H\theta'(\lambda_1' \wedge \lambda_2') \Rightarrow C\theta(\lambda_1' \wedge \lambda_2')]\sigma\\ &\Rightarrow \forall\sigma(H\theta'\lambda' \Rightarrow C\theta\lambda')\sigma\end{aligned}$$

5.4 Proof is by induction on the structure of λ .

Case 1. λ is ordinary.

$\lambda = \lambda'$ so 5.2 applies.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$.

We use the induction hypotheses:

$$\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda_1 \Rightarrow C\theta\lambda_1)\sigma,$$
$$\forall\sigma(H \Rightarrow C)\theta\sigma \Rightarrow \forall\sigma(H\theta'\lambda_2 \Rightarrow C\theta\lambda_2)\sigma.$$

So we have

$$\begin{aligned}\forall\sigma(H \Rightarrow C)\theta\sigma &\Rightarrow (\forall\sigma(H\theta'\lambda_1 \Rightarrow C\theta\lambda_1)\sigma \wedge \forall\sigma(H\theta'\lambda_2 \Rightarrow C\theta\lambda_2)\sigma)\\ &\Rightarrow \forall\sigma[(H\theta'\lambda_1 \Rightarrow C\theta\lambda_1) \wedge (H\theta'\lambda_2 \Rightarrow C\theta\lambda_2)]\sigma\\ &\Rightarrow \forall\sigma[(H\theta'\lambda_1 \vee H\theta'\lambda_2) \Rightarrow (C\theta\lambda_1 \vee C\theta\lambda_2)]\sigma\\ &\Rightarrow \forall\sigma[H\theta'(\lambda_1 \vee \lambda_2) \Rightarrow C\theta(\lambda_1 \vee \lambda_2)]\sigma\\ &\Rightarrow \forall\sigma(H\theta'\lambda \Rightarrow C\theta\lambda)\sigma\end{aligned}$$

Lemma 6. If θ and λ are pure disjunctive generalized substitutions then

$$\forall\sigma[(A\theta\lambda' \wedge B\theta'\lambda) \Rightarrow (A \wedge B)\theta\lambda]\sigma$$

Proof. Proof is by induction on the structure of λ .

Case 1. λ is ordinary.

Since $\lambda = \lambda'$ we need to establish

$$\forall\sigma[(A\theta\lambda \wedge B\theta'\lambda) \Rightarrow (A \wedge B)\theta\lambda]\sigma. \tag{11}$$

This is shown by induction on the structure of θ.

Case 1.1. θ is ordinary.

$$(A\theta\lambda \wedge B\theta'\lambda) = (A\theta\lambda \wedge B\theta\lambda) = (A \wedge B)\theta\lambda.$$

Case 1.2. $\theta = \theta_1 \vee \theta_2$. We will use the induction hypotheses:

$$\forall\sigma[(A\theta_1\lambda \wedge B\theta_1'\lambda) \Rightarrow (A \wedge B)\theta_1\lambda]\sigma,$$

$$\forall\sigma[(A\theta_2\lambda \wedge B\theta_2'\lambda) \Rightarrow (A \wedge B)\theta_2\lambda]\sigma.$$

We need to show

$$((A\theta\lambda \wedge B\theta'\lambda) \Rightarrow (A \wedge B)\theta\lambda)\sigma \tag{12}$$

for all σ's such that (12) is ground[4]. We will show it for an arbitrary σ satisfying that condition.

$$\begin{aligned}
A\theta\lambda\sigma \wedge B\theta'\lambda\sigma &\Rightarrow A(\theta_1 \vee \theta_2)\lambda\sigma \wedge B(\theta_1' \wedge \theta_2')\lambda\sigma \\
&\Rightarrow (A\theta_1\lambda\sigma \vee A\theta_2\lambda\sigma) \wedge B\theta_1'\lambda\sigma \wedge B\theta_2'\lambda\sigma \\
&\Rightarrow (A\theta_1\lambda\sigma \wedge B\theta_1'\lambda\sigma \wedge B\theta_2'\lambda\sigma) \\
&\qquad \vee (A\theta_2\lambda\sigma \wedge B\theta_1'\lambda\sigma \wedge B\theta_2'\lambda\sigma) \\
&\Rightarrow (A\theta_1\lambda\sigma \wedge B\theta_1'\lambda\sigma) \vee (A\theta_2\lambda\sigma \wedge B\theta_2'\lambda\sigma)
\end{aligned} \tag{13}$$

By induction hypothesis ((13) is ground since (12) is),

$$\begin{aligned}
&\Rightarrow (A \wedge B)\theta_1\lambda\sigma \vee (A \wedge B)\theta_2\lambda\sigma \\
&\Rightarrow (A \wedge B)(\theta_1 \vee \theta_2)\lambda\sigma \\
&\Rightarrow (A \wedge B)\theta\lambda\sigma
\end{aligned}$$

So (12) is shown for an arbitrary σ. So (11) is established.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$. We use the induction hypotheses:

$$\forall\sigma[(A\theta\lambda_1' \wedge B\theta'\lambda_1) \Rightarrow (A \wedge B)\theta\lambda_1]\sigma,$$

$$\forall\sigma[(A\theta\lambda_2' \wedge B\theta'\lambda_2) \Rightarrow (A \wedge B)\theta\lambda_2]\sigma.$$

We need to show

$$[(A\theta\lambda' \wedge B\theta'\lambda) \Rightarrow (A \wedge B)\theta\lambda]\sigma \tag{14}$$

[4]The substitution σ can be presumed to be an ordinary substitution.

