

Subsumption, a Sometimes Undervalued Procedure

Larry Vos, Ross Overbeek, and Ewing Lusk

1 Motivation and Background

Rarely do circumstances permit one to simultaneously acknowledge a colleague's contributions, evaluate the relative importance of the basic elements of a field, and compare various paradigms for attacking a class of problems. However, through a fortuitous set of occurrences, we have that opportunity. We shall, therefore, take full advantage of the situation by presenting an evaluation of J. A. Robinson's contribution of *subsumption* [RobinsonJ65a] to the field of automated reasoning.

We shall also take the opportunity to compare various approaches to the automation of reasoning, citing evidence to support the position that those paradigms that do *not* rely on the use of subsumption suffer a severe disadvantage when compared to those paradigms that do. The evidence, which we cite in section 4, is gleaned from a number of experiments that focus on a set of theorems of increasing difficulty. Even though we sprinkle this article throughout with observations some of which may be considered philosophy, we recognize that data of the type we present in section 4 is the final and ultimate test. In particular, when measuring the power of a given paradigm or a given automated reasoning program, the most important question concerns the likelihood of success when one attempts to obtain a proof or complete an assignment. The second most important question focuses on the ease, usually measured in CPU time, of obtaining the proof or completing the assignment.

In addition to establishing the vital nature of subsumption—recognizing the danger of our intention—we intend in this article to correct history for some, and perhaps for many. Specifically, although Robinson is best known for his formulation of binary resolution [RobinsonJ65a] and next known for his introduction of hyperresolution [RobinsonJ65b], his contribution of subsumption [RobinsonJ65a] proves to be, in the context of automated reasoning circa 1989, his most important. We address this topic more fully in section 2.

Of the various arguments one might give to substantiate such a—in the minds of some—radical claim concerning the relative importance of Robinson's contributions, the simplest rests on the fact that no substitute for subsumption exists that offers nearly comparable power, which contrasts with the fact

Reprinted from *Festschrift for J. A. Robinson*, ed. J.-L. Lassez and Gordon Plotkin, MIT Press, Cambridge, Mass., pp. 3–40 (1991). Published with permission of the MIT Press, Cambridge, Mass., 1991.

that a number of inference rules now exist that offer far more power than does either binary resolution or hyperresolution. Not only does the use of subsumption purge the database of information that is logically weaker than existing information and information that is therefore not needed (because of the way unification works), even more important, the use of subsumption prevents the spawning of descendants (immediate and later) of the unneeded information. Granted, Robinson's formulation of binary resolution and hyperresolution laid the foundation for much research focusing on inference rules, and he deserves substantial credit in that regard. But where would automated reasoning programs be now (in 1989) were it not for subsumption? In this article we shall show why the correct answer is that such programs would be essentially nowhere, and, in particular, we strongly suggest that approaches that do not take advantage of subsumption suffer accordingly.

Before turning (in sections 2 through 4) to the meat of this article, let us first provide in this section the basis for a deep appreciation of the importance of subsumption. We note that, although we frequently focus on clauses as the form in which information is retained, comparable remarks apply for other paradigms for the automation of reasoning. To complete one's appreciation for the importance of subsumption, we discuss in section 2 subsumption in relation to the size of deduction steps, and then in section 3 we give some history to set the stage for the experimental evidence we present in section 4. In addition to that evidence, we shall also offer (in Section 4) certain challenges to those researchers interested in testing the power of their programs and also to those researchers who prefer a paradigm (for the automation of reasoning) based on logic programming.

A full appreciation of the importance of subsumption begins with the observation that when one is relying on the assistance of an automated reasoning program, one frequently encounters a common annoyance, an occurrence that can waste a considerable amount of computer time; specifically, a reasoning program can deduce the same piece of information many, many times. A case in point is a lemma from lattice theory, a lemma called SAM's lemma, which is an acronym for "semiautomated mathematics" (see test problem 12, section 6.4.1 of [Wos87]). When a successful attempt was made to prove that lemma using hyperresolution and a notation that deemphasizes the role of equality, more than 6,000 clauses identical to retained clauses were generated and then discarded. (This lemma, discovered by Guard's program [Guard69], is considered to be the first example of a contribution to mathematics made by a reasoning program; later contributions include the work on complete sets of reductions begun in the middle 1970s and the studies that led to answering specific open questions from mathematics [Chou88, Winker79, Winker81, Wos85] and from logic [Wos83, Wos84b, Wos88].) The deduction of information that is already known certainly produces no satisfaction.

Perhaps less annoying but in most cases more likely—although not in the case of the cited example focusing on SAM's lemma—is the deduction of a

trivial instance of some already-deduced fact. For example, far too frequently, $0 + a = a$, $0 + b = b$, and the like are deduced and then discarded as instances of the already-present fact that $0 + x = x$. Indeed, during the successful attempt to prove SAM's lemma just cited, almost 5,000 additional clauses were discarded because they were proper instances of clauses already retained. Both classes of information—identical copies and proper instances of retained information—are classified as redundant information (see section 2.3 of [Wos87]).

Incidentally, for those who might be curious about why our figures concerning the number of clauses that are subsumed are higher than various figures published elsewhere, the explanation rests with the omission in earlier experiments of closure (totality) axioms. For example, in group theory when the chosen notation is the so-called *P* formulation, the axiom (in clause form) $P(x, y, \text{prod}(x, y))$, sometimes referred to as the closure axiom, is often omitted from the presentation of a problem to an automated reasoning program. Because one cannot in general know a priori which axioms are required for a proof of a given theorem, no justification exists—at least in our view—for the omission of axioms that are independent of the others that are included, and closure axioms are usually independent of the other axioms for a theory. Removal of the axiom $P(x, y, \text{prod}(x, y))$ from the *P* formulation of the axioms for group theory—a practice often encountered—coupled with the addition of clauses used to attempt to prove the commutator theorem (see section 4) in fact produces a satisfiable set of clauses. In other words, without the closure axiom, one cannot prove the commutator theorem (in the *P* formulation). Therefore, we find the practice of omitting the appropriate closure axioms without an excellent defense to be unjustified and often to produce misleading information. In particular, one can be misled about the possible provability of some given result. Even worse, one can be misled about the power of some chosen automated reasoning program.

If a method could be found to discourage deducing redundant information—for more reasons than the fact that subsumption would come into play far less—the effectiveness of automated reasoning programs would be sharply enhanced. The enhancement would not result directly from the CPU time saved by generating far fewer clauses, for clause generation in itself is not that expensive. Instead, the enhancement would result from sharply reducing the CPU time consumed by subsumption, demodulation, and the like, each applied to a clause after it is generated. One of the most appealing approaches to saving the CPU time consumed by processing redundant information focuses on formulating an appropriate strategy to sharply curtail, if not actually eliminate, the generation of such clauses (see research problem 6 of [Wos87]).

Until such a strategy is found—in view of the cited results from the experiment with SAM's Lemma and from the experiments we shall present in section 4, which establish that some means must be employed to avoid wasting a great deal of CPU time—an automated reasoning program requires some means to cope with redundant information. Neither of the following two approaches—both of which obviously differ sharply from the use of subsumption—is remotely satis-

factory. In one approach, the program is simply instructed to retain the redundant information—the duplicate clauses and those that are instances of other clauses—in the database of information. In the other approach, essentially no new information is retained. In other words, the program considers the assignment in the spirit of logic programming. Although logic programming is clearly a remarkable aid for solving problems and answering questions of a certain type—those that are algorithmic in character—without the sophisticated use of strategy and the use of procedures such as subsumption and demodulation, in most cases this powerful programming language does not offer the needed characteristics for proving even moderately difficult theorems from mathematics.

To see what can go wrong with the first of the two given approaches, one can imagine instructing an automated reasoning program to solve a problem in which equality is present, where the chosen program simply retains each item of deduced information even if redundant. With such an approach to coping with redundant information, a reasoning program can fail to complete assignments that are quite trivial, where the failure is due directly to the extra information that serves no real purpose. At the most obvious level, the extra and unneeded information—for example, $a = a$, $\text{product}(b, c) = \text{product}(b, c)$, and $\text{inverse}(c) = \text{inverse}(c)$ —can easily be chosen as the focus of attention, sending the program on one fruitless path after another. In contrast, by using subsumption, the clause equivalents of such unneeded information would be purged, for the clause $\text{EQUAL}(x, x)$ for reflexivity of equality would typically be present, and that clause would subsume the unneeded clauses. As a consequence, the program would be prevented from even considering the corresponding fruitless paths.

If the second approach to coping with redundant information is adopted—that of retaining *no* intermediate information—a quite different hazard is encountered. In particular, failure to retain information (such as that corresponding to powerful lemmas) can cause a program to pursue a vast number of worthless paths in search of the key result. For example, if the program that does not retain new information, and hence does not use subsumption, in effect deduces that $\text{inverse}(\text{inverse}(a)) = a$ or a clause containing the equivalent literal, then that program is free to pursue the corresponding path in search of potentially valuable results. In contrast, with subsumption, the deduction and retention of the lemma asserting that the inverse of the inverse of x is equal to x would result in the immediate purging of those clauses that lead to the type of fruitless path under discussion.

An example of the approach that retains no new information—and one that is currently receiving substantial attention—relies on a paradigm based on logic programming. With such a paradigm—although the corresponding analysis is not quite so straightforward, in part because lemmas that take the form of a positive unit clause are not encountered—the lack of clause retention and the inaccessibility to subsumption can result in the pursuit of many fruitless paths. For example, an attempt to solve the subgoal $\text{-P}(\text{prod}(x, y), \text{prod}(\text{inv}(y), \text{inv}(x)), e)$ may result in attempting to solve a second subgoal that is an identical copy of

the first subgoal or a proper instance of that subgoal. When such occurs, the secondary task should be abandoned, somewhat in the spirit of subsumption. Without clause retention and the use of subsumption, a paradigm based on logic programming is subject to pursuing a large number of paths of the type just exemplified (discussed in more detail in section 2.3). In other words, one cannot simply bypass, without cost, the need to cope with redundant information. (For a rather detailed discussion of this topic, see section 2.2.3 of [Wos89]).

Regardless of how one approaches the automation of reasoning, when attempting to answer a deep question or solve a hard problem, some way must exist for coping with the myriad conclusions that will inevitably be drawn. Therefore, an automated reasoning program should be given the capacity to retain new information, such as general lemmas that it has found, and yet be in a position to purge its database of information that is captured by other bits of information already present.

Robinson's contribution of subsumption—although, just at the surface, a simple and obvious notion—is the best solution for purging a database of unwanted information. In addition, discarding redundant information meshes beautifully with inference rules that enable an automated reasoning program to focus directly on statements involving universally quantified variables with the objective of deducing, where possible, statements in which such variables occur. By purging information captured by existing information, an automated reasoning program cannot be led astray by the captured information; more important, it cannot be led astray by all of the descendants—immediate and later—of such information. In short, as we shall show throughout this article, Robinson's contribution of subsumption merits the highest accolades, especially if viewed in the context of automated reasoning circa 1989, for the use of subsumption is essential for the study of even moderately difficult questions from mathematics and logic.

One might immediately point out that subsumption can be very costly. Indeed, many experiments with different reasoning programs have shown this cost can be substantial—until now. William McCune, with his formulation of *discrimination trees* [McCune89], has reduced the cost of forward subsumption—at least with regard to unit clauses—to a far less significant figure. For example, in an experiment focusing on the formula XHK , $XHK = e(x, e(e(y, z), e(e(x, z), y)))$, a formula that provides a complete axiomatization for equivalential calculus, the use of discrimination trees by McCune's program OTTER [McCune90] reduced the time used by forward subsumption by a factor of 95. Even more important than the obvious improvement in program performance is the second-order effect on one's experimentation and research. In particular, by completing each of a number of experiments in less than 10 CPU minutes, in contrast to the more than 4 CPU hours that each would have required, we succeeded in formulating a systematic approach for the pertinent study, an approach that might easily have eluded us were it not for the dramatically improved turnaround. Soon—or so it appears—the cost for back subsumption will also be reduced correspondingly

by relying on an extension of McCune's use of discrimination trees. In other words, one of the main objections, and perhaps the only serious one, to the use of subsumption will soon be sharply reduced in importance. Why then—as discussed in research problem 6 of [Wos87]—is it still important to find a strategy for avoiding or sharply reducing the generation of redundant information?

