PROOF METHODS IN AN AGENDA-BASED, NATURAL-DEDUCTION THEOREM PROVER

Mabry Tyson

Artificial Intelligence Center
SRI International, Menlo Park, CA 94025

ABSTRACT

This note describes several methods of finding proofs used in APRVR, an agenda-based, natural-deduction theorem prover. APRVR retains a complete tree of all pending or completed goals and is able to choose the next goal to be processed from an agenda of pending goals. Through this mechanism some proof methods can be utilized that had been unavailable to an earlier prover that was not agenda-based. One approach allows information discovered in one path in an attempted proof to trigger a case split in another part of the attempted proof (**NONLOCAL CASE SPLIT**). Another procedure enables better handling of splitting a conjunction (**AND-SPLIT**) by making it possible to use more information in determining which conjunct should be split off first.

I INTRODUCTION

APRVR ([3]) is based upon earlier work by W. W. Bledsoe on his interactive theorem prover, IMPLY ([2]). Both provers are natural-deduction systems for first-order logic that utilize the concepts of subgoaling, backward chaining, and forward chaining. APRVR's control structure is flat, choosing goals from an agenda, rather than being recursive, as IMPLY is, proceeding from a goal to its subgoals only or exiting to its parent goal. Using an agenda allows the theorem prover to try briefly several possible paths that might lead to a proof, thereby yielding more information about the paths. The theorem prover can then spend more effort on the path that appears most promising until the proof succeeds or the prover decides that the path is not as attractive as was first thought.

APRVR proves theorems in first-order predicate calculus by first applying Skolemization to remove any quantifiers before proving the resultant open formula. The substitutions for free variables required during the proof of a goal (or subgoal) are returned as the value of that proof. If a goal is proved by generating subgoals, the substitutions returned during the proof of one subgoal may be needed for generating the remaining subgoals or confirming that all the subgoals are consistent and can therefore be combined as a proof of the goal.

II AND-SPLIT

When the conclusion of a goal consists of several conjuncts, the goal can be achieved by splitting it into several subgoals, one for each conjunct. In the propositional case, independent proofs of these subgoals suffice to prove the goal. In first-order predicate logic, the possible occurence of existential variables common to several conjuncts complicates matters, so that independent proofs of the conjuncts cannot be combined into a proof of the goal if the substitutions for the variables are in conflict. For example, in proving

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

we can not allow the independent proofs (after Skolemization) of

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

and

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

to be combined because of the conflicting substitutions for the common variable x.

One method (used in IMPLY) to avoid generating the two independent but conflicting proofs is to first find one of them and then apply the indicated substitution to the remaining conjunct before proving it. In most cases, a proof, if indeed any exists, of the remaining goal will not cause a conflict, thus allowing the proof of the original goal to be completed. In the example above, the second goal would become

$$P(a) \wedge Q(b) \Rightarrow Q(a),$$

which is obviously unprovable.

This research was performed at the University of Texas at Austin and was supported in part by National Science Foundation under Grant MCS 80-11417. Any opinions, findings, and conclusions or recomendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

This method of choosing one conjunct, proving it first, and then using the indicated substitutions in proving the remaining conjuncts is a sequential AND-split. Proving the conjuncts one at a time is conceptually simple and works efficiently most of the time.

Unfortunately this sequential nature leads to problems. Let us consider the case in which the first goal has several proofs with differing substitutions for the common variables. If a theorem prover were to find all the proofs it could possibly find, it might waste time trying to find additional proofs after discovering the one that is really needed. On the other hand, it might also waste time if it stops looking for additional proofs once the first has been found and, using the substitutions returned, proceeds to try the (possibly) impossible remaining goals. The theorem prover is trapped into trying the wrong substitution simply because it had been returned by the first proof found. If the theorem prover had found a different first proof (or finds another one after realizing that the first proof did not lead anywhere), the remaining goals might be easily proved. In a recursive system, once the proof attempt on the first conjunct has exited with the first proof, there is no simple way to continue it.