for all σ for which (14) is ground. We will show it for an arbitrary σ satisfying that condition.

$$\begin{aligned}A\theta\lambda'\sigma \land B\theta'\lambda\sigma &\Rightarrow A\theta(\lambda_1' \land \lambda_2')\sigma \land B\theta'(\lambda_1 \lor \lambda_2)\sigma\\ &\Rightarrow A\theta\lambda_1'\sigma \land A\theta\lambda_2'\sigma \land (B\theta'\lambda_1\sigma \lor B\theta'\lambda_2\sigma)\\ &\Rightarrow (A\theta\lambda_1'\sigma \land A\theta\lambda_2'\sigma \land B\theta'\lambda_1\sigma) \lor (A\theta\lambda_1'\sigma \land A\theta\lambda_2'\sigma \land B\theta'\lambda_2\sigma)\\ &\Rightarrow (A\theta\lambda_1'\sigma \land B\theta'\lambda_1\sigma) \lor (A\theta\lambda_2'\sigma \land B\theta'\lambda_2\sigma) \qquad (15)\end{aligned}$$

By induction hypothesis ((15) is ground since (14) is),

$$\begin{aligned}&\Rightarrow (A \land B)\theta\lambda_1\sigma \lor (A \land B)\theta\lambda_2\sigma\\ &\Rightarrow (A \land B)\theta(\lambda_1 \lor \lambda_2)\sigma\\ &\Rightarrow (A \land B)\theta\lambda\sigma\end{aligned}$$

QED.

A.7 Soundness Theorem for ANDSPLIT

Theorem 3. If θ and λ are pure disjunctive substitutions then

$$\begin{aligned}&\forall\sigma(H \Rightarrow A)\theta\sigma \qquad (16)\\ \land\, &\forall\sigma(H \Rightarrow B\theta')\lambda\sigma \qquad (17)\\ \Rightarrow &\forall\sigma(H \Rightarrow A \land B)(\theta\lambda \lor \lambda)\sigma\end{aligned}$$

Proof. We need to show

$$(H \Rightarrow A \land B)(\theta\lambda \lor \lambda)\sigma \qquad (18)$$

for every substitution σ such that (18) is ground.

Rewriting this we get

$$(H\theta'\lambda'\sigma \land H\lambda'\sigma) \Rightarrow ((A \land B)\theta\lambda\sigma \lor (A \land B)\lambda\sigma)$$

Now

$$\begin{aligned}H\theta'\lambda'\sigma \land H\lambda'\sigma &\Rightarrow A\theta\lambda'\sigma \land H\lambda'\sigma && \text{by (16) and Lemma 5.3}\\ &\Rightarrow A\theta\lambda'\sigma \land B\theta'\lambda\sigma && \text{by (17)}\\ &\Rightarrow (A \land B)\theta\lambda\sigma && \text{by Lemma 6}\\ &\Rightarrow (A \land B)\theta\lambda\sigma \lor (A \land B)\lambda\sigma\end{aligned}$$

QED.

The only non-trivial rules left are backward chaining and forward chaining. The generalized soundness theorems for these are

Backward Chaining:

$$\begin{aligned}&\forall\sigma(B \Rightarrow C)\theta\sigma\\ &\wedge \forall\sigma(H \Rightarrow A\theta')\lambda\sigma\\ &\Rightarrow \forall\sigma((H \wedge (A \Rightarrow B)) \Rightarrow C)(\theta\lambda \vee \lambda)\sigma\end{aligned}$$

Forward Chaining:

$$\begin{aligned}&\forall\sigma(H \Rightarrow A)\theta\sigma\\ &\wedge \forall\sigma((H \wedge (A \Rightarrow B) \wedge B\theta) \Rightarrow C\theta)\lambda\sigma\\ &\Rightarrow \forall\sigma((H \wedge (A \Rightarrow B)) \Rightarrow C)(\theta\lambda \vee \lambda)\sigma\end{aligned}$$

These require the lemmas

$$\forall\sigma(H \Rightarrow C)\theta'\lambda'\sigma \Rightarrow \forall\sigma(H\theta'\lambda \Rightarrow C\theta'\lambda)\sigma,$$
$$\forall\sigma(H \Rightarrow C)\theta'\lambda'\sigma \Rightarrow \forall\sigma(H\theta\lambda' \Rightarrow C\theta\lambda')\sigma.$$

A.8 Examples

Example 1. $P(x) \Rightarrow (P(a) \wedge P(b))$

This requires the generalized substitution $(\{a/x\} \vee \{b/x\})$. Without further knowledge of where this goal is used, APRVR can not make the assumption that it had a hypothesis that stated $P(x)$ was true for all x. When APRVR can determine that this goal was generated from such a situation, APRVR will simply return the empty substitution.

Example 2.

$$\begin{aligned}&P(a,b,z) \wedge Q(a,b,d,y)\\ &\wedge R(a,b,f) \wedge R(e,c,d)\\ &\Rightarrow (P(x,y,z) \wedge Q(a,y,z,c)) \wedge R(x,y,z)\end{aligned}$$

This is not a theorem. The proof of

$$P(x,y,z) \wedge Q(a,y,z,c)$$

requires the composition of the substitutions $\{a/x, b/y\}$ and $\{d/z, c/y\}$. With generalized substitutions we then have to prove

$$R(x,y,z)(\{a/x,b/y,d/z\} \wedge (d/z,c/y\})$$

which translates to

$$R(a,b,d) \wedge R(x,c,d).$$

This can not be proved from the hypotheses.