The answer rests with the various processes applied to each newly generated clause or newly drawn conclusion. Among such processes are those for canonicalization (demodulation [Wos67]), for evaluating significance (weighting [McChar76]), for index building, and for information integration. Those additional processes, which should be applied to newly generated clauses because their use is more beneficial than their cost, are nevertheless expensive. In other words, although clause generation is not in itself particularly expensive, and although subsumption—if all goes as planned—may become relatively inexpensive, automated reasoning programs would be considerably more efficient if the generation of redundant information were substantially reduced. The remarkable efficiency that many reasoning programs already exhibit—as we shall discuss throughout the remainder of this article—rests in part with the use of subsumption.

2 Subsumption in Its Relation to the Size and Nature of the Deduction Step

At this point we focus on the need for subsumption in the context of different types of reasoning and different approaches to the automation of reasoning. At one extreme, a program can attempt to imitate the manner in which people frequently reason, for example, relying heavily on instantiation (see chapter 4 of [Wos87]). At the other extreme, a program can reason in a manner that a person might well find unnatural or even unpleasant, for example, relying heavily on paramodulation [Robinson69, Wos73, Wos87] for treating equality as if it is “understood”.

From a different perspective, at one end of the reasoning spectrum, a program can rely on an inference rule such as binary resolution, which takes deduction steps of essentially an identical size, and rather small ones at that. Near the other end of the spectrum is linked inference [Wos84a], which gives the user the option of setting the size of the deduction step and the option of using semantic criteria. And at the end of the spectrum is an approach based on logic programming [Stickel88], which entirely avoids the retention of new information.

Depending on the particular approach to the automation of reasoning, the direct need for and the obvious value of subsumption vary widely. For an example of an approach for which subsumption appears to be irrelevant, we need only look to automated reasoning programs based on emulating logic programming. Since such approaches typically retain no information, subsumption plays no (direct) role; in section 2.3 we shall see that this issue is cloudier than it first

appears. However, in those cases where subsumption appears to be of little or no value, one discovers that such approaches often lack the needed power for attacking deep questions and hard problems, especially if the questions and problems come from mathematics or logic.

To develop this point and related points more fully and to amplify our comment in section 1 that when viewed in the context of automated reasoning circa 1989, Robinson's contribution of subsumption is far more important than either his contribution of binary resolution or his contribution of hyperresolution, we pose the following six questions, which we answer in this article.

1. Why not design and implement an automated reasoning program whose reasoning imitates a person's? For example, what obstacles are presented by the use of instantiation? After all, in the classical approach to artificial intelligence and in certain efforts currently advocated, one finds an emphasis on the emulation of people for the reasoning aspects of problem solving.
2. Why not rely heavily on binary resolution? After all, experiments and intuition each suggest that binary resolution is far superior to instantiation.
3. In contrast to using binary resolution with the potential of retaining a great deal of information, why not rely on an approach based on logic programming—an approach currently receiving substantial attention—and have the reasoning program retain no new information?
4. What are the strengths and weaknesses of hyperresolution, the inference rule introduced by Robinson [RobinsonJ65b] with the object of offering substantially more power than binary resolution does?
5. Is there some point on the reasoning spectrum between retaining too much information (as often occurs when the reasoning program relies heavily on using binary resolution) and retaining no information (as occurs with an approach based on logic programming) where the user of the reasoning program can play an active role in deciding what types of information to retain and what size of deduction step to take?
6. Finally, one of the most important questions, Why do we consider subsumption to be Robinson's most important contribution to automated reasoning circa 1989?

Let us consider and answer the six questions in the order given. Then let us turn (in section 3) to some history and comparative analyses to complete the stage setting for presenting (in section 4) the results of our experiments concerning the value of, and even further the need for, subsumption.

2.1 Person-Oriented Reasoning

Our treatment of the first question—that concerning the weakness of relying mainly on person-oriented reasoning—will clearly be in part philosophical. Nevertheless, because automated reasoning is already receiving rapidly growing interest and soon will offer substantial power and wide applicability, we take this opportunity to address the subject of person-oriented versus computer-oriented reasoning. By discussing this dichotomy, which will entail some elementary observations, we can take a brief but hard look at one extreme point on the spectrum for the automation of reasoning. This discussion will also provide some of the material that shows why we prefer the paradigm on which we rely for our experiments, and it will also provide the basis for one of the many reasons we give for the indispensability of subsumption.

Our discussion is in part motivated by the desire to briefly focus on the difference in emphasis found in classical artificial intelligence versus that found in automated reasoning. We also wish to provide some hints concerning the weakness resident in approaches—designed to solve the deep and difficult problem of how to effectively automate reasoning—that are based on an inexpensive or quick or simply appealing solution. We provide further hints in this regard when we give our answer to the third question, that focusing on the use of a paradigm based on logic programming. Finally, our discussion is also motivated by the desire to pay tribute to Robinson for his emphasis on the use of universally quantified variables and on the corresponding use of unification.

Before we give our answer to the question under discussion, let us make clear that, although we are not in favor of the approach taken in classical artificial intelligence, we nevertheless are convinced that a study of how people reason may indeed lead to the discovery of useful clues to increasing the effectiveness of automated reasoning programs. The error, in our view, is to mistake the discovery of such clues for a strong suggestion that automated reasoning programs are best designed if they *emulate* a person's reasoning. A far more promising course is to use those clues—and any others one may find—to suggest computer-oriented methods of reasoning, possibly including methods that capture and generalize certain aspects of person-oriented reasoning. Therefore, we heartily endorse a wide range of studies, including the study of person-oriented reasoning; the useful clues are clearly hard to find, and one should look for clues wherever one's intuition dictates.

The simplest answer to the first question—and a partial explanation of why we do not endorse the approach taken in classical artificial intelligence—is that people and computers are totally different entities, and, to us, no obvious justification exists for attempting to imitate, in contrast to studying, a person's reasoning. Given our view, no surprise will be produced by our lack of endorsement for the classical approach to artificial intelligence with its emphasis on emulation of methods that people employ. Instead, although the better mathematicians—Hilbert, Gödel, and von Neumann, for example—are justly revered for the power

of their minds, we prefer to concentrate on designing and implementing an automated reasoning program that complements a person. In contrast, if one were to focus on designing a program relying on person-oriented reasoning, then the inference rule *instantiation* might come heavily into play. With some elementary examples, let us quickly illustrate two weaknesses of that inference rule.

The first and obvious weakness of using instantiation rests with its fecundity. In particular, compared to binary resolution and the reliance on universally quantified variables and unification, a program using instantiation and syllogistic reasoning at the *ground level* can deduce an inordinately large number of conclusions. For example, from “if x is a mother then x is female” and “if x is female then x is not male”, the use of binary resolution yields the equivalent of the obvious fact that “if x is a mother then x is not male”. In contrast, with instantiation, an automated reasoning program can deduce myriad instances of each of the two given facts and also—with ground-level syllogistic reasoning—of the given obvious conclusion, an instance for every person mentioned in the problem under consideration and also for every relative of those people, and more. How marvelous of Robinson to change the emphasis in 1963 from the consideration of ground clauses to the consideration of clauses in which universally quantified variables are present. Even further, how significant to move the focus—by emphasizing the use of unification—to the deduction of clauses containing variables of the same type. With regard to the given example, what an achievement for Robinson—to formulate the inference rule binary resolution to deduce a single conclusion that captures all of those conclusions that would otherwise result from using instantiation coupled with syllogistic reasoning. Even further in the direction of capturing unneeded information, but with the objective of discarding such, we have Robinson’s more significant contribution of subsumption for which no substitute currently exists.

Of course, binary resolution usually does not go far enough—does not offer enough deductive power—which is why Robinson later formulated hyperresolution. On the other hand, as we discuss in section 2.3, one can go too far, for example, by relying on one of the dialects of logic programming in which no new information is retained.

The second weakness of relying on instantiation rests with the lack of an effective strategy for choosing instances that are fairly likely to play a useful role. For a simple example, if asked to prove commutativity for groups in which the square of every element x is the identity e , the mathematician wisely chooses the instance $(yz)(yz) = e$ on which to base a proof. In contrast to the given example of the mathematician’s ability, currently no strategy exists that enables an automated reasoning program to choose—with sufficient wisdom—from among the myriad instances presented by even fairly difficult theorems. But, for controlling binary resolution and the other inference rules commonly used in many automated reasoning programs, fairly powerful strategies are frequently employed, which makes the use of such inference rules far more effective than the use of instantiation. The need for strategy that restricts and for strategy that directs a

program's reasoning can hardly be doubted if one attempts to prove even moderately difficult theorems from mathematics or logic. However, for those who are skeptical of this need, we offer the challenge of trying to obtain—without using a restriction strategy such as the set of support strategy [Wos65] or a direction strategy such as weighting [McCharen76]—proofs of the theorems we cite in Section 4. If that challenge is accepted by thoroughly testing the power of reasoning programs that rely on a paradigm based on logic programming, we conjecture that one will discover that such programs are not ordinarily adequate for attacking deep questions.

2.2 Binary Resolution in 1989

In addition to answering the second of the six questions posed earlier—that concerning reasons for not relying heavily on binary resolution—the material in this section supports one of the key aspects of our position that Robinson's contribution of subsumption is his most important, at least for automated reasoning of today. Specifically, in this section we briefly focus on why binary resolution is no longer Robinson's most significant contribution.

Robinson's introduction of binary resolution in its full generality in the summer of 1963 at Argonne National Laboratory marked a singular advance in automated reasoning, then known as mechanical theorem proving. Indeed, Robinson's emphasis on the use of unification played—and still plays—a key role in the entire history and rapid advance of automated reasoning. We note that, in contrast to the published version of binary resolution [RobinsonJ65a], Robinson—when he first presented the rule in its full generality at Argonne—also introduced the inference rule factoring [see Chapter 4 of Wos87]; we return to this topic shortly.

Binary resolution, which nicely captures and generalizes both modus ponens and syllogism, provides an excellent example of the type of reasoning that emphasizes the importance of drawing general conclusions, extends the type of reasoning people sometimes employ, and adapts well to computers. Paramodulation, which (by generalizing equality substitution) also extends the type of reasoning people sometimes employ, provides an even more striking example of computer-oriented reasoning that is *not* person-oriented—a type of reasoning, in fact, that a person ordinarily finds unnatural. Unfortunately, Robinson's published version of binary resolution—in contrast to the version that he presented in the summer of 1963 and that appeared earlier in a preprint—hides the need for the inference rule factoring, at least from the viewpoint of implementation. In particular, the published version of binary resolution permits more than one literal in one clause to be unified with more than one literal in the other clause under consideration. In contrast, in the preprint, binary resolution was required to focus on exactly one literal in each of the two chosen clauses, and factoring was defined to fill the void.

To illustrate how the published version in effect hides the need for factoring,

we need only consider the canonical example consisting of the following two clauses.

$$\begin{array}{l} P(x) \mid P(y) \\ -P(w) \mid -P(z) \end{array}$$

If one restricts the reasoning to applying only the version of binary resolution that appeared in the preprint—which is the version found in the majority of automated reasoning programs that offer the rule—then no proof can be found starting with the given set of two clauses even though that set is unsatisfiable. However, as is well known, if one is allowed the use of factoring, then a proof is quickly obtained, for factoring applied to the first clause yields the clause $P(x)$, whose use leads, in one binary resolution step, to unit conflict.

On the other hand, if one revisits the given set of two clauses but restricts the reasoning to the version of binary resolution in which more than one literal can be unified in each of the two chosen clauses, then one can obviously unify both literals in the first clause with both literals in the second to obtain the empty clause. Such an action, however, in effect relies on factoring each of the two clauses in order to unify two literals in one with two literals in the other. The given example is in fact representative of what in effect occurs if one implements the published version of binary resolution. As is obvious, the analysis does not involve, in any way, the distinction of proof termination based on unit conflict versus that based on the use of the empty clause.

