Journal of Automated Reasoning **16:** 223–239, 1996.

Studies in Automated Reasoning

Using Hints to Increase the Effectiveness of an Automated Reasoning Program: Case Studies

ROBERT VEROFF
University of New Mexico, Albuquerque, NM 87131, U.S.A.
e-mail: veroff@cs.unm.edu

(Received: 6 March 1995)

Abstract. In this article we consider the use of hints to help guide the search for a proof. Under the hints strategy, the value of a generated clause is determined, in part, by whether or not the clause subsumes or is subsumed by a user-supplied hint clause. We have implemented the hints strategy and have experimented with it extensively. We summarize our experiences for a variety of reasoning tasks, including proof checking, proof completion, and proof finding. We conclude that the hints strategy has value beyond simply "giving the proof to find the proof."

Key words: hints strategy, automated reasoning programs, proof checking, proof completion, proof finding, weighting.

1. Introduction

In this article we describe our experiences with a new strategy, called the *hints strategy*, for increasing the effectiveness of automated reasoning programs. Under the hints strategy, a generated clause is given special consideration (as defined by the user) if it subsumes or is subsumed by a user-supplied hint clause. We have implemented the new strategy in the automated reasoning program OTTER [6] and have experimented with it extensively. What began as an exercise in the obvious – verifying the steps of a given proof – turned out to be more interesting and more valuable than we expected. The hints strategy is applicable to a variety of reasoning tasks, including proof checking, proof completion, and proof finding. Using the hints strategy, we have successfully checked proofs, found and corrected an error in a published proof, completed partial proofs, and used known proofs to help find shorter proofs and proofs of related theorems. It is especially significant, we believe, that each of the reported successes came quite easily.

The hints strategy is closely related to the weighting strategy [5]. In Section 2 we give a brief overview of the hints and weighting strategies in order to highlight the key similarities and differences. In Section 3 we summarize our experiments with the hints strategy for a variety of reasoning tasks. We conclude that the hints strategy is a valuable enhancement to an automated reasoning program.

2. Hints and Weighting

Search strategies are used to restrict and direct the application of inference rules. The weighting strategy [5] is used to restrict and direct the search by associating a value, called a weight, with every clause in the database of clauses and by ordering the application of inference rules according to these weights. User-supplied weight templates indicate preferences for symbols and terms appearing in clauses. Weight templates can be used to express intuition or knowledge about a problem or problem area in order to help guide the search for a proof or to prevent the consideration of certain clauses. Weighting also can be used to give preferences for the type of proof found. For example, when doing a circuit design, one might prefer a solution that emphasizes the use of "and" gates over the use of "or" gates. Years of experience have demonstrated the value of the weighting strategy to automated reasoning.

The hints strategy enhances the weighting strategy by adding a new dimension to the determination of weight values. The hints strategy is similar to the weighting strategy in that it makes use of user-defined patterns to determine the relative importance of generated clauses, but there are some significant differences.

Weighting: Weighting focuses on the assignment of a weight to each clause. User-supplied weight templates define a recursive mapping $wt()$ from terms and atoms to corresponding weight values. For example, the OTTER weight template

```
weight(f($(3),g($(2))), -15)
```

defines a mapping for terms of the form $f(t_1, g(t_2))$, where t_1 and t_2 are arbitrary terms. The $(3) and $(2) patterns in the template are multipliers for the weights of the corresponding subterms; the -15 indicates a value that is added to the result. Specifically,

$$wt(f(t_1, g(t_2))) = 3 * wt(t_1) + 2 * wt(t_2) - 15.$$

The weight of a clause is then defined as a simple function of the weights of its atoms.

The process of matching weight templates and terms can be defined in terms of instantiation. For example, the weight template

```
weight(f($(3),g($(2))), -15)
```

matches any instance of the term $f(x, g(y))$; the term is derived from the weight template by replacing each $ multiplier by a unique variable identifier.