Example 3.

$$\begin{aligned} &P(a,b,z) \wedge Q(a,b,d,y) \\ &\wedge R(a,b,d) \wedge R(e,c,d) \\ &\Rightarrow (P(x,y,z) \wedge Q(a,y,z,c)) \wedge R(x,y,z) \end{aligned}$$

This is similar to example 2 but is a theorem. With generalized substitutions it is proved but IMPLY (which does not have generalized substitutions) could not prove it.

Example 4. $(P(f(x)) \wedge Q(g(y))) \Rightarrow (P(y) \wedge Q(x))$

This theorem is proved using the generalized substitution $(\{f(g(y))/y, g(y)/x\} \vee \{g(y)/x\})$.

Example 5.

$$Q(a,f(b)) \Rightarrow [(P(f(x)) \Rightarrow [P(f(a)) \wedge P(y)]) \wedge Q(x,y)]$$

The first step in proving this is to prove

$$Q(a,f(b)) \wedge P(f(x)) \Rightarrow P(f(a)) \wedge P(y).$$

This requires the combination of the substitutions $\{a/x\}$ and $\{f(x)/y\}$. With generalized substitutions we then need to prove

$$(Q(a,f(b)) \Rightarrow Q(x,y))(\{a/x, f(x)/y\} \wedge \{f(x)/y\})$$

which becomes

$$Q(a,f(b)) \Rightarrow (Q(a,f(x)) \wedge Q(x,f(x))).$$

This is clearly not provable.

APPENDIX B.

EXAMPLES OF PROOFS BY APRVR

B.1 Proof of *2.13

Note: The ORR, NOTT, and IMPP are the disguised version of APRVR's disjunction, negation, and implication operators. The formulas in the proof presentation below the list of lemmas are "pretty printed" in infix notation. Skolem functions and variables include in them the name of the proposition from which they came. Variables are distinguished by a dollar sign ($). So `P*1:01$` is a variable that derived from proposition *1.01 in *Principia Mathematica*.

This is the proof output from APRVR:

```
Proving
                        (ORR (P*2:13)
                             (NOTT (NOTT (NOTT (P*2:13)))))

Using as lemmas:
                        (DF (IMPP P*1:01$ Q*1:01$)
                            (ORR (NOTT P*1:01$) Q*1:01$))

                        (IMPP (ORR P*1:2$ P*1:2$) P*1:2$)

                        (IMPP Q*1:3$ (ORR P*1:3$ Q*1:3$))

                        (IMPP (ORR P*1:4$ Q*1:4$)
                              (ORR Q*1:4$ P*1:4$))

                        (IMPP (ORR P*1:5$ (ORR Q*1:5$ R*1:5$))
                              (ORR Q*1:5$ (ORR P*1:5$ R*1:5$)))

                        (IMPP (IMPP Q*1:6$ R*1:6$)
                              (IMPP (ORR P*1:6$ Q*1:6$)
                                    (ORR P*1:6$ R*1:6$)))

                        (IMPP (IMPP P*2:01$ (NOTT P*2:01$))
                              (NOTT P*2:01$))

                        (IMPP Q*2:02$ (IMPP P*2:02$ Q*2:02$))
```

```
(IMPP (IMPP P*2:03$ (NOTT Q*2:03$))
      (IMPP Q*2:03$ (NOTT P*2:03$)))

(IMPP (IMPP P*2:04$
            (IMPP Q*2:04$ R*2:04$))
      (IMPP Q*2:04$
            (IMPP P*2:04$ R*2:04$)))

(IMPP (IMPP Q*2:05$ R*2:05$)
      (IMPP (IMPP P*2:05$ Q*2:05$)
            (IMPP P*2:05$ R*2:05$)))

(IMPP (IMPP P*2:06$ Q*2:06$)
      (IMPP (IMPP Q*2:06$ R*2:06$)
            (IMPP P*2:06$ R*2:06$)))

(IMPP P*2:07$ (ORR P*2:07$ P*2:07$))

(IMPP P*2:08$ P*2:08$)

(ORR (NOTT P*2:1$) P*2:1$)

(ORR P*2:11$ (NOTT P*2:11$))

(IMPP P*2:12$ (NOTT (NOTT P*2:12$)))
```

Here is the "trick". Backward chaining would not find a formula of this type. Once the formulas that `P->*$2` and `P->*$3` are proved, the theorem is proved. With the normal (non-PM) logical operators, this would not be needed.

```
(Goal 60)  The formula
    (   P->*$2
     -> (   P->*$3
         -> ((P*2:13) ORR ---(P*2:13))))
 was found to be true:
     This was proved by matching (and possibly chaining)
     with the hypotheses:
         (   (Q*1:6$ -> R*1:6$)
          -> ((P*1:6$ ORR Q*1:6$) -> (P*1:6$ ORR R*1:6$))
         )
     Substitutions:
     P->*$2 by (Q*1:6$ -> ---P*2:13)
     P->*$3 by (P*2:13 ORR Q*1:6$)
     P*1:6$ by P*2:13
    R*1:6$ by ---P*2:13
(Goal 60) Now it remains to be proven that both the
hypothesis and the hypothesis of the conclusion are true.
 Then the conclusion of the conclusion (what is being
proven) will be known to be true.
So we need to prove:
     (    (Q*1:6$ -> ---(P*2:13))
      AND ((P*2:13) ORR Q*1:6$))
      (Goal 61)  This has a simple proof:
           The AND in the conclusion is split into two
           parts.
           The first conjunct
               ((P*2:13) ORR Q*1:6$)
           is proved by:
                This was proved by matching (and possibly
                chaining) with the hypotheses:
                    (P*2:11$ ORR -P*2:11$)
                Substitutions:
                P*2:11$ by P*2:13
                Q*1:6$ by -P*2:13
           The second conjunct
               (Q*1:6$ -> ---(P*2:13))
           which becomes (after applying the substitution)
               (   -(P*2:13)
                -> ---(P*2:13))
           is proved by:
                This was proved by matching (and possibly
                chaining) with the hypotheses:
                    (P*2:11$ ORR -P*2:11$)
                Substitutions:
                P*2:11$ by --P*2:13
           The two substitutions used in proving the two
           conjuncts combine to form the substitution
           Generalized substitution:
               P*2:11$ by P*2:13
               Q*1:6$ by -P*2:13
           OR
               P*2:11$ by --P*2:13
```