Regardless of which definition of binary resolution one prefers, the important point about this inference rule is that its use enables a reasoning program to reason directly from statements containing universally quantified variables to yield (depending on the nature of the corresponding unification) statements containing such variables. Although this inference rule provided an excellent beginning for the field and is still useful in certain situations, in the context of automated reasoning circa 1989, the rule is far less useful than other available inference rules. For example, UR-resolution [McCharen76] with its emphasis on unit clauses and hyperresolution [RobinsonJ65b], with its emphasis on positive clauses, are ordinarily far more useful and far more powerful. Nevertheless, just as the electron was thought to be a basic particle of physics, similarly early in the history of the field one might have thought that binary resolution would be the basis for automated reasoning. Although not the case, binary resolution can be thought to be a basic element of which many, but not all, inference rules are composed. Of course, even more than binary resolution, the common strand throughout automated reasoning is the use of unification.

The reason that binary resolution is no longer so useful rests with the size of the deduction step a reasoning program makes when this rule is successfully applied. Simply stated, the step is almost always too small—too small in the sense that the corresponding information is only a fragment of the conclusion one is seeking. For example, if stated in ordinary linguistic terms, the deduction of

if (Q and R) then S
 from

P and if (P and Q and R) then S

is typically a fragment of the desired conclusion S . Indeed, as the experienced researcher in automated reasoning recognizes, the given example is in effect an example of a hyperresolution step and one of the three binary resolution steps that one can take to produce the corresponding conclusion.

Not to be unkind but rather to give some insight, in many cases hyperresolution is to binary resolution as binary resolution is to instantiation: binary resolution simply lacks the power needed to reason efficiently in the context of most applications today. It is in part this observation that causes us to take the position that binary resolution is no longer Robinson's most significant contribution to automated reasoning.

2.3 The Logic-Programming Paradigm for the Automation of Reasoning

Before resuming our primary case for the fact that subsumption is the most significant contribution made by Robinson—and a contribution that is indeed outstanding—we turn to the third question, that concerning the reasons for *not* relying on an approach based on logic programming. We must address this issue with some delicacy, for we have high regard for some of the researchers who advocate such an approach. Nevertheless, because this paradigm is currently receiving substantial interest [Stickel88, Plaisted88], because good science requires that researchers with evidence and experience speak openly about the strengths and weaknesses of new trends, and because subsumption is not obviously relevant for this paradigm, we devote a separate subsection to the value of logic programming to the automation of reasoning.

In our view, if one wishes to attack deep questions and hard problems—especially if the questions come from mathematics or logic—then reliance on a paradigm heavily based on logic programming is an unwise choice. In contrast, for problems in which one has an algorithm that quickly homes in on a solution rather than searching for the needed information, logic programming does indeed offer excellent performance. Unfortunately, for most of mathematics and logic, one does not have the luxury of such an algorithm. Instead, one is forced to search—preferably with the aid of a sophisticated strategy—a potentially large space of conclusions. Indeed, it is the size of this search space that causes W. W. Bledsoe [Bledsoe87] to assert that too many of the questions in mathematics will still be out of reach if all that is changed is access to a computer 1,000 times faster than the fastest currently available—a view we share.

Perhaps part of the appeal of a paradigm based on logic programming rests with its clean lines. For example, with such a paradigm, no effort is required to implement and no CPU cycles are required to execute subsumption (to purge redundant information), demodulation (to canonicalize information), and weight-

ing (to choose where next to focus the program's attention). When all is equal, the cleaner and simpler the design of a program the better. Unfortunately, for automated reasoning, all is not equal. The nature of deep questions and hard problems whose answers or solutions depend on logical reasoning typically requires—for their answers or solutions—the use of subsumption, demodulation, and strategy of various types.

Since this article focuses heavily on subsumption, let us briefly discuss an automated reasoning program's need for subsumption when the program is assisting in a serious study of mathematics or logic. Simultaneously, let us show why the avoidance of subsumption by programs relying on a paradigm based on logic programming materially weakens their effectiveness. We begin our discussion with the case in which the automated reasoning program retains no intermediate clauses—in the style of logic programming.

On the surface, one might say that the subject of subsumption is irrelevant for such a program, but such a position is flawed. In particular, if such a program—while considering, for example, a theorem from elementary group theory—attempts to solve

$$-P(\text{inv}(\text{prod}(\text{inv}(x),y)),\text{inv}(x),\text{inv}(y))$$

as a subgoal, the program may later attempt to solve

$$-P(\text{inv}(\text{prod}(\text{inv}(a),b)),\text{inv}(a),\text{inv}(b))$$

as another subgoal, then solve

$$-P(\text{inv}(\text{prod}(\text{inv}(c),d)),\text{inv}(c),\text{inv}(d))$$

as a third subgoal, and then other numerous instances of the more general subgoal. Potentially, what a waste of considerable computer time, even to the point of possibly preventing completion of the given assignment! Indeed, as our experiments cited in section 4 show, without subsumption, our programs are subject to a similar disaster. With subsumption, on the other hand, the retention of intermediate clauses would enable an automated reasoning program based on logic programming to avoid pursuing so many fruitless paths. The efficiency would be increased even more dramatically if demodulation were also employed.

Recognizing the value of retaining intermediate clauses, some reasoning programs employ a modification of the paradigm based on (pure) logic programming. For such programs, subsumption—as shown by our experiments—plays a vital role by discarding redundant information, copies of and instances of existing clauses. However, if a program does not also use demodulation to rewrite information into a desired canonical form, then a class of information—related to what we are calling redundant—is encountered. For example, again borrowing from group theory, once a program has solved

$$-P(a,b,c)$$

as a subgoal, the program should not be forced to solve each of

$$-P(\text{prod}(e,a),b,c)$$

and

$$-P(a,\text{prod}(e,b),c)$$

and

$-P(\text{prod}(e, \text{prod}(e, a)), b, c)$

and the like as subgoals. Without a procedure such as demodulation—as without subsumption—we again have the potential of extreme ineffectiveness.

In contrast, with demodulation and specifically the demodulator

$\text{EQUAL}(\text{prod}(e, x), x)$

among the input clauses, a far different story is told. For those who enjoy history, we recommend chapter 2 of [Wos87] which contains an anecdote describing how demodulation was discovered, an anecdote that focuses on an example nearly identical to that just cited. Of course, subsumption again comes into play in a vital way, for its use purges various clauses that result from demodulating a clause to produce a copy or an instance of an existing clause.

Since we are obviously building our case *against* the use of a paradigm based on logic programming—at least when the objective is assistance with answering deep questions and solving hard problems—and at the same time building a case *for* the retention of clauses, let us look at a third class of information one might wish to avoid. We recall that the first class of information consists of clauses that are purged by subsumption because they are copies of or instances of existing clauses; the user's actions and tastes have essentially no effect on the purging. On the other hand, the user's actions and tastes can play an indirect role in the decision to purge the members of the second class of information, those clauses that are rewritten with demodulation and then purged with subsumption. The user's choice of demodulators provides the indirect means for purging. In contrast to the members of the first two classes of information, the purging of the members of the third class of information is solely at the direct discretion of the user of the automated reasoning program. In particular, by setting the weights (priorities [McCharen76]) appropriately, one can cause the automated reasoning program to purge information that one, on the basis of knowledge or intuition, deems undesirable.

A review of the preceding material shows why we strongly conjecture that to be effective for solving difficult problems, an automated reasoning program must retain intermediate clauses. The retention of information virtually requires—as our experiments strongly suggest—the use of both subsumption and demodulation. Indeed, even among those researchers who prefer a paradigm based on logic programming, there exist individuals who advocate clause retention. We find it natural to conjecture that, especially for those interested in a logic programming paradigm, the trend toward accepting the need for retaining intermediate information will grow, and that the recognition of the need for subsumption and demodulation will also soon follow—perhaps to the point of rediscovering a general-purpose automated reasoning program. We also predict that, among those individuals who accept our challenge of attempting to prove the theorems we discuss in section 4, many will come to the conclusion we espouse, a paradigm for the automation of reasoning that is based on logic programming in its purest form offers inadequate power for attacking deep questions.

Our preceding remark does not imply that we consider the effort devoted

to such a paradigm to be of little or no use. Quite the contrary. For simple theorems and for well-focused reasoning, such a paradigm is indeed attractive. In addition, some of the technology that has evolved because of the study of such paradigms is proving useful for paradigms not based on logic programming. For example, the use of pointers in place of substitution application materially speeds up demodulation [McCune88]. Also, we expect that clause compilation will prove extremely important to the implementation of linked inference rules (discussed in section 2.5; see also [Wos84a]).

Even without these examples, we applaud efforts devoted to diverse paradigms for the automation of reasoning. Diverse and unrelated or distantly related approaches must be studied if a field is to reach its full potential. However, good science requires that software designers inform potential users of the weaknesses of a program. For example, when software is suspected of lacking the power for attacking difficult problems or proving deep theorems, an explicit warning should be issued. Such a warning can save researchers substantial time and energy devoted to the development and extension of systems based on a paradigm that lacks the desired properties.

To summarize, although the paradigm (for the automation of reasoning) based on logic programming is intellectually seductive and attractively simpler than, for example, the paradigm on which the reasoning programs developed at Argonne are based, the logic-programming paradigm does not work well enough for attacking deep questions and hard problems.

2.4 Hyperresolution, Its Strengths and Weaknesses

To complete the aspect of our case asserting that subsumption has turned out to be even more important than Robinson's work on inference rules, we need to discuss the inference rule hyperresolution. This discussion also answers the fourth question posed earlier, that focusing on the strengths and weaknesses of hyperresolution.

Just as Robinson exhibited excellent insight by introducing binary resolution to move the domain of discourse from the emphasis on ground expressions to an emphasis on expressions in which universally quantified variables occur, he again exhibited powerful insight by introducing hyperresolution, which moves the emphasis from small deduction steps to larger ones. Hyperresolution, which can be viewed—and even implemented—as combining a sequence of binary resolution steps into a single step, was the first of a number of such inference rules. Robinson, therefore, deserves substantial credit for introducing the notion of inference rules that skip over less important intermediate conclusions. Other examples of such inference rules—influenced by Robinson's success—include Overbeek's UR-resolution [McCharen76] and Wos's linked inference rules [Wos84a].

The most obvious strength of hyperresolution is its use to enable an automated reasoning program to avoid deducing numerous intermediate and partial conclusions, conclusions of the type produced by using binary resolution. The

absence of such conclusions usually contributes—in most cases markedly—to the effectiveness of a reasoning program.

By requiring that all conclusions obtained with hyperresolution consist of positive literals only, this inference rule has the additional strength of offering certain strategic advantages. In particular, the search space is materially reduced—especially for those problems for which such positive information plays the key role. However, even when a significant reduction in the size of the search space does occur, subsumption still plays a vital role. Evidence of this fact was given in section 1 where we discussed the successful attempt to prove SAM's lemma; additional evidence will be given in section 4.

The emphasis on positive clauses—while offering the cited strategic advantage—also presents an important weakness of this inference rule. Specifically, keying on syntactic criteria is far less attractive and far less powerful than keying on semantic criteria.

For example, if one writes the clause

$$\text{-LT}(a, b) \mid \text{EVEN}(a)$$

with LT meaning less than and with the intention of having an automated reasoning program focus heavily on the corresponding fact, then hyperresolution will indeed consider this clause as a nucleus. On the other hand, if one writes

$$\text{GTE}(a, b) \mid \text{EVEN}(a)$$

with GTE meaning greater than or equal—obtaining an equivalent statement—then hyperresolution will not consider this clause as a nucleus. Little imagination is required to generalize this example to see that the user of a reasoning program can be forced into a not necessarily preferred representation of information.

A solution that appeals to us focuses on the replacement of syntactic criteria by semantic criteria (see research problems 5 and 10 of [Wos87]). Specifically, rather than keying on positive and negative literals as hyperresolution does, we conjecture that it would be far better to key on *if* literals and *then* literals, defined by the user for each problem. Then with the so-called semantic version of hyperresolution, the program would be required to remove from a nucleus all *if* literals and to introduce no new *if* literals. With semantic hyperresolution applied to the example cited earlier to illustrate the weakness of (syntactic) hyperresolution, one's actions would be forced in no way, for one could simply declare $\text{-LT}(a, b)$ or, equally, $\text{GTE}(a, b)$ to be an *if* literal.