The obvious way to mitigate this problem in an agenda-based theorem prover is to allow the proof of the first conjunct to be continued if the first proof does not lead anywhere. Therefore, if the first proof leads to a trapping substitution, a second proof would be sought. While this is easy to describe, it is difficult to put into practice. Blindly allowing the reactivation of goals that have been proved once — simply because there might have been another satisfying substitution — leads to much wasted effort. It became clear to me in my experimentation that restarting already proved goals usually caused extraneous attempts to reprove goals that did not need a different proof.

A less obvious method for reducing the trapping problem also takes advantage of the agenda mechanism. Instead of deciding which will be the first conjunct to be proved, attempt to prove each of the conjuncts individually, treating it as though it were the first. If the first proof found for any of the conjuncts uses the substitutions necessary for the desired solution, that solution will be found without having to restart any already proven goal.

However, this technique entails duplication of effort. If a goal has N conjuncts, N! subgoals could eventually be created. Thus, rather than letting this technique run unchecked, APRVR creates goals for each conjunct, schedules them on the agenda, and also schedules an additional goal on the agenda for subsequent examination of the results of those goals after some effort has been expended on them. If some of the subgoals have then been proved, the associated secondary subgoals (created by applying the returned substitutions to the remaining conjuncts) are allowed to remain active while the attempts to prove the other conjuncts are deactivated. If none of the initial subgoals have been proved, one of the conjuncts is chosen as the first conjunct while the other proof attempts are deactivated.

This technique should be valuable wherever some of the conjuncts are critical in assigning substitutions, while others simply verify that the result of applying the substitutions has certain common properties. A simple example in group theory would be a goal whose conclusion is the conjunction

$$x \cdot a = a \cdot x \ \wedge \ x \cdot a = e,$$

where e is the identity element, x a variable, and a a constant. The first conjunct may have many satisfying substitutions (e.g., replacing x with e) while the second one is more critical. With this technique, both conjuncts would be tried; the quick solution for the second conjunct (substituting a^{-1} for x) would be found and applied to the first conjunct, thereby leading readily to a proof.

III NONLOCAL CASE SPLIT

A very interesting use of the agenda is the nonlocal generation of case splits. In attempting to prove a theorem, people often "paint themselves into a corner" — that is, they get to a point at which, in some cases, the subgoal in question is false. For the other cases, the subgoal is provable. As an example, a proof of a theorem in field theory might reduce to proving $(a \cdot b = a \cdot c) \Rightarrow b = c$. Two cases exist, $a \neq 0$ and $a = 0$, of which the first is provable but not the second. At this point a person will consider why one case is impossible and will ascertain the reason for this. He will then back up in his proof and try a different proof for that case. In this example, there may be another proof to prove the case where a is 0 (or it may prove that a cannot be 0).

In certain situations (such as when triggered by an OR in the hypothesis or lemmas), APRVR will try to prove a goal by doing a case split; for each case, a separate subgoal is generated with that case as an added hypothesis. When starting a case split, APRVR determines the amount of effort it wants to expend on that goal before concluding that it is perhaps not provable. It then reschedules the goal that generated the case split for activation when that effort has been completed. If all the cases are proved beforehand, the goal is proved and the value is returned. But, if APRVR does not prove the goal within the allotted time, a different procedure must be attempted (but the present case split is left on the agenda and could still succeed). If some of the cases have been proved and the case split is ground (ie., contains no variables), APRVR will try the case split earlier in the proof in the hope that the failing cases will

be provable from a higher point in the tree. (The provable cases can always be proved by the same path as before.) Consequently, APRVR will back up along the path of the proof to an appropriate point, where it will then attempt the case split.