The generalized substitution is just an artifact of QKIMPLY not using variables standardized apart for two instances of a lemma.

```
      (Goal 61)  The extraneous substitutions due to the
      lemmas were removed, leaving
          Substitutions:
          Q*1:6$ by -P*2:13
```

```
(Goal 60)  So we have established
    ((P*2:13) ORR ---(P*2:13))
```

B.2 Proof of *2.17

The following is a transcript of the output of APRVR during the proof of *2.17.

```
Proving *2:17

Using control system AGENDA-SYSTEM
```

Skolemization has already occurred.

```
Proving:
((-Q*2:17 -> -P*2:17) -> (P*2:17 -> Q*2:17))
Using lemmas:

(DF (P*1:01 -> Q*1:01) (-P*1:01 ORR Q*1:01))

((P*1:2 ORR P*1:2) -> P*1:2)

(Q*1:3 -> (P*1:3 ORR Q*1:3))

((P*1:4 ORR Q*1:4) -> (Q*1:4 ORR P*1:4))

(    (P*1:5 ORR (Q*1:5 ORR R*1:5))
 -> (Q*1:5 ORR (P*1:5 ORR R*1:5)))

(    (Q*1:6 -> R*1:6)
 -> ((P*1:6 ORR Q*1:6) -> (P*1:6 ORR R*1:6)))

((P*2:01 -> -P*2:01) -> -P*2:01)

(Q*2:02 -> (P*2:02 -> Q*2:02))

((P*2:03 -> -Q*2:03) -> (Q*2:03 -> -P*2:03))

(    (P*2:04 -> (Q*2:04 -> R*2:04))
 -> (Q*2:04 -> (P*2:04 -> R*2:04)))

(    (Q*2:05 -> R*2:05)
 -> ((P*2:05 -> Q*2:05) -> (P*2:05 -> R*2:05)))

(    (P*2:06 -> Q*2:06)
 -> ((Q*2:06 -> R*2:06) -> (P*2:06 -> R*2:06)))

(P*2:07 -> (P*2:07 ORR P*2:07))

(P*2:08 -> P*2:08)
```

```
(-P*2:1 ORR P*2:1)

(P*2:11 ORR -P*2:11)

(P*2:12 -> --P*2:12)

(P*2:13 ORR ---P*2:13)

(--P*2:14 -> P*2:14)

((-P*2:15 -> Q*2:15) -> (-Q*2:15 -> P*2:15))

((P*2:16 -> Q*2:16) -> (-Q*2:16 -> -P*2:16))
```

This is the main goal:

```
  Making 0(GOAL 0 )
                (    (-Q*2:17 -> -P*2:17)
                 -> (P*2:17 -> Q*2:17))

Working 0 (GOAL 452)
             (    (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
  Making 1(THMOPS 10 )
                (    (-Q*2:17 -> -P*2:17)
                 -> (P*2:17 -> Q*2:17))

Working 1 (THMOPS 462)
             (    (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
  Making 2(HAND 20 )
                (    (-Q*2:17 -> -P*2:17)
                 -> (P*2:17 -> Q*2:17))
 (Nothing else to be done on goal 1)

Working 2 (HAND 472)
             (    (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))

Working 2 (HAND 562)
             (    (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
```

Backwards chaining:

```
Trying to match goal with conclusion of lemma:
```

```
                      (   (P*2:06$1 -> Q*2:06$1)
                       -> (   (Q*2:06$1 -> R*2:06$1)
                           -> (P*2:06$1 -> R*2:06$1)))
  Making 3(GOAL 123 )
                   (   (   (Q*2:06$1 -> R*2:06$1)
                        -> (P*2:06$1 -> R*2:06$1))
                    -> (   (-Q*2:17 -> -P*2:17)
                        -> (P*2:17 -> Q*2:17)))
```

A list of the first few goals on the agenda and their priorities:

```
--------- 2(562) 3(975) 0(1130)
Working 2 (HAND 562)
             (   (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
Trying to match goal with conclusion of lemma:
                      (   (Q*2:05$1 -> R*2:05$1)
                       -> (   (P*2:05$1 -> Q*2:05$1)
                           -> (P*2:05$1 -> R*2:05$1)))
  Making 4(GOAL 123 )
                   (   (   (P*2:05$1 -> Q*2:05$1)
                        -> (P*2:05$1 -> R*2:05$1))
                    -> (   (-Q*2:17 -> -P*2:17)
                        -> (P*2:17 -> Q*2:17)))

--------- 2(562) 4(975) 3(975) 0(1130)
Working 2 (HAND 562)
             (   (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
Trying to match goal with conclusion of lemma:
                      (   (Q*1:6$1 -> R*1:6$1)
                       -> (   (P*1:6$1 ORR Q*1:6$1)
                           -> (P*1:6$1 ORR R*1:6$1)))
  Making 5(GOAL 123 )
                   (   (   (P*1:6$1 ORR Q*1:6$1)
                        -> (P*1:6$1 ORR R*1:6$1))
                    -> (   (-Q*2:17 -> -P*2:17)
                        -> (P*2:17 -> Q*2:17)))

--------- 2(562) 4(975) 3(975) 0(1130) 5(1145)
Working 2 (HAND 562)
             (   (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))

Working 2 (HAND 670)
             (   (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
Trying to match conclusion of goal with conclusion of
 lemma:
                        (    (P*2:16$1 -> Q*2:16$1)
                         -> (-Q*2:16$1 -> -P*2:16$1))
  Making 6(GOAL 708 )
                     (    (-Q*2:16$1 -> -P*2:16$1)
                      -> (P*2:17 -> Q*2:17))
```