Hyperresolution also suffers in many cases from the important strategic weakness of inadequate use of the denial of the conclusion to be proved. Specifically, there typically exist, among those clauses that are present because the desired result is assumed to be false in order to prepare the way for searching for a proof by contradiction, clauses containing no positive literals. Even further, many of those nonpositive clauses are unit clauses. Such clauses, by definition, cannot be used as satellites, and frequently such clauses cannot be used as nuclei, because the needed satellites are lacking. Therefore, the nonpositive clauses that are present in the presentation of a problem are often ignored by hyperresolution—ignored until the final step of the proof search. In other words,

from a global perspective, the use of hyperresolution as the sole rule of inference forces an automated reasoning program, in far too many cases, to prove a theorem under consideration by, so to speak, *finding* the conclusion of that theorem, simply completing the search for a proof by contradiction by merely noting that the conclusion that was found happens to contradict the input assumption that the conclusion is false. The search for a proof should not complete simply because the desired result was discovered more or less by accident—because a clause was finally deduced that happens to conflict with the denial. Indeed, far more effective than relying on such a find is to *key* the search for a proof by contradiction on the assumption that the theorem is false. Although hyperresolution—as well as, to a slightly lesser extent, paramodulation—suffers from an inadequate use of the denial of the conclusion of the theorem under consideration, Robinson deserves much credit for introducing inference rules that combine in one step a number of less significant conclusions.

Although not precisely a weakness of hyperresolution, we note that in many cases hyperresolution does not take large enough steps and, obviously, the size of the steps it takes is not directly a function of the user's instructions. To see what we mean by this remark, let us now turn to the fifth question.

2.5 Linked Inference Rules

In this section we answer the fifth question, a question that asks about some type of reasoning that gives the user of an automated reasoning program the opportunity of playing an active role in deciding what kinds of information to retain and also gives the user the option of determining the size of the deduction steps that are taken.

Without doubt, the idea for the inference rules (linked inference rules [Wos84a]) discussed in this section stemmed from a familiarity with hyperresolution and UR-resolution. Indeed, in imitation of Robinson's recognition of the importance of combining a number of small deduction steps in one large deduction step, the first linked inference rule (linked UR-resolution [Wos84a]) was formulated by Wos and introduced in 1982. The motivation for the formulation of linked inference rules rests to some extent with the interest in taking deduction steps larger than those offered by hyperresolution, to a greater extent with the need to circumvent the syntactic constraints present in various standard inference rules, and to the greatest extent with the desire to permit the user to define which information is significant enough to merit retention.

The idea is to relegate certain information to being implicit, information that would ordinarily be retained explicitly. Indeed, just as hyperresolution—if appropriately implemented [Overbeek75]—relegates to being implicitly present the so-called intermediate clauses that would otherwise arise by taking the corresponding binary resolution steps, linked inference rules equally avoid the generation of certain intermediate conclusions. In other words, linked inference rules share an important property with the reasoning found in a number of well-known

inference rules and also in logic programming. On the other hand, in contrast to logic programming, linked inference rules are designed to permit the user to instruct an automated reasoning program to retain information that the user deems significant.

From a different perspective, where binary resolution (too often) takes deduction steps that are too small, and where logic programming (too often) takes deduction steps that are too large, linked inference rules give the user control over their size. Taking very small deduction steps introduces—with a high probability—a great deal of unneeded information, which results in a marked degradation of program effectiveness. Although far less obvious, taking very large deduction steps has too high a potential of skipping over key information, again resulting in a sharp degradation of program effectiveness [Wos89].

Linked inference rules include generalizations of well-known inference rules such as hyperresolution and UR-resolution. The generalizations typically rest on a relaxation of the constraints found in the standard version. For example, a literal in a nucleus for UR-resolution can be removed—in the linked version—by a nonunit clause. In addition, the user can affect the size of the deduction step taken and is permitted to decide what criteria are to be used for determining the successful completion of a linked inference rule. The completion criteria typically take the form of some semantic requirement, such as “accept a conclusion only if it is a unit clause in the predicate FEMALE”.

To illustrate how linked inference rules work and how information is relegated to being implicit, let us consider the statements “Gail is female”, “everyone who is female is **not** male”, and “the job of nurse is held by a male”. If we apply linked UR-resolution to the corresponding clauses

- (1) FEMALE(Gail)
- (2) -FEMALE(x) | -MALE(x)
- (3) -HASAJOB(x,nurse) | MALE(x)

with the instruction that the clauses to be retained must consist of literals in the predicate HASAJOB, then the clause

- (4) -HASAJOB(Gail,nurse)

will be deduced. In contrast to the standard version of UR-resolution, which would permit an automated reasoning program to deduce

- (5) -MALE(Gail)

from clauses (1) and (2), linked UR-resolution treats clause (5) as being only implicitly present, using clause (2) to link clause (1) to clause (3). In particular, if a program considers clauses (1) and (3) as the only hypotheses, no conclusion results, but clause (2) provides the needed literals to link the two hypotheses together to permit the deduction of clause (4). One can construct similar examples that even more closely capture the spirit of logic programming, a topic addressed in detail in [Wos89].

Preliminary experiments indicate that the use of subsumption, even for linked inference rules that take large deduction steps, will prove very significant. Still further, subsumption may prove most useful *during* the application of a linked inference rule—even before its completion. Although early experiments suggested such a use would be rather expensive, a new approach offers the possibility of sharply reducing this cost. That new approach will be based in part on the use of McCune's discrimination trees and in part on an ordering strategy currently under formulation. We also expect to borrow from logic programming by employing clause compilation. Additional experiments will be required to determine when it is wise to invoke subsumption during an attempt to complete an application of a linked inference rule.

2.6 Subsumption as Robinson's Most Significant Contribution

We close section 2 with our answer to the sixth question, that focusing on why we consider subsumption to be—in the context of automated reasoning circa 1989—Robinson's most significant contribution to the field.

For any or all of the following four reasons, one might not share our view that subsumption is Robinson's most important contribution to automated reasoning as it is practiced today. First, one might not have examined appropriate experimental data. Second, one might consider Robinson's work on binary resolution or on hyperresolution more significant. Third, one might prefer to rely on a paradigm for which subsumption is irrelevant. Fourth and finally, one might be misled by the naturalness, or even simplicity, of the notion of subsumption. Let us discuss these four factors in reverse order.

From a general viewpoint, we consider that subsumption is to automated reasoning as the wheel is to transportation. For us, this analogue holds the keys to defusing the fourth cited factor. In particular, one might consider the wheel to be an obvious solution to overcoming the friction one would encounter without it, say when dragging one's burden over uneven terrain. Such a view might quickly mislead one into giving little credit to the inventor of the wheel. Of course, as it has turned out to this point, unless one is airborne or seaborne, no substitute for the wheel has yet been found, and the wheel works most effectively. Subsumption has the following analogous properties: its formulation is an obvious solution (to coping with unwanted copies and unwanted instances of existing statements), no substitute for subsumption has yet been found, and subsumption does the job nicely.

As for the third given factor—although our analogue might seem rather sharp—the preference for reliance on a paradigm for which subsumption is irrelevant is like having a preference for avoiding gravity. Indeed, although places exist in which gravity is negligible and—as in the case of a paradigm based on logic programming—there exist many problems for which clause retention is unnecessary and, therefore, subsumption is irrelevant, for planetary living and for

attacking deep questions and hard problems, gravity and, respectively, clause retention and the use of subsumption must be accepted.

With regard to the second factor—that concerning the relative significance of Robinson's other contributions—here we encounter an area for possible debate. After all, moving the domain of discourse from the ground level to the variable level, as evidenced in binary resolution, marked an important advance. Then the move from deduction steps in the spirit of syllogism and modus ponens to deduction steps that combine a number of such steps—as evidenced in hyperresolution—was itself an excellent move, and it set the trend for much work that followed in the area of inference rule. Indeed, one can make a case for the value—historic and other—of the corresponding research. Nevertheless, since small deduction steps introduce too many insignificant conclusions, since semantics should be emphasized far more than syntax, since there now exist inference rules more powerful than either binary resolution or hyperresolution, and since no substitute has yet been found for clause retention coupled with the use of subsumption and demodulation, we find the case clear for evaluating subsumption to be the most significant of Robinson's contributions to automated reasoning circa 1989.

Finally, for the factor focusing on experimentation, rather than citing results at this point, we instead devote all of section 4 to such results. There we also issue specific challenges for researchers who are dubious about the need for subsumption, the need for demodulation, and the need for clause retention. However, before we present experimental results and issue our challenges, to complete the setting and to further strengthen the case for clause retention and hence for subsumption, we turn in section 3 to pertinent history and a perspective concerning problem difficulty.

3 History and Problem Difficulty

To put into perspective many of the observations made in earlier sections, let us examine the nature of various experiments and the attitude toward those experiments from the virtual inception of the field until now. Near the beginning in 1963, when the field was called mechanical theorem proving, the ability to solve a test problem called the Davis-Putnam example received some acclaim. A glance at the example by a researcher who has entered the field after 1970 will likely produce substantial puzzlement at finding that any satisfaction was derived from coping with the example. After all, one can correctly classify the Davis-Putnam example as trivial and uninteresting, at least by today's standards.

This correct classification does not imply that the early researchers were easily impressed; instead, the more accurate observation asserts that early in the history of a field, researchers typically have few achievements, and those that they have are usually unimpressive after but a few years. Even more important,

binary resolution had not yet been formulated when the Davis-Putnam example was first studied. From a global perspective, this example provides an excellent illustration of how a few years of research can dramatically affect a field. In the case of automated reasoning, the introduction of binary resolution enabled a program to cope immediately and easily with an example that had required, at one time, approximately thirty minutes of hand computation and had also been out of reach of an existing reasoning program employing an entirely unrelated approach.

Here are the corresponding clauses (with a change of notation from the original) for the Davis-Putnam example. If an attempt to cope with this example meets with the slightest difficulty, then one has discovered an overwhelming weakness in the corresponding reasoning program.

$$\begin{aligned} &.(1 P(x,y) \\ &-P(y, f(x, y)) \mid -P(f(x, y), f(x, y)) \mid Q(x, y) \\ &-P(y, f(x, y)) \mid -P(f(x, y), f(x, y)) \mid -Q(x, f(x, y)) \mid -Q(f(x, y), f(x, y)) \end{aligned}$$

Beginning in 1963 and during the next two decades of automated theorem proving, most of the problems for experimentation were taken from algebra. Those most frequently offered as challenge problems were from group theory, ring theory, and Henkin models [McCharen76]. Indeed, one of the first interesting theorems that was proved asserts that groups in which the square of every element is the identity are Abelian. (For those who enjoy history, in chapter 2 of [Wos87] one can read about the interesting connection this theorem has to the discovery of the set of support strategy.) Even though this theorem has always been correctly classified as a simple classroom exercise, the first proof of this theorem obtained with a computer program nevertheless produced substantial excitement. After all, its proof had actually appeared in a journal on mathematics, and proving a theorem that had appeared in the literature did at that time mark a singular advance. Far more interesting and deserving of exuberance was the successful proof that in groups in which the cube of every element is the identity, certain commutators are equal to the identity. We suggest in section 4 that this theorem provides an interesting challenge problem, one that appears to require the use of both subsumption and demodulation to obtain a proof.

Especially in the first decade of experimentation, the attacks on the problems frequently studied shared the following properties.

- The deductions leading to a proof were in most cases “unfocused”. No algorithm existed—nor does one yet exist—for determining a priori precisely which formulas hold the key to the proof, and programs progressed by methodically exploring what could be derived from (in general) progressively more complex formulas.
- Goal reduction mechanisms, such as SL resolution or model elimination, did not work well. Although these inference mechanisms were later to

be used with stunning success in areas in which the deductions could legitimately be highly structured, they have never been successfully used to attain a proof of any of the more difficult challenge problems from algebra.

- The essential key to success in attacking the various challenge problems was (and still appears to be) to contain the combinatoric explosion usually encountered when seeking the corresponding proof—a feat that many researchers continue to regard as fundamentally hopeless. In the order of discovery, the four basic mechanisms found to be most effective in controlling the potential explosion were subsumption, the set of support strategy, demodulation, and weighting. Certainly the choice of inference rule proved to be important. However, the actions of purging redundant information (with subsumption), increasing the likelihood of drawing useful conclusions (with the set of support strategy), simplifying those conclusions (with demodulation), and restricting the set of terms being considered (with weighting) all seemed—and still seem—essential to success.