As an example, APRVR might decide to split on whether a constant, a, is less than, greater than, or equal to 0. APRVR would create three subgoals and enter them in the agenda along with their parent goal, which is now rescheduled. Suppose the second two subgoals get proved quickly, but the first subgoal, involving $a<0$, has not been proved before the parent goal again reaches the head of the agenda; APRVR will then look for a higher goal on which to try the case split. The only new proof to generate there is the proof for the case $a<0$.*

APRVR is not as intelligent as a human theorem prover, so it can not make as clever a decision about where to try the case split. At present, APRVR examines the structure of the proof and backs the case split as far up the proof tree as possible until a goal is reached at which backing up any farther would reach a goal that would not necessarily be provable for those cases that were successful at a lower level. This might be just below an AND-split goal if APRVR had been working on the first branch of the AND-split and had yet to do the second branch. 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 (since it is not known yet whether the second branch is provable).

Assuming the case split is necessary for a proof, the chosen point may not be optimal — yet not be a bad choice. If the proper point for the case split is below that selected, only a little extra work is required in passing the cases down to where the case split is necessary. If the proper point is actually above where APRVR chooses, then the proofs of the chosen goal for whatever cases do work will be reused when they are requested from above. The advantage of choosing where APRVR chooses, and no higher, is that any higher goal may not be provable for any of the cases. Further analysis of what substitutions are made, as well as where and why some of the cases fail, might lead to a better choice of a goal on which to try the case split.

IV OTHER PROPOSED METHODS

There are surely other methods available to a prover with an agenda mechanism like that of APRVR. More nonlocal methods might be found in those instances in which information from one part of the proof might affect what is being done elsewhere. Perhaps if a goal were found to be false (possibly by a counterexample), APRVR could trace back up the proof tree to the point at which a false ancestor of this goal had been generated. Other, similar goals might also be purged.

APRVR uses demon goals to monitor the progress on AND-splits and case splits. These goals are placed on the agenda so that, when they are chosen, APRVR pauses in its normal attempts at finding a proof and stops to analyze what is happening in the part of the proof the demon goal was watching. These demon goals might be used in other contexts to monitor what APRVR is doing. While the idea of self-monitoring is not unique to agenda mechanisms, it was rather easy to implement by scheduling the demons in the same way as ordinary goals.

Since APRVR keeps the entire proof tree, if one goal is found similar (analogous) to another, the successful proof of one could be used as a guide in proving the other and modified wherever the similarity broke down. There are a number of problems to be overcome to accomplish this, but it should be possible.

SUMMARY

Although I had expected the agenda mechanism's major source of power to be the ability to make a better choice of paths, I discovered new heuristics that would aid in finding proofs that were not possible in the recursive IMPLY theorem prover. The only drawback of the agenda-based APRVR was a relative weakness in man-machine interaction, which is one of IMPLY's strengths. This weakness consists of the agenda system's tendency to change contexts more freely than is done by people. APRVR was capabile of having all the power of IMPLY (although certain extensions to IMPLY were not implemented) — and even more. It proved a number of standard problems given to theorem provers; the most difficult new problem (suggested by W. W. Bledsoe) proved by APRVR comes from part of a proof that a continuous function attains its minimum over a closed region.

AM8:

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

Although this problem depends heavily on inequalities, no special-purpose machinery was incorporated (which might have expedited the proof). APRVR used

*Although the examples given are for cases involving inequalities, this nonlocal case split mechanism works on any type of case. The backing-up of unproved cases in [1] is somewhat related, but is limited to inequalities.

both the AND-split and nonlocal case split mechanisms presented here in finding the proof without any human intervention.

REFERENCES

[1] W. W. Bledsoe and M. Tyson, "Typing and Proof by Cases," in *Machine Intelligence 8,* D. Michie, ed. (Ellis Horwood Limited, Chichester, Sussex, England, 1977).

[2] W. W. Bledsoe and M. Tyson, "The UT Interactive Prover," ATP-17A, University of Texas at Austin, Austin, Texas (1978).

[3] W. M. Tyson, *APRVR: A Priority-Ordered Agenda Theorem Prover,* Ph.D. Dissertation, University of Texas at Austin, Austin, Texas (1981).