... Some output not involved with the proof removed here.

```
--------- 21(732) 2(800) 28(852) 25(864) 23(864) 4(975)
          3(975)
Working 21 (HAND 732)
              (    (--P*2:17 -> --Q*2:17)
               -> (P*2:17 -> Q*2:17))

Working 21 (HAND 864)
              (    (--P*2:17 -> --Q*2:17)
               -> (P*2:17 -> Q*2:17))
Trying to match conclusion of goal with conclusion of
 lemma:
                        (    (P*2:16$3 -> Q*2:16$3)
                         -> (-Q*2:16$3 -> -P*2:16$3))
```

APRVR reuses goals.

```
Goal being created found to exist already as goal 6 up to
 renaming
     Have to generate a renaming goal
  Making 37(RENAME 908 )
                     T

--------- 2(800) 28(852) 25(864) 23(864) 21(864) 4(975)
          3(975)
Working 2 (HAND 800)
              (    (-Q*2:17 -> -P*2:17)
               -> (P*2:17 -> Q*2:17))
Trying to forward chain with lemmas:(    (    -P*2:15$2
                                          -> Q*2:15$2)
                                     -> (    -Q*2:15$2
                                          -> P*2:15$2))
  Making 38(GOAL 177 )
                    (    (-Q*2:17 -> -P*2:17)
                     -> (-P*2:15$2 -> Q*2:15$2))
```

APRVR handles multiple proofs.

```
Subgoal just created has 7 simple proofs
Goal 38 is proved with the value:
                 Substitutions:
                 P*2:15$2 by Q*2:17
                 Q*2:15$2 by -P*2:17
     Waking parents: 2
Adding a new value to goal 38
     New value = Substitutions:
                 Q*2:15$2 by (-Q*2:17 -> -P*2:17)
     Waking parents: 2
Value just passed up is same as before
Value just passed up is same as before
Value just passed up is same as before
Adding a new value to goal 38
     New value = Substitutions:
                 P*2:15$2 by -P*2:17
                 Q*2:15$2 by Q*2:17
     Waking parents: 2
Adding a new value to goal 38
     New value = Substitutions:
                 P*2:15$2 by -P*2:17
                 Q*2:15$2 by --Q*2:17
     Waking parents: 2

Working 2 (HAND -800)
            (   (-Q*2:17 -> -P*2:17)
             -> (P*2:17 -> Q*2:17))
Matched hypothesis of goal 2 with hypothesis of lemma:
                     (   (-P*2:15$2 -> Q*2:15$2)
                      -> (-Q*2:15$2 -> P*2:15$2))
          Try to prove conclusion of lemma implies
           conclusion of goal.
Goal being created found to exist already as goal 17
Applying that substitution would result in this goal being
 repeated
Matched hypothesis of goal 2 with hypothesis of lemma:
                     (   (-P*2:15$2 -> Q*2:15$2)
                      -> (-Q*2:15$2 -> P*2:15$2))
          Try to prove conclusion of lemma implies
           conclusion of goal.
  Making 39(GOAL 50 )
                 (   (   -(-Q*2:17 -> -P*2:17)
                      -> P*2:15$2)
                  -> (P*2:17 -> Q*2:17))
```

```
Matched hypothesis of goal 2 with hypothesis of lemma:
                    (    (-P*2:15$2 -> Q*2:15$2)
                     -> (-Q*2:15$2 -> P*2:15$2))
          Try to prove conclusion of lemma implies
           conclusion of goal.
  Making 40(GOAL 50 )
                 (    (--P*2:17 -> Q*2:17)
                  -> (P*2:17 -> Q*2:17))

Working 40 (GOAL 502)
            (    (--P*2:17 -> Q*2:17)
             -> (P*2:17 -> Q*2:17))
  Making 41(THMOPS 60 )
                 (    (--P*2:17 -> Q*2:17)
                  -> (P*2:17 -> Q*2:17))

Working 41 (THMOPS 512)
            (    (--P*2:17 -> Q*2:17)
             -> (P*2:17 -> Q*2:17))
  Making 42(HAND 70 )
                 (    (--P*2:17 -> Q*2:17)
                  -> (P*2:17 -> Q*2:17))
 (Nothing else to be done on goal 41)

Working 42 (HAND 522)
            (    (--P*2:17 -> Q*2:17)
             -> (P*2:17 -> Q*2:17))

Working 42 (HAND 612)
            (    (--P*2:17 -> Q*2:17)
             -> (P*2:17 -> Q*2:17))
Trying to match goal with conclusion of lemma:
                   (    (P*2:06$5 -> Q*2:06$5)
                    -> (    (Q*2:06$5 -> R*2:06$5)
                        -> (P*2:06$5 -> R*2:06$5)))
  Making 43(GOAL 173 )
                 (   (    (Q*2:06$5 -> R*2:06$5)
                     -> (P*2:06$5 -> R*2:06$5))
                  -> (    (--P*2:17 -> Q*2:17)
                     -> (P*2:17 -> Q*2:17)))
Subgoal just created has 3 simple proofs
Goal 43 is proved with the value:
                 Substitutions:
                 Q*2:06$5 by --P*2:17
                 R*2:06$5 by Q*2:17
                 P*2:06$5 by P*2:17
```

```
     Waking parents: 42
Value just passed up is same as before
Value just passed up is same as before

Working 42 (HAND -612)
             (    (--P*2:17 -> Q*2:17)
              -> (P*2:17 -> Q*2:17))
Matched goal 42 with conclusion of lemma:
                      (    (P*2:06$5 -> Q*2:06$5)
                       -> (    (Q*2:06$5 -> R*2:06$5)
                           -> (P*2:06$5 -> R*2:06$5)))
          Now try to prove its hypothesis
  Making 44(GOAL 170 )
                 (P*2:17 -> --P*2:17)
Subgoal just created has 2 simple proofs
Goal 44 is proved with the value:
                 No substitutions necessary
     Waking parents: 42
Value just passed up is same as before

Working 42 (HAND -612)
             (    (--P*2:17 -> Q*2:17)
              -> (P*2:17 -> Q*2:17))
Goal proved by backchaining via goals 43 and 44
Goal 42 is proved with the value:
                 No substitutions necessary
     Waking parents: 41

Working 41 (THMOPS -INFINITY)
             (    (--P*2:17 -> Q*2:17)
              -> (P*2:17 -> Q*2:17))
Goal 41 is proved with the value:
                 No substitutions necessary
     Waking parents: 40

Working 40 (GOAL -1180)
             (    (--P*2:17 -> Q*2:17)
              -> (P*2:17 -> Q*2:17))
Goal 40 is proved with the value:
                 No substitutions necessary
     Waking parents: 2

Working 2 (HAND -800)
             (    (-Q*2:17 -> -P*2:17)
              -> (P*2:17 -> Q*2:17))
Goal proved by forward chaining via goals 38 and 40
Goal 2 is proved with the value:
                  No substitutions necessary
     Waking parents: 1

Working 1 (THMOPS -INFINITY)
            (   (-Q*2:17 -> -P*2:17)
             -> (P*2:17 -> Q*2:17))
Goal 1 is proved with the value:
                  No substitutions necessary
     Waking parents: 0

Working 0 (GOAL -1130)
            (   (-Q*2:17 -> -P*2:17)
             -> (P*2:17 -> Q*2:17))
Goal 0 is proved with the value:
                  No substitutions necessary
     Waking parents:
*** Proved:
No substitutions necessary
```