Not long after the discovery of logic programming, a sharp reversal in emphasis occurred. The selection of problems was restricted, for many researchers, to those solvable with “algorithmic deduction”. In this context, exceedingly long chains of inference, made at previously unheard of speeds, produced impressive results. One key to such high performance was the elimination of the overhead of deriving new clauses and, more important, the overhead of attempting to simplify clauses with demodulation and the overhead of attempting to discard clauses with subsumption. Goal solving offered exactly the right paradigm for reasoning in the context of solving problems that admit of an algorithmic attack.

Today a number of researchers, including members of the automated reasoning group at Argonne, are investigating the integration of the two highly disparate styles of deduction, that focusing on a paradigm in which clause retention is deemed essential and that focusing on a paradigm based on logic programming. In particular, we are studying the adaptation of various techniques resident in logic programming to enhance our current approach. To successfully merge the two disparate paradigms, we conjecture that one must keep in mind the characteristics of problems for which algorithmic reasoning does not suffice versus those for which it does. For example, a logic programmer might be most amused if someone began with the axioms

```
append([],L,L)
append([H|T],L,[H|T2]) :- append(T,L,T2)
```

and the goal

```
:- append([a,b,c],[d,e,f],L)
```

and, instead of relying on logic programming, proceeded to derive a proof by contradiction using, say, UR-resolution in a system relying on the use of sub-

sumption, demodulation, and weighting. To use these complex mechanisms for such a simple task would correctly be considered a serious error. Conversely, if one relies mainly on logic programming when searching for proofs of theorems of even moderate difficulty, then one is committing, in our view, an equally serious error.

If such is the case, then how do we account for the current obvious eagerness to use logic programming as the basis for the design of programs whose object is to prove theorems? The explanation rests in part with the understandable wish to find a shortcut to producing a program with the appropriate degree and type of power but, more important, rests also with the confusion concerning how to measure program performance (see [Wos89] for a discussion of both factors). In particular, logic programming systems are frequently rated in terms of LIPS (logical inferences per second), a measure that is almost always based on a single benchmark—reversing a list (the “naive reverse” benchmark). Unfortunately, from the viewpoint of a mathematician or logician, the corresponding operations do not measure logical inferences—such a term is clearly misleading. Indeed, anyone marketing a system that could do no useful computation more difficult than reversing a list of 1,000 elements would probably not gain a major share of the market.

Similarly, marketing a theorem-proving program that could prove simple theorems but no more would not inspire much excitement, except for those applications for which proving simple theorems is one of the main activities. Program verification is one such important application, as both Boyer and Moore have repeatedly said. Unfortunately, some of the earlier mentioned confusion has resulted from overestimating the significance of success with this type of theorem. For example, success in proving the simple theorem called (by other researchers) Wos10 has caused and continues to cause researchers confusion regarding the power of the program used to obtain a proof of this theorem. The theorem, which asks one to prove commutativity for groups in which the square of every element x is the identity e , was widely used to measure program performance. In addition, with all programs designed and implemented at Argonne, as well as with programs designed at many other centers throughout the world, this theorem was used to debug and evaluate systems early in their development. This simple theorem, although providing an excellent example for illustration, does not provide much of a test of the value of a new idea or of a new program. Therefore, the glee that sometimes accompanies an announcement of a successful attempt to obtain a proof of this theorem is not warranted.

As evidence of the difficulty presented by Wos10, even with a complete axiomatization of groups—seventeen clauses in a notation that does not emphasize the role of equality (see chapter 6 of [Wos87])—the theorem was proved in less than 5 CPU seconds in the mid 1960s on an IBM 704 *without* the use of demodulation. With currently available programs, running on a Sun 3 workstation, a proof is obtained in less than 2 CPU seconds. This theorem is but the first of a set of ever more difficult theorems that we discuss in section 4, where we present

experimental results and offer challenges. To obtain a far more accurate measure of the value of a new idea or new program, we offer the following three theorems of increasing difficulty, theorems that have also been used to test and evaluate automated reasoning programs; indeed, the first and second are included among the five theorems on which we focus in section 4.

1. In a ring, if (for all x) the square of x is x , then the ring is commutative.
2. In a group, if (for all x) the cube of x is the identity e , then the equation $[[x, y], y] = e$ holds, where $[x, y]$ is the product of x, y , the inverse of x , and the inverse of y .
3. In a ring, if (for all x) the cube of x is x , then the ring is commutative.

In automated reasoning, a still frequent but unfortunate practice is to evaluate the effectiveness and power of a program by its performance on Wos10 and with complete disregard for whether the program can be used to solve more difficult problems. This practice is one of the forces that causes us to issue the challenges presented in section 4. Since the potential for combinatoric explosion increases sharply as the difficulty of the problem under consideration increases, especially new researchers in the field of automated reasoning should be warned that Wos10 is a trivial problem. For evaluating the power of a new technique or a reasoning program, the far more accurate test is how well the technique or program copes with the combinatoric explosion that is encountered with more difficult problems (such as the three just listed), and not with how much faster the technique or program solves a trivial problem.

4 Experimental Results and Challenges

What still impresses us—"amazes" is probably a more accurate word—is the lack of universal acceptance and endorsement of experimentation as the only true test of the power and value of paradigms and programs. Certainly in the first decade of the field (1963–1973), automated reasoning—although it was not called that—suffered severely from inadequate experimentation. Today the inadequacy often takes a subtler form: too often, experiments are *not* concerned with moderate to difficult theorems and problems. If such are avoided, then how can the field discover—and eventually overcome—the obstacles to having access to a powerful automated reasoning colleague (see chapter 2 of [Wos87] for a discussion of those obstacles)?

Almost from the beginning there existed reasoning programs designed and used for experimentation. Typical of progress in science, the results of those experiments have been analyzed, and a fair amount of "folk wisdom" has emerged; in some cases, theoretical results from complexity analysis have reinforced central observations. Within that folk wisdom one finds several widely accepted tenets that are misleading, misleading at least in the sense that they hold far

less often than commonly accepted. Among those tenets, the following three—the first two of which we also espoused at various times—merit consideration.

- A. The growth of the space of clauses to be searched is inescapably exponential.
- B. If one cannot obtain a proof in the first few minutes, then success will never occur.
- C. Insights gained from experimenting with simple theorems normally apply to a study of more complex theorems.

We suspect that the folk wisdom embodied in these three assertions is fundamentally flawed. To substantiate our suspicion, we present a graduated set of theorems for experimentation, theorems we proved by using hyperresolution as the chosen inference rule. What makes the experiments we discuss of particular relevance to the primary issue in this article is the evidence they provide of the importance of subsumption to automated reasoning. The following five theorems form the graduated set on which we experimented and which we also propose as challenge problems.

1. In a group, if (for all elements x) the square of x is the identity e , then the group is commutative.
2. In a group, if (for all x) the cube of x is the identity e , then the equation $[[x, y], y] = e$ holds, where $[x, y]$ is the product of x, y , the inverse of x , and the inverse of y .
3. In a ring, if (for all x) the square of x is x , then the ring is commutative.
4. The formula $XGK = e(x, e(e(y, e(z, x)), e(z, y)))$ is a shortest single axiom for the equivalential calculus.
5. The formula $i(i(i(x, y), z), i(i(z, x), i(u, x)))$ is a shortest single axiom for the implicational calculus.

Proofs of each of these five theorems have been obtained by one or more existing automated reasoning programs; in the appendix we include a proof of each of the five theorems.

To encourage others to try the same experiments, let us examine certain statistical aspects of the five proofs (table 1). A study of these statistics supports our suspicion concerning tenets *A*, *B*, and *C*. (In the appendix to this article we include the precise clauses that we submitted to OTTER [McCune90] to obtain the proofs.)

Table 1
 Statistical aspects of the proofs of theorems 1 to 5

Theorem	Length of proof	No. kept clauses	No. subsumed clauses	time (sec.) on Sun 3)
1	4	6	136	1.16
2	23	826	20,402	242
3	31	408	50,742	637
4	51	28,392	482,119	29,486
5	89	20,341	4,319,586	41,130

The statistics we cite show that one can obtain proofs that are far longer than one would think possible if the growth of the search space were inescapably exponential. In particular, in the last case the 89 hyperresolvents correspond to 177 binary resolvents, and the proof completes at level 39—the proof tree is 39 levels deep. Clearly these results contradict the publicly expressed intuitions (concerning tenet A) of many researchers in the field, including ourselves at an earlier time.

From a different perspective—that of computation time—we note that to prove the fourth theorem requires approximately eight CPU hours on a Sun 3 workstation, while to prove the fifth requires over 11 CPU hours. As is obvious—but often not admitted—proofs can be obtained after an exceedingly impressive amount of computation time has been expended, which supports our rejection of tenet B. For the record, we conjecture that proofs of twice the length and complexity of the fifth theorem will be obtained within the next five years. We also conjecture that, at first, those longer proofs will require orders of magnitude more computation time than was required to obtain a proof of the fifth theorem.

With regard to our position concerning the importance of using subsumption, let us now examine more closely the results of our experiments. Our objective, of course, is to show that subsumption plays a vital role in obtaining these proofs. Let us consider some data pertinent to the first theorem in our sequence of five (table 2).

The first and obvious conclusion one can draw from this data is that subsumption plays a significant role. Nevertheless, we must point out that subsumption is not the whole answer to conducting a successful attack for obtaining proofs of theorems of varying difficulty. Indeed, in the results presented earlier, we noted that six clauses were deduced and kept in the proof generated by OTTER.

Table 2
Data from the first two levels of theorem 1

Type of clause	No.
Clauses generated during level 1:	206
Clauses deleted by an identity check:	125
Clauses that could be deleted by a complete subsumption check:	138
Clauses deleted at level 2 (using the $206 - 138 = 68$ clauses that survived a subsumption check at level 1):	37,560
Clauses kept at level 2 (those that passed a subsumption test):	22,318

One might well ask how such a result is possible, given that $68 + 22,318$ new clauses can be generated and pass a subsumption test by level 2. The answer, in this case, lies in the use of demodulation and weighting. Actually, for difficult problems, we have found that a number of techniques must all be employed simultaneously for proofs to be obtained. In particular, for most difficult problems, subsumption, demodulation, the set of support strategy, and weighting are all required. Our challenge to researchers is the following: find a system that can prove any of theorems 2 through 5, *without* using subsumption.

We doubt that such a system currently exists. If we are correct, we will have shown that the use of subsumption is indeed indispensable and also that the introduction of subsumption to automated reasoning is Robinson's most significant contribution.

5 Conclusions

When Robinson turned his attention to the problem of proving theorems with a computer program, he clearly recognized that what was needed was a means to reason, and reason logically. However, obviously an important distinction exists between what is needed and how to fulfill that need. To reason logically but with the intention of offering adequate power, Robinson introduced the full version of binary resolution. He noted that one means to obtain the desired power was to avoid deducing unneeded conclusions, conclusions of the type yielded by relying on instantiation coupled with syllogistic reasoning. But, as it turns out and as discussed earlier, in most cases binary resolution yields too many small conclusions when compared to what is needed and what is possible. Indeed, compared to binary resolution, Robinson's hyperresolution does a better job of avoiding the deduction of unneeded conclusions. Unfortunately, hyperresolution relies on syntactic rather than semantic criteria, and for many cases even this inference rule does not take large enough deduction steps.

One might then conclude that a paradigm that totally avoids the retention

of new information is clearly the best approach of all, for it is the limiting case. However, as our experiments strongly suggest, such is not the case—at least if one is attacking deep questions and hard problems. Rather, the retention of information coupled with the use of subsumption, demodulation, and various types of strategy provide the basis for a powerful and general-purpose paradigm for the automation of reasoning. Subsumption in particular, as our experiments show, is vital for attacking deep questions and hard problems with the assistance of an automated reasoning program. Therefore, although we compliment J. A. Robinson for his excellent research focusing on inference rules, in our view, he deserves even more acclaim for his contribution of subsumption.

Indeed, because information retention is crucial for certain types of investigation, because redundant information must be purged, and because no substitute for subsumption currently exists—in contrast to the fact that inference rules exist that offer more power than either binary resolution or hyperresolution does—we consider Robinson's contribution of subsumption to be his most important, at least for automated reasoning circa 1989.