Some statistics on goal types.

```
Goal type     Total      # Proved      # w/Subgoals
GOAL           32           7               6
RENAME          1           0               0
THMOPS          6           2               6
HAND            6           2               5

Totals:        45          11              17
```

APRVR chose a goal from the agenda 53 times. The new goal was neither a new son nor a woken parent 22 of those times.

```
There were 53 changes of goals with 22 changes unexpected.
```

The proof as printed by APRVR.

```
Proving
                   (IMPP (IMPP (NOTT (Q*2:17))
                               (NOTT (P*2:17)))
                         (IMPP (P*2:17) (Q*2:17)))

Using as lemmas:
```

Same lemmas as in the trace output above.

```
(Goal 2)  Going to forward-chain using hypothesis
    (    (-P*2:15$2 -> Q*2:15$2)
     -> (-Q*2:15$2 -> P*2:15$2))
To set up the chaining we have to prove
    (    (-(Q*2:17) -> -(P*2:17))
     -> (-P*2:15$2 -> Q*2:15$2))
Which is proved by:
     (Goal 38)  This has a simple proof:
          The hypothesis and conclusion of the IMPP
          matched.
          Substitutions:
          P*2:15$2 by Q*2:17
          Q*2:15$2 by -P*2:17
     (Goal 38)
(Goal 2)  Now we continue with the proof but are now
proving
    (    (--(P*2:17) -> (Q*2:17))
     -> ((P*2:17) -> (Q*2:17)))
(Goal 42)  Going to back-chain using hypothesis
    (    (P*2:06$5 -> Q*2:06$5)
     -> ((Q*2:06$5 -> R*2:06$5)
         ->
         (P*2:06$5 -> R*2:06$5)))
To set up the chaining we have to prove
    (    ((Q*2:06$5 -> R*2:06$5)
         ->
         (P*2:06$5 -> R*2:06$5))
     -> (    (--(P*2:17) -> (Q*2:17))
         -> ((P*2:17) -> (Q*2:17))))
Which is proved by:
     (Goal 43)  This has a simple proof:
          The hypothesis and conclusion of the IMPP
          matched.
          Substitutions:
          Q*2:06$5 by --P*2:17
          R*2:06$5 by Q*2:17
          P*2:06$5 by P*2:17
     (Goal 43)
(Goal 42)  Now we continue with the proof but are now
proving
    ((P*2:17) -> --(P*2:17))
(Goal 44)  This has a simple proof:
     This was proved by matching (and possibly chaining)
     with the hypotheses:
         (P*2:11$ ORR -P*2:11$)
    Substitutions:
    P*2:11$ by -P*2:17
```