Appendix: Clauses Submitted to and Proofs Found by OTTER for Theorems 1 through 5

The OTTER theorem prover, as a research vehicle, has a large number of user-settable options in order to provide the user with extensive control over its operations. At the same time, proofs of difficult theorems can be obtained without fine-tuning OTTER to individual problems. As an illustration, the proofs of all five theorems here were obtained with the same set of simple options, specified to OTTER in the following way:

```
set(hyper_res). % use hyperresolution only
set(back_demod). % enable back demodulation
set(atom_wt_max_args). % compute weight by symbol count,
                        % using the maximum instead of sum
                        % of argument weights
assign(max_weight,N). % apply a weight cutoff for
                       % keeping new clauses
```

The `max_weight` parameter N varies from problem to problem. In each case the statistics given in the paper are taken from the run with the lowest value of N that allowed the proof to be completed. The appropriate N was determined through an iterative deepening process. (For the `not` symbol, in place of “-” OTTER requires the use of “.”.)

THEOREM 1:

```

% P(x,y,z) means that the product of x and y is z.

assign(max_weight,2).
list(axioms).

% e is a two-sided identity.
P(e,x,x).
P(x,e,x).

% g(x) is a two-sided inverse for x.
P(g(x),x,e).
P(x,g(x),e).

% closure
P(x,y,f(x,y)).

% associativity
-P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
-P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).

% Product is well defined.
-P(x,y,u) | -P(x,y,v) | EQUAL(u,v).

% equality axioms
EQUAL(x,x).
-EQUAL(x,y) | EQUAL(y,x).
-EQUAL(x,y) | -EQUAL(y,z) | EQUAL(x,z).

% equality substitution axioms
-EQUAL(u,v) | -P(u,x,y) | P(v,x,y).
-EQUAL(u,v) | -P(x,u,y) | P(x,v,y).
-EQUAL(u,v) | -P(x,y,u) | P(x,y,v).
-EQUAL(u,v) | EQUAL(f(u,x),f(v,x)).
-EQUAL(u,v) | EQUAL(f(x,u),f(x,v)).
-EQUAL(u,v) | EQUAL(g(u),g(v)).

end_of_list.

list(sos).

% The square of every x is the identity.
P(x,x,e).

% Denial: there exist two elements that do not commute.

```

```

P(a,b,c).
-P(b,a,c).

end_of_list.

```

Proof of theorem 1

```

1 [] P(e,x,x).
2 [] P(x,e,x).
6 [] -P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
7 [] -P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).
18 [] P(x,x,e).
19 [] P(a,b,c).
20 [] -P(b,a,c).
21 [hyper,19,7,18,2] P(c,b,a).
22 [hyper,19,6,18,1] P(a,c,b).
23 [hyper,21,6,18,1] P(c,a,b).
25 [hyper,23,7,22,19] P(b,a,c).
26 [binary,25,20] .

```

THEOREM 2:

```

assign(max_weight,4).
list(axioms).

% e is a two-sided identity.
P(e,x,x).
P(x,e,x).

% g(x) is a two-sided inverse for x.
P(g(x),x,e).
P(x,g(x),e).

% closure
P(x,y,f(x,y)).

% associativity
-P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
-P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).

% Product is well defined.
-P(x,y,u) | -P(x,y,v) | EQUAL(u,v).

```

```

% equality axioms
EQUAL(x,x).
-EQUAL(x,y) | EQUAL(y,x).
-EQUAL(x,y) | -EQUAL(y,z) | EQUAL(x,z).

% equality substitution axioms
-EQUAL(u,v) | -P(u,x,y) | P(v,x,y).
-EQUAL(u,v) | -P(x,u,y) | P(x,v,y).
-EQUAL(u,v) | -P(x,y,u) | P(x,y,v).
-EQUAL(u,v) | EQUAL(f(u,x),f(v,x)).
-EQUAL(u,v) | EQUAL(f(x,u),f(x,v)).
-EQUAL(u,v) | EQUAL(g(u),g(v)).

end_of_list.

list(sos).

% x cubed is e
-P(x,x,y) | P(x,y,e).
-P(x,x,y) | P(y,x,e).

% Denial: for some a and b, [[a,b],b] is not e.
P(a,b,c).
P(c,g(a),d).
P(d,g(b),h).
P(h,b,j).
P(j,g(h),k).
-P(k,g(b),e).

end_of_list.

```

Proof of theorem 2

```

1 [] P(e,x,x).
2 [] P(x,e,x).
3 [] P(g(x),x,e).
5 [] P(x,y,f(x,y)).
6 [] -P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
7 [] -P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).
8 [] -P(x,y,u) | -P(x,y,v) | EQUAL(u,v).
18 [] -P(x,x,y) | P(x,y,e).
19 [] -P(x,x,y) | P(y,x,e).
20 [] P(a,b,c).

```

```

21 [] P(c,g(a),d).
22 [] P(d,g(b),h).
23 [] P(h,b,j).
24 [] P(j,g(h),k).
25 [] -P(k,g(b),e).
29 [hyper,20,7,5,5] P(f(x,a),b,f(x,c)).
37 [hyper,20,6,5,5] P(a,f(b,x),f(c,x)).
61 [hyper,21,7,3,2] P(d,a,c).
64 [hyper,21,6,2,5] P(c,g(a),f(d,e)).
76 [hyper,61,6,5,5] P(d,f(a,x),f(c,x)).
85 [hyper,22,7,3,2] P(h,b,d).
90,89 [hyper,85,8,23] EQUAL(j,d).
98 [hyper,85,6,5,61] P(h,f(b,a),c).
103 [back_demod,24,demod,90] P(d,g(h),k).
154 [hyper,103,7,3,2] P(k,h,d).
220 [hyper,18,5] P(x,f(x,x),e).
221 [hyper,19,5] P(f(x,x),x,e).
259,258 [hyper,64,8,21] EQUAL(f(d,e),d).
296 [hyper,98,7,154,5] P(d,f(b,a),f(k,c)).
434 [hyper,220,7,29,2] P(f(x,c),f(b,b),f(x,a)).
437 [hyper,220,7,5,2] P(f(x,y),f(y,y),x).
438 [hyper,220,7,3,2] P(e,f(x,x),g(x)).
459 [hyper,221,7,76,5,demod,259] P(f(c,a),a,d).
679,678 [hyper,438,8,1] EQUAL(g(x),f(x,x)).
704 [back_demod,25,demod,679] -P(k,f(b,b),e).
711 [hyper,459,6,37,296] P(f(c,a),f(c,a),f(k,c)).
791 [hyper,711,19] P(f(k,c),f(c,a),e).
880 [hyper,791,7,437,434] P(k,f(b,b),e).
881 [binary,880,704] .

```

THEOREM 3:

```

assign(max_weight,4).
list(axioms).

% 0 is an additive identity.
S(0,x,x).
S(x,0,x).

% g(x) is an additive inverse for x.
S(g(x),x,0).
S(x,g(x),0).

% closure of addition

```

$S(x,y,j(x,y))$.

% associativity of addition

$-S(x,y,u) \mid -S(y,z,v) \mid -S(u,z,w) \mid S(x,v,w)$.

$-S(x,y,u) \mid -S(y,z,v) \mid -S(x,v,w) \mid S(u,z,w)$.

% Addition is well defined.

$-S(x,y,u) \mid -S(x,y,v) \mid \text{EQUAL}(u,v)$.

% equality axioms

$\text{EQUAL}(x,x)$.

$-\text{EQUAL}(x,y) \mid \text{EQUAL}(y,x)$.

$-\text{EQUAL}(x,y) \mid -\text{EQUAL}(y,z) \mid \text{EQUAL}(x,z)$.

% equality substitution axioms

$-\text{EQUAL}(u,v) \mid -S(u,x,y) \mid S(v,x,y)$.

$-\text{EQUAL}(u,v) \mid -S(x,u,y) \mid S(x,v,y)$.

$-\text{EQUAL}(u,v) \mid -S(x,y,u) \mid S(x,y,v)$.

$-\text{EQUAL}(u,v) \mid \text{EQUAL}(j(u,x),j(v,x))$.

$-\text{EQUAL}(u,v) \mid \text{EQUAL}(j(x,u),j(x,v))$.

$-\text{EQUAL}(u,v) \mid \text{EQUAL}(g(u),g(v))$.

% commutativity of addition

$-S(x,y,z) \mid S(y,x,z)$.

% multiplication by 0

$P(0,x,0)$.

$P(x,0,0)$.

% closure for multiplication

$P(x,y,f(x,y))$.

% associativity of multiplication

$-P(x,y,u) \mid -P(y,z,v) \mid -P(u,z,w) \mid P(x,v,w)$.

$-P(x,y,u) \mid -P(y,z,v) \mid -P(x,v,w) \mid P(u,z,w)$.

% distributive laws

$-P(x,y,v1) \mid -P(x,z,v2) \mid -S(y,z,v3) \mid -P(x,v3,v4) \mid S(v1,v2,v4)$.

$-P(x,y,v1) \mid -P(x,z,v2) \mid -S(y,z,v3) \mid -S(v1,v2,v4) \mid P(x,v3,v4)$.

$-P(y,x,v1) \mid -P(z,x,v2) \mid -S(y,z,v3) \mid -P(v3,x,v4) \mid S(v1,v2,v4)$.

$-P(y,x,v1) \mid -P(z,x,v2) \mid -S(y,z,v3) \mid -S(v1,v2,v4) \mid P(v3,x,v4)$.

% Multiplication is well defined.

$-P(x,y,u) \mid -P(x,y,v) \mid \text{EQUAL}(u,v)$.

```

% equality substitution axioms
-EQUAL(u,v) | -P(u,x,y) | P(v,x,y).
-EQUAL(u,v) | -P(x,u,y) | P(x,v,y).
-EQUAL(u,v) | -P(x,y,u) | P(x,y,v).
-EQUAL(u,v) | EQUAL(f(u,x),f(v,x)).
-EQUAL(u,v) | EQUAL(f(x,u),f(x,v)).

end_of_list.

list(sos).

% The square of every x is x.
P(x,x,x).

% Denial: there exist elements that do not commute.
P(a,b,c).
-P(b,a,c).

end_of_list.

```

Proof of theorem 9

```

1 [] S(0,x,x).
2 [] S(x,0,x).
3 [] S(g(x),x,0).
5 [] S(x,y,j(x,y)).
6 [] -S(x,y,u) | -S(y,z,v) | -S(u,z,w) | S(x,v,w).
7 [] -S(x,y,u) | -S(y,z,v) | -S(x,v,w) | S(u,z,w).
19 [] P(0,x,0).
21 [] P(x,y,f(x,y)).
22 [] -P(x,y,u) | -P(y,z,v) | -P(u,z,w) | P(x,v,w).
23 [] -P(x,y,u) | -P(y,z,v) | -P(x,v,w) | P(u,z,w).
24 [] -P(x,y,v1) | -P(x,z,v2) | -S(y,z,v3) | -P(x,v3,v4)
    | S(v1,v2,v4).
25 [] -P(x,y,v1) | -P(x,z,v2) | -S(y,z,v3) | -S(v1,v2,v4)
    | P(x,v3,v4).
27 [] -P(y,x,v1) | -P(z,x,v2) | -S(y,z,v3) | -S(v1,v2,v4)
    | P(v3,x,v4).
28 [] -P(x,y,u) | -P(x,y,v) | EQUAL(u,v).
31 [] -EQUAL(u,v) | -P(x,y,u) | P(x,y,v).
34 [] P(x,x,x).
35 [] P(a,b,c).