QKIMPLY used a lemma to prove the goal. It returned a substitution including the substitution for the variable in that lemma. The variable did not occur in the goal so could be removed from the substitution.

```
(Goal 44)  The extraneous substitutions due to the lemmas
were removed, leaving
    No substitutions necessary

(Goal 42)  is proved by chaining.
(Goal 2)  is proved by chaining.
```

B.3 Proof of AM8

Here is the proof of AM8 as recorded by APRVR. Unfortunately, this does not include any trace of how the cases were generated (since that is not part of the actual proof). What APRVR does is to backchain on the second lemma, resulting in two conjuncts in the conclusion. The first of these matches with the first hypothesis and the second backchains on the third hypothesis. A split is done on the OR in the first lemma and two different substitutions are found for one of the disjuncts so two different sets of cases are created under goal 0.

```
Proving
                        (->
                         (AND
                          (P (F (G W$)) (F W$))
                          (-> (P (F X$1) (F W$))
                              (P (G W$) X$1))
                          (-> (NOT (P (L) T$))
                              (P (F (L)) (F T$)))
                          (->
                           (NOT (P X$ (L)))
                           (AND (P (T X$) X$)
                                (NOT (P (F X$) (F (T X$)))))))
                         (P (F U$) (F (T#1 U$))))

Using as lemmas:
                        (OR (NOT (P X$2 Y$))
                            (NOT (P Y$ X$2))
                            (AND (P X$2 Y$) (P Y$ X$2)))

                        (-> (AND (P X$3 Y$1) (P Y$1 Z$))
                            (P X$3 Z$))

                        (OR (P X$4 Y$2) (P Y$2 X$4))

                        (P X$5 X$5)

                        (-> (AND (P X$6 Y$3) (P Y$3 X$6))
                            (P (F X$6) (F Y$3)))
```

Unfortunately, the proof mechanism does not record a description of why the cases were generated.

```
 (Goal 170)  causes a case-split on the instantiated
 hypothesis
```

```
(   -(P (G (T#1 (L))) (L))
 OR -(P (L) (G (T#1 (L))))
 OR ((P (G (T#1 (L))) (L)) AND (P (L) (G (T#1 (L)))))
)
 (Goal 224)  The AND in the conclusion
     (     (   -(P (G (T#1 (L))) (L))
            -> (P (F U$) (F (T#1 U$))))
      AND (   -(P (L) (G (T#1 (L))))
            -> (P (F U$) (F (T#1 U$))))
      AND (    (    (P (G (T#1 (L))) (L))
                AND (P (L) (G (T#1 (L)))))
            -> (P (F U$) (F (T#1 U$)))))
 is split into two parts.
 The conjunct
     (   -(P (L) (G (T#1 (L))))
      -> (P (F U$) (F (T#1 U$))))
 is proved first:
      (Goal 311)  This has a simple proof:
```

A "simple proof" means that QKIMPLY was able to prove it.

```
              This was proved by matching (and possibly
              chaining) with the hypotheses:
                  (P (F (L)) (F (T#1 (L))))
              Substitutions:
              U$ by L
```

That hypothesis was the result of forward chaining using

`-(P (L) T$) -> (P (F (L)) (F T$))`

using `(T#1 (L))` for `T$`

and `(P (F (G W$)) (F W$))`

using `(T#1 (L))` for `W$`

and `((P X$3 Y$1) AND (P Y$1 Z$)) -> (P X$3 Z$)`

after promoting

`-(P (L) (G (T#1 (L))))`

```
      (Goal 311)
 (Goal 224)  The remaining conjunct is
     (     (   -(P (G (T#1 (L))) (L))
            -> (P (F U$) (F (T#1 U$))))
      AND (    (    (P (G (T#1 (L))) (L))
                AND (P (L) (G (T#1 (L)))))
```

```
                    -> (P (F U$) (F (T#1 U$)))))
        which becomes (after applying the substitution)
            (    (   -(P (G (T#1 (L))) (L))
                    -> (P (F (L)) (F (T#1 (L)))))
             AND (   (     (P (G (T#1 (L))) (L))
                         AND (P (L) (G (T#1 (L)))))
                    -> (P (F (L)) (F (T#1 (L))))))
             (Goal 312)  The AND in the conclusion
                 (    (   -(P (G (T#1 (L))) (L))
                           -> (P (F (L)) (F (T#1 (L)))))
                  AND (   (     (P (G (T#1 (L))) (L))
                                AND (P (L) (G (T#1 (L)))))
                           -> (P (F (L)) (F (T#1 (L))))))
             is split into two parts.
             The conjunct
                 (   -(P (G (T#1 (L))) (L))
                  -> (P (F (L)) (F (T#1 (L)))))
             is proved first:
```

The -(P (G (T#1 (L))) (L)) is promoted and a number of forward chainings occur. These include using

(1) -(P (G (T#1 (L))) (L))

with (-(P X$ (L)) -> ((P (T X$) X$) AND -(P (F X$) (F (T X$)))))

to generate

(2) -(P (F (G (T#1 (L)))) (F (T (G (T#1 (L))))))

and

(3) (P (T (G (T#1 (L)))) (G (T#1 (L)))).

Then (2) is used with

((P X$ Y$) OR (P Y$ X$))

to get

(4) (P (F (T (G (T#1 (L))))) (F (G (T#1 (L)))))

which is then used with

(P (F (G (W$))) (F W$))

and (((P X$ Y$) AND (P Y$ Z$)) -> (P X$ Z$))

to generate

(5) (P (F (T (G (T#1 (L))))) (F (T#1 (L)))).