```

```

36 [] -P(b,a,c).
39 [hyper,34,27,34,5,5] P(j(x,x),x,j(x,x)).
44 [hyper,34,27,19,5,5] P(j(0,x),x,j(0,x)).
45 [hyper,34,27,19,5,1] P(j(0,x),x,x).
50 [hyper,34,27,19,5,5] P(j(x,0),x,j(x,0)).
51 [hyper,34,27,19,5,2] P(j(x,0),x,x).
82 [hyper,35,27,34,5,5] P(j(b,a),b,j(b,c)).
96 [hyper,35,25,34,5,5] P(a,j(b,a),j(c,a)).
101 [hyper,35,23,35,34] P(c,b,c).
104 [hyper,35,23,21,21] P(f(x,a),b,f(x,c)).
108 [hyper,35,22,34,35] P(a,c,c).
152 [hyper,108,25,35,5,5] P(a,j(b,c),j(c,c)).
161 [hyper,108,23,21,21] P(f(x,a),c,f(x,c)).
173 [hyper,39,24,39,5,34] S(j(x,x),j(x,x),j(x,x)).
185,184 [hyper,45,28,44] EQUAL(j(0,x),x).
190,189 [hyper,51,28,50] EQUAL(j(x,0),x).
204 [hyper,82,22,34,82] P(j(b,a),j(b,c),j(b,c)).
209 [hyper,96,23,96,34] P(j(c,a),j(b,a),j(c,a)).
213 [hyper,104,23,21,34] P(f(b,c),a,f(b,a)).
240 [hyper,173,7,3,3] S(0,j(x,x),0).
267 [hyper,240,27,19,152,5,demod,185] P(a,j(b,c),0).
303 [hyper,240,7,5,5,demod,185] S(x,x,0).
312 [hyper,303,7,5,5,demod,190] S(j(x,y),y,x).
317 [hyper,303,6,5,5,demod,185] S(x,j(x,y),y).
325 [hyper,312,27,209,96,303] P(c,j(b,a),0).
326 [hyper,312,27,204,267,5,demod,190] P(b,j(b,c),j(b,c)).
342 [hyper,325,25,101,317,5,demod,190] P(c,a,c).
353 [hyper,342,22,161,213] P(f(b,a),c,f(b,a)).
359 [hyper,326,25,34,317,317] P(b,c,c).
364,363 [hyper,359,28,21] EQUAL(f(b,c),c).
404 [hyper,353,28,161,demod,364] EQUAL(f(b,a),c).
459 [hyper,404,31,21] P(b,a,c).
460 [binary,459,36] .

```

THEOREM 4:

```

assign(max_weight,20).
list(axioms).

% condensed detachment
-P(x) | -P(e(x,y)) | P(y).

end_of_list.

```

```

list(sos).

% the formula XGK
P(e(x,e(e(y,e(z,x)),e(z,y))))).

% the negation of PYO, which is a known single axiom
-P(e(e(e(a,e(b,c)),c),e(b,a))).

end_of_list.

```

Proof of theorem 4

```

1 [] -P(x) | -P(e(x,y)) | P(y).
2 [] P(e(x,e(e(y,e(z,x)),e(z,y))))).
3 [] -P(e(e(e(a,e(b,c)),c),e(b,a))).
4 [hyper,2,1,2] P(e(e(x,e(y,e(z,e(e(u,e(v,z))),e(v,u))))),e(y,x))).
5 [hyper,4,1,2] P(e(e(e(e(x,e(y,z)),e(y,x)),e(z,u)),u)).
6 [hyper,5,1,5] P(e(x,x)).
9 [hyper,6,1,2] P(e(e(x,e(y,e(z,z))),e(y,x))).
12 [hyper,9,1,5] P(e(x,e(e(e(y,e(z,u)),e(z,y)),e(u,e(x,e(v,v)))))).
14 [hyper,9,1,2] P(e(e(x,e(x,y)),y)).
15 [hyper,9,1,5] P(e(e(x,x),e(y,y))).
17 [hyper,9,1,2] P(e(e(x,e(y,e(e(z,e(u,e(v,v))),e(u,z))),e(y,x))).
23 [hyper,14,1,9] P(e(x,e(y,e(y,e(x,e(z,z)))))).
25 [hyper,14,1,2] P(e(e(x,e(y,e(e(z,e(z,u)),u)),e(y,x))).
27 [hyper,15,1,2] P(e(e(x,e(y,e(e(z,z),e(u,u))),e(y,x))).
61 [hyper,23,1,4] P(e(x,e(y,e(y,x)))).
81 [hyper,61,1,2] P(e(e(x,e(y,e(z,e(u,e(u,z))))),e(y,x))).
344 [hyper,25,1,14] P(e(x,e(y,e(y,e(x,e(e(z,e(z,u)),u)))))).
607 [hyper,81,1,2] P(e(e(e(x,e(x,y)),e(y,z)),z)).
940 [hyper,344,1,4] P(e(x,e(y,e(e(z,e(z,y)),x))).
978 [hyper,940,1,607] P(e(e(x,e(x,y)),e(z,e(z,y)))).
992 [hyper,940,1,5] P(e(e(x,e(x,y)),e(e(z,e(u,y)),e(u,z)))).
993 [hyper,940,1,4] P(e(x,e(e(y,e(z,e(u,e(u,x)))),e(z,y))).
1008 [hyper,978,1,4] P(e(x,e(y,e(y,e(e(z,e(u,x)),e(u,z)))))).
1593 [hyper,992,1,25] P(e(e(x,e(e(y,e(y,x)),z)),e(u,e(u,z)))).
1597 [hyper,993,1,81] P(e(e(e(x,e(x,y)),e(y,e(z,e(z,u))),u)).
4308 [hyper,17,1,1008] P(e(e(e(x,y),e(e(y,e(x,z)),e(u,u))),z)).
4312 [hyper,17,1,978] P(e(e(x,e(y,e(z,z))),e(u,e(u,e(y,x)))).
4326 [hyper,17,1,2] P(e(e(e(x,y),e(e(y,e(x,e(z,z))),u)),u)).
4706 [hyper,4312,1,25] P(e(e(x,e(x,e(y,z))),e(z,e(y,e(u,u)))).
4711 [hyper,4312,1,4] P(e(x,e(e(y,z),e(e(z,e(y,x)),e(u,u)))).
5248 [hyper,4326,1,1593] P(e(e(e(e(x,e(x,y)),z),e(y,e(u,u))),z)).

```

5255 [hyper,4326,1,12] $P(e(x, e(e(y, e(e(z, z), e(y, x))), e(u, u))))$.
 5354 [hyper,4711,1,4326] $P(e(e(e(x, e(y, y)), e(z, e(x, z))), e(u, u))$.
 5357 [hyper,4711,1,1597] $P(e(x, e(y, e(z, e(z, e(y, x))))))$.
 5627 [hyper,5248,1,4308] $P(e(e(e(x, y), e(x, e(y, e(z, z))))), e(u, u))$.
 5730 [hyper,5255,1,27] $P(e(e(x, e(e(y, y), e(x, z))), z)$.
 5897 [hyper,5354,1,4706] $P(e(e(x, e(y, x)), e(e(y, e(z, z)), e(u, u))))$.
 6338 [hyper,5627,1,5730] $P(e(e(x, y), e(x, e(y, e(z, z)))))$.
 6373 [hyper,6338,1,6338] $P(e(e(x, y), e(e(x, e(y, e(z, z))), e(u, u))))$.
 6945 [hyper,6373,1,27] $P(e(e(x, e(y, e(z, z))), e(x, y))$.
 6951 [hyper,6945,1,5897] $P(e(e(x, e(y, x)), e(y, e(z, z))))$.
 7300 [hyper,6951,1,6945] $P(e(e(x, e(y, x)), y))$.
 7445 [hyper,7300,1,5357] $P(e(x, e(y, e(y, e(x, e(e(z, e(u, z)), u))))))$.
 7688 [hyper,7300,1,2] $P(e(e(x, e(y, e(e(z, e(u, z)), u))), e(y, x))$.
 9403 [hyper,7445,1,4] $P(e(x, e(y, e(e(z, e(y, z)), x))))$.
 9647 [hyper,9403,1,607] $P(e(e(x, e(y, x)), e(z, e(z, y))))$.
 9908 [hyper,9647,1,4] $P(e(x, e(y, e(e(e(z, e(u, x)), e(u, z)), y))))$.
 10860 [hyper,7688,1,2] $P(e(e(x, e(e(y, e(x, y)), z)), z)$.
 10996 [hyper,10860,1,4711] $P(e(e(e(x, y), e(y, x)), e(z, z))$.
 11273 [hyper,10996,1,7300] $P(e(e(x, y), e(y, x))$.
 11323 [hyper,10996,1,992] $P(e(e(x, e(y, e(e(z, u), e(u, z))))), e(y, x))$.
 15435 [hyper,11323,1,9908] $P(e(e(e(x, y), e(y, e(x, z))), z)$.
 21396 [hyper,15435,1,11273] $P(e(x, e(e(y, z), e(z, e(y, x)))))$.
 28393 [hyper,21396,1,4] $P(e(e(e(x, e(y, z)), z), e(y, x))$.
 28395 [binary,28393,3] .

THEOREM 5:

```

assign(max_weight,20).
list(axioms).

% condensed detachment
-P(x) | -P(i(x,y)) | P(y).

end_of_list.

list(sos).

% formula suspected of being a single axiom
P(i(i(i(x,y),z),i(i(z,x),i(u,x)))).

% the negation of a known single axiom
-P(i(i(a,b),i(i(b,c),i(a,c)))).

end_of_list.

```

Proof of theorem 5

- 1 [] $\neg P(x) \mid \neg P(i(x,y)) \mid P(y)$.
- 2 [] $P(i(i(i(x,y),z),i(i(z,x),i(v,x))))$.
- 3 [] $\neg P(i(i(a,b),i(i(b,c),i(a,c))))$.
- 4 [hyper,2,1,2] $P(i(i(i(i(x,y),i(z,y)),i(y,u)),i(v,i(y,u))))$.
- 5 [hyper,4,1,4] $P(i(x,i(i(y,z),i(z,i(y,z))))$.
- 7 [hyper,5,1,2] $P(i(i(i(i(x,y),i(y,i(x,y))),z),i(u,z)))$.
- 12 [hyper,7,1,2] $P(i(x,i(i(i(y,i(z,y)),z),i(u,z)))$.
- 15 [hyper,12,1,2] $P(i(i(i(i(i(x,i(y,x)),y),i(z,y)),u),i(v,u)))$.
- 99 [hyper,15,1,4] $P(i(x,i(i(y,z),i(u,i(y,z))))$.
- 101 [hyper,99,1,99] $P(i(i(x,y),i(z,i(x,y))))$.
- 126 [hyper,101,1,2] $P(i(i(i(x,i(y,z)),y),i(u,y)))$.
- 132 [hyper,126,1,2] $P(i(i(i(x,y),i(z,i(y,u))),i(v,i(z,i(y,u))))$.
- 166 [hyper,132,1,101] $P(i(x,i(y,i(z,z))))$.
- 172 [hyper,166,1,166] $P(i(x,i(y,y)))$.
- 174 [hyper,166,1,2] $P(i(i(i(x,i(y,y)),z),i(u,z)))$.
- 175 [hyper,172,1,172] $P(i(x,x))$.
- 176 [hyper,172,1,2] $P(i(i(i(x,x),y),i(z,y)))$.
- 177 [hyper,175,1,2] $P(i(i(i(x,y),x),i(z,x)))$.
- 182 [hyper,176,1,4] $P(i(x,i(y,i(z,y))))$.
- 186 [hyper,182,1,2] $P(i(i(i(x,i(y,x)),z),i(u,z)))$.
- 187 [hyper,177,1,2] $P(i(i(i(x,y),i(y,z)),i(u,i(y,z))))$.
- 204 [hyper,187,1,2] $P(i(i(i(x,i(y,z)),i(u,y)),i(v,i(u,y))))$.
- 226 [hyper,204,1,2] $P(i(x,i(i(i(i(y,z),u),z),i(y,z))))$.
- 228 [hyper,204,1,2] $P(i(i(i(x,i(y,z)),i(u,i(z,v))),i(w,i(u,i(z,v))))$.
- 229 [hyper,226,1,226] $P(i(i(i(i(x,y),z),y),i(x,y)))$.
- 232 [hyper,229,1,2] $P(i(i(i(x,y),i(i(x,y),z)),i(u,i(i(x,y),z))))$.
- 290 [hyper,232,1,186] $P(i(x,i(i(i(y,i(z,y)),u),u)))$.
- 292 [hyper,232,1,177] $P(i(x,i(i(i(y,z),y),y)))$.
- 293 [hyper,232,1,176] $P(i(x,i(i(i(y,y),z),z)))$.
- 294 [hyper,232,1,174] $P(i(x,i(i(i(y,i(z,z)),u),u)))$.
- 301 [hyper,232,1,126] $P(i(x,i(i(i(y,i(z,u)),z),z)))$.
- 308 [hyper,292,1,292] $P(i(i(i(x,y),x),x))$.
- 310 [hyper,308,1,2] $P(i(i(x,i(x,y)),i(z,i(x,y))))$.
- 311 [hyper,293,1,308] $P(i(i(i(x,x),y),y))$.
- 313 [hyper,290,1,311] $P(i(i(i(x,i(y,x)),z),z))$.
- 315 [hyper,294,1,313] $P(i(i(i(x,i(y,y)),z),z))$.
- 317 [hyper,301,1,315] $P(i(i(i(x,i(y,z)),y),y))$.
- 319 [hyper,317,1,2] $P(i(i(x,i(y,i(x,z))),i(u,i(y,i(x,z))))$.
- 324 [hyper,310,1,310] $P(i(x,i(i(y,i(y,z)),i(y,z)))$.
- 326 [hyper,324,1,324] $P(i(i(x,i(x,y)),i(x,y)))$.