This is then used with

```
                         ((P (F X$) (F W$)) -> (P (G W$) X$))
```

to get

```
(6)                      (P (G (T#1 (L))) (T (G (T#1 (L))))).
```

From (2) and

```
                    (((P X$ Y$) AND (P Y$ X$)) -> (P (F X$) (F Y$)))
```

is generated (by reverse backward chaining)

```
(7)                     ( -(P (G (T#1 (L))) (T (G (T#1 (L)))))

                      OR -(P (T (G (T#1 (L)))) (G (T#1 (L))))).
```

Combining this with (3) gives

```
(8)                      -(P (G (T#1 (L))) (T (G (T#1 (L)))))
```

which contradicts (6). So the hypothesis gets reduced to FALSE.

```
                    (Goal 316)  This has a simple proof:
                        At this point, one of the hypotheses
                        was found to be identical to FALSE
                        and thus this was proved.
                    (Goal 316)
                (Goal 312)  The remaining conjunct is
                    (    (    (P (G (T#1 (L))) (L))
                        AND (P (L) (G (T#1 (L)))))
                     -> (P (F (L)) (F (T#1 (L)))))
                    (Goal 318)  This has a simple proof:
                        This was proved by matching (and
                        possibly chaining) with the
                        hypotheses:
```

Again, this is generated by forward chaining.

```
                         (P (F (L)) (F (T#1 (L))))
                        No substitutions necessary
                    (Goal 318)
                (Goal 312)  The two substitutions of the
                conjuncts combine to form
                No substitutions necessary
            (Goal 224)  The two substitutions of the conjuncts
            combine to form
            Substitutions:
            U$ by L
(Goal 170)  All cases proved.
Adding in the substitution to get the cases gives a
```

substitution of
Substitutions:
U$ by L

REFERENCES

[Ballantyne-75]
Ballantyne, Michael and W. W. Bledsoe, *Automatic Proofs of Limit Theorems in Analysis,* ATP-23, The University Of Texas at Austin, 1975.

[Bledsoe-71]
Bledsoe, W. W., "Splitting and Reduction Heuristics in Automatic Theorem Proving," *Artificial Intelligence* **2**, 1971.

[Bledsoe-72]
Bledsoe, W. W., R. S. Boyer, and W. H. Henneman, "Computer Proofs of Limit Theorems," *Artificial Intelligence* **3**, 1972.

[Bledsoe-74]
Bledsoe, W. W. and Peter Bruell, "A Man-Machine Theorem Proving System," *Artificial Intelligence* **5**, 1974.

[Bledsoe-77]
Bledsoe, W. W. and Mabry Tyson, "Typing and Proof by Cases in Program Verification," *Machine Intelligence 8,* Ellis Horwood Limited, 1977.

[Bledsoe-78]
Bledsoe, W. W. and Mabry Tyson, *The UT Interactive Prover,* ATP-17A, Univ. of Texas at Austin, 1978.

[Boyer-75]
Boyer, R. S. and J S. Moore, "Proving theorems about Lisp functions," *JACM* **22**, 1975.

[Boyer-79]
Boyer, R. S. and J S. Moore, *A Computational Logic,* Academic Press, New York, 1979.

[Bruell-79]
Bruell, Peter, *An Agenda Driven Theorem Prover,* Ph. D. Dissertation, The University Of Texas at Austin, 1979.

[Chang-73]
Chang, Chin-Liang, and Richard Char-Tung Lee, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, Inc., New York, 1973.

[Gelernter-59]
Gelernter, H., "Realization of a Geometry-Theorem Proving Machine," *Proceedings of an International Conference on Information Processing,* UNESCO House, Paris, 1959.

[Good-75]
Good, D. I., R. L. London, and W. W. Bledsoe, "An Interactive Verification System," *IEEE Trans. on Software Engineering 1*, 1975.

[Lenat-76]
Lenat, Douglas B., *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search,* AIM-286, Artificial Intelligence Laboratory, Stanford University, 1976.

[Loveland-78]
Loveland, D. W., *Automated Theorem Proving: A Logical Basis,* North Holland, New York, 1978.

[Meehan-79]
Meehan, James, *The New UCI Lisp Reference Manual,* Lawrence Erlbaum Associates, Hillsdale, N. J., 1979.

[Nevins-74]
Nevins, Arthur J., *A Relaxation Approach to Splitting in an Automatic Theorem Prover,* MIT-AI-Lab Memo 302, MIT, 1974.

[Newell-57]
Newell, A., J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," *Proceedings of the Western Joint Computer Conference*, 1957.

[Nilsson-80]
Nilsson, Nils J., *Principles of Artificial Intelligence,* Tioga Publishing Co., Palo Alto, 1980.

[Oppen-78]
Oppen, D. C., "A $2^{2^{2^n}}$ Upper Bound on the Complexity of Presburger Arithmetic," *JCSS* **16**, 3, June, 1978.

[Robinson-65]
Robinson, J. A., "A Machine-oriented Logic Based on the Resolution Principle," *JACM* **12**, 1965.

[Robinson-79]
Robinson, J. A., *Logic: Form and Function,* North Holland, New York, 1979.

[Teitelman-78]
Teitelman, Warren, *Interlisp Reference Manual,* Xerox, Palo Alto, 1978.

[Tyson-79]
Tyson, Mabry and W. W. Bledsoe, "Conflicting Bindings and Generalized Substitutions," *Proceedings of the Fourth Workshop on Automated Deduction*, Austin, 1979.

[Whitehead-70]
Whitehead, Alfread North and Bertrand Russell, *Principia Mathematica to *56,* Cambridge University Press, 1970.