- 336 [hyper,326,1,204] $P(i(i(i(x,i(y,z)),i(u,y)),i(u,y)))$.
- 342 [hyper,326,1,187] $P(i(i(i(x,y),i(y,z)),i(y,z)))$.
- 347 [hyper,326,1,132] $P(i(i(i(x,y),i(z,i(y,u))),i(z,i(y,u))))$.
- 351 [hyper,342,1,2] $P(i(i(i(x,y),i(z,x)),i(u,i(z,x))))$.
- 378 [hyper,351,1,326] $P(i(i(i(x,y),i(z,x)),i(z,x)))$.
- 380 [hyper,351,1,229] $P(i(x,i(y,i(z,x))))$.
- 415 [hyper,380,1,2] $P(i(i(i(x,i(y,i(z,u))),z),i(v,z)))$.
- 428 [hyper,415,1,326] $P(i(i(i(x,i(y,i(z,u))),z),z))$.
- 432 [hyper,428,1,2] $P(i(i(x,i(y,i(z,i(x,u)))),$
 $i(v,i(y,i(z,i(x,u))))$).
- 470 [hyper,319,1,2] $P(i(x,i(i(y,z),i(i(i(z,u),y),z))))$.
- 472 [hyper,319,1,326] $P(i(i(x,i(y,i(x,z))),i(y,i(x,z))))$.
- 474 [hyper,470,1,470] $P(i(i(x,y),i(i(i(y,z),x),y)))$.
- 485 [hyper,474,1,380] $P(i(i(i(i(x,i(y,z)),u),z),i(x,i(y,z))))$.
- 503 [hyper,474,1,310] $P(i(i(i(i(x,i(y,z)),u),i(y,i(y,z))),$
 $i(x,i(y,z)))$).
- 511 [hyper,474,1,177] $P(i(i(i(i(x,y),z),i(i(y,u),y)),i(x,y)))$.
- 514 [hyper,474,1,126] $P(i(i(i(i(x,y),z),$
 $i(i(u,i(y,v)),y)),i(x,y)))$.
- 921 [hyper,228,1,2] $P(i(x,i(i(i(y,z),u),i(z,u))))$.
- 925 [hyper,921,1,921] $P(i(i(i(x,y),z),i(y,z)))$.
- 930 [hyper,925,1,2] $P(i(x,i(i(x,y),i(z,y))))$.
- 939 [hyper,930,1,474] $P(i(i(i(i(i(x,y),i(z,y)),u),x),$
 $i(i(x,y),i(z,y))))$.
- 944 [hyper,930,1,2] $P(i(i(i(i(i(x,y),z),i(u,z)),x),i(v,x)))$.
- 1005 [hyper,944,1,326] $P(i(i(i(i(i(x,y),z),i(u,z)),x),x))$.
- 1009 [hyper,1005,1,514] $P(i(i(i(i(i(x,i(y,z),u),$
 $i(y,z)),v),z),i(y,z)))$.
- 1196 [hyper,432,1,326] $P(i(i(x,i(y,i(z,i(x,u))),$
 $i(y,i(z,i(x,u))))$).
- 2020 [hyper,1009,1,511] $P(i(i(i(x,i(i(i(y,z),y),u),$
 $i(i(y,z),y)),y))$).
- 2106 [hyper,2020,1,939] $P(i(i(x,y),i(i(i(x,z),x),y)))$.
- 2115 [hyper,2106,1,930] $P(i(i(i(x,y),x),i(i(x,z),i(u,z))))$.
- 2150 [hyper,2115,1,503] $P(i(x,i(i(x,y),y)))$.
- 2151 [hyper,2115,1,485] $P(i(x,i(y,i(i(x,z),i(u,z))))$.
- 2439 [hyper,2150,1,2106] $P(i(i(i(x,y),x),i(i(x,z),z)))$.
- 2460 [hyper,2150,1,474] $P(i(i(i(i(i(x,y),y),z),x),i(i(x,y),y)))$.
- 2509 [hyper,2151,1,925] $P(i(x,i(y,i(i(i(z,x),u),i(v,u))))$.
- 2546 [hyper,2439,1,485] $P(i(x,i(y,i(i(x,z),z))))$.
- 2552 [hyper,2439,1,2] $P(i(i(i(i(x,y),y),i(x,z)),i(u,i(x,z))))$.
- 2607 [hyper,2546,1,925] $P(i(x,i(y,i(i(i(z,x),u),u))))$.
- 3233 [hyper,2552,1,326] $P(i(i(i(i(x,y),y),i(x,z)),i(x,z)))$.
- 3285 [hyper,3233,1,2607] $P(i(x,i(i(i(y,i(i(x,z),z)),u),u)))$.

- 3288 [hyper, 3233, 1, 2509] $P(i(x, i(i(i(y, i(i(x, z), z)), u), i(v, u))))$.
- 3290 [hyper, 3233, 1, 2151] $P(i(x, i(i(i(i(x, y), y), z), i(u, z))))$.
- 3335 [hyper, 3285, 1, 378] $P(i(i(i(x, i(i(i(y, z), u), u), y), y))$.
- 3339 [hyper, 3285, 1, 336] $P(i(i(i(x, i(i(i(y, i(z, u)), v), v), z), z))$.
- 3379 [hyper, 3290, 1, 472] $P(i(i(i(i(x, y), y), z), i(x, z)))$.
- 3618 [hyper, 3335, 1, 2460] $P(i(i(x, i(i(i(x, y), z), z)), i(i(i(x, y), z), z)))$.
- 5069 [hyper, 3288, 1, 347] $P(i(i(i(x, i(i(i(y, z), u), u), v), i(z, v)))$.
- 5152 [hyper, 3339, 1, 939] $P(i(i(x, y), i(i(i(z, i(x, u)), y), y)))$.
- 7789 [hyper, 5069, 1, 939] $P(i(i(i(x, y), z), i(i(i(u, x), z), z)))$.
- 10734 [hyper, 3618, 1, 5152] $P(i(i(i(i(x, y), i(x, z)), y), y))$.
- 10828 [hyper, 10734, 1, 7789] $P(i(i(i(x, i(i(y, z), i(y, u))), z), z))$.
- 11223 [hyper, 10828, 1, 939] $P(i(i(x, i(y, z)), i(i(y, x), i(y, z))))$.
- 11723 [hyper, 11223, 1, 2151] $P(i(i(x, y), i(x, i(i(y, z), i(u, z))))$.
- 15438 [hyper, 11723, 1, 3379] $P(i(x, i(i(x, y), i(i(y, z), i(u, z))))$.
- 20335 [hyper, 15438, 1, 1196] $P(i(i(x, y), i(i(y, z), i(x, z))))$.
- 20344 [binary, 20335, 3] .

Acknowledgments

This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

- [Bledsoe87] Bledsoe, W. W., private communication (1987).
- [Chou88] Chou, S. C., *Mechanical Geometry Theorem Proving*, D. Reidel, Dordrecht, Holland (1988).
- [Guard69] Guard, J., Oglesby, F., Bennett, J., and Settle, L., "Semi-automated mathematics", *Journal of the ACM* 16:49–62 (1969).
- [McCharen76] McCharen, J., Overbeek, R., and Wos, L., "Complexity and related enhancements for automated theorem-proving programs", *Computers and Mathematics with Applications* 2:1–16 (1976).
- [McCune88] McCune, W., unpublished information (1988).
- [McCune89] McCune, W., unpublished information (1989).

- [McCune90] McCune, W., "OTTER 2.0 users guide", technical report ANL-90/9, Argonne National Laboratory, Argonne, Ill. (1990).
- [Overbeek75] Overbeek, R., "An implementation of hyper-resolution", *Computers and Mathematics with Applications* 1:201–214 (1975).
- [Plaisted88] Plaisted, D. A., "Non-Horn clause logic programming without contrapositives", *Journal of Automated Reasoning* 4, no. 3:287–325 (1988).
- [RobinsonG69] Robinson, G., and Wos, L., "Paramodulation and theorem-proving in first-order theories with equality", pp. 135–150 in *Machine Intelligence*, vol. 4, ed. B. Meltzer and D. Michie, Edinburgh U.P., Edinburgh (1969).
- [RobinsonJ65a] Robinson, J., "A machine-oriented logic based on the resolution principle", *Journal of the ACM* 12:23–41 (1965).
- [RobinsonJ65b] Robinson, J., "Automatic deduction with hyper-resolution", *International Journal of Computer Mathematics* 1:227–234 (1965).
- [Stickel88] Stickel, M., "A Prolog technology theorem prover: Implementation by an extended Prolog compiler", *Journal of Automated Reasoning* 4, no. 4:353–380 (1988).
- [Winker79] Winker, S., and Wos, L., "Automated generation of models and counterexamples and its application to open questions in ternary Boolean algebra", pp. 251–256 in *Proceedings of the Eighth International Symposium on Multiple-Valued Logic*, IEEE and ACM, Rosemont, Ill. (1978).
- [Winker81] Winker, S., Wos, L., and Lusk, E., "Semigroups, antiautomorphisms, and involutions: A computer solution to an open problem, I", *Mathematics of Computation* 37:533–545 (1981).
- [Wos65] Wos, L., Robinson, G., and Carson, D., "Efficiency and completeness of the set of support strategy in theorem proving", *Journal of the ACM* 12:536–541 (1965).
- [Wos67] Wos, L., Robinson, G., Carson, D., and Shalla, L., "The concept of demodulation in theorem proving", *Journal of the ACM* 14:698–709 (1967).
- [Wos73] Wos, L., and Robinson, G., "Maximal models and refutation completeness: Semidecision procedures in automatic theorem proving", pp. 609–639 in *Word Problems: Decision Problems and the Burnside Problem in Group Theory*, ed. W. Boone, F. Cannonito, and R. Lyndon, North-Holland, New York (1973).
- [Wos83] Wos, L., Winker, S., Veroff, R., Smith, B., and Henschen, L., "Ques-

tions concerning possible shortest single axioms in equivalential calculus: An application of automated theorem proving to infinite domains", *Notre Dame Journal of Formal Logic* **24**:205–223 (1983).

[Wos84a] Wos, L., Veroff, R., Smith, B., and McCune, W., "The linked inference principle. II: The user's viewpoint," pp. 316–332 in *Proceedings of the Seventh International Conference on Automated Deduction*, vol. 170, *Lecture Notes in Computer Science*, ed. R. E. Shostak, Springer-Verlag, New York (1984).

[Wos84b] Wos, L., Winker, S., Veroff, R., Smith, B., and Henschen, L., "A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains", *Artificial Intelligence* **2**:303–356 (1984).

[Wos85] Wos, L., "Automated reasoning," *American Mathematical Monthly* **92**:85–92 (1985).

[Wos87] Wos, L., *Automated Reasoning: 39 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, N.J. (1987).

[Wos88] Wos, L., and McCune, W. W., "Searching for fixed point combinators by using automated theorem proving: A preliminary report", technical report ANL-88-10, Argonne National Laboratory, Argonne, Ill. (1988).

[Wos89] Wos, L., and McCune, W., "Automated theorem proving and logic programming: A natural symbiosis", *Journal of Logic Programming* (to appear).