Hints: In contrast to weighting, the hints strategy focuses directly on the identification of key clauses rather than on the general calculation of weights. Any generated clause that subsumes or is subsumed by a user-supplied hint clause is identified as being "interesting". The weight of such a clause is adjusted (either

positively or negatively) according to user preferences; the cases of subsuming a hint, being subsumed by a hint, or both are controlled separately. The case in which a generated clause subsumes a hint clause is probably the most interesting and potentially valuable. Such generated clauses can be viewed as key milestones on the way to a proof. (If a fact is deemed significant, then for many applications of automated reasoning, anything that subsumes the fact would be just as significant.) The case in which a clause is subsumed by a hint looks more like weighting (i.e., identifying instances of user-defined patterns) and may be less interesting, although there are problems for which finding instances of key clauses is significant. The third case, identifying clauses that both subsume and are subsumed by a hint, is particularly useful for various types of proof checking.

In summary, the hints strategy enhances the weighting strategy, first, by focusing on entire facts (i.e., clauses) rather than terms and subterms, and second, by using subsumption as a criterion for determining the value of a generated clause. Being based on subsumption, the hints strategy adds a semantic, or logical, component to the evaluation of a clause.

3. Case Studies

The hints strategy has been implemented in OTTER and experimented with extensively. In the current implementation, the user can specify a list of hint clauses, a *hint weight value*, and one of the following three conditions.*

- A generated clause subsumes a hint clause.
- A hint clause subsumes a generated clause.
- A generated clause is equivalent to a hint clause. (For our purposes, clauses A and B are equivalent if and only if A and B subsume each other.)

The hint weight value is *added* to the weight – as computed in the usual way under the weighting strategy – of any generated clause that satisfies the specified condition. Since the user-specified hint weight value can be positive or negative, the hints strategy can be used either to focus on or to avoid generated clauses that satisfy the specified condition.

In this section, we summarize our experiments for a variety of reasoning tasks and for a variety of application areas. The reasoning tasks range from proof checking to finding new proofs. The applications include problems from ring theory, various logical calculi, and Robbins algebra [14]. Our objective is to point out the range of reasoning tasks for which the hints strategy is valuable; hence we organize our case studies by reasoning task rather than by application.

* The user can specify more than one condition, each with its own hint weight value, but this typically is not done. It does appear desirable, however, to allow the user to specify different hint weight values for different hint clauses. This option is being considered for future implementations.

3.1. PROOF CHECKING

Proof checking can mean a variety of things, ranging from verifying specific applications of inference rules to completing proofs for which only key, high-level steps are given. In the limiting case, the distinction between proof checking and proof finding can be somewhat arbitrary. For the sake of discussion, we have grouped our experiments roughly into three categories: proof checking, proof completion, and proof finding. The differences among the categories are in the nature of the hints provided and the extent to which the automated reasoning program is required to follow the hints.

Say we are given a set of hypotheses, a set of inference rules, and a sequence $S = \{C_1, C_2, \ldots, C_k\}$ of clauses purported to be a proof of the clause C_k. There are several questions that we might ask about this "proof", including, for example, the following.

- Is the proof correct? That is, is each clause in the sequence S derivable (with the given inference rules) from the hypotheses and clauses appearing earlier in the sequence?
- Is the proof complete? That is, is each clause in S derivable with a *single* application of an inference rule? Of course, an incomplete proof is not a proof in the formal sense; proof completion is discussed in the next section.
- Is the proof minimal? That is, is there a proper subsequence of clauses from S (not necessarily adjacent in S) that gives a complete proof of C_k? A proof would not be minimal if, for example, it contained derived clauses that do not actually participate in the derivation of C_k.
- Is the proof minimum? That is, does there exist a "shorter proof" of C_k with respect to some given measure for the length of a proof. For one example, we could ask whether there is a proof consisting of strictly fewer derived clauses. Alternatively, we could focus on the depth of the corresponding proof trees and ask whether there is a proof with strictly less depth.

We have used OTTER to help us answer these and a variety of related questions.* The following three examples illustrate the applicability of the hints strategy to answering questions about correctness and completeness. Example 6, later in this article, discusses the applicability of the hints strategy to the problem of finding shorter proofs.

Example 1, A Straightforward Proof-Checking Exercise

The five axioms

```
(EX1-1) P(i(x,i(y,x)))                    # Axiom 1.
(EX1-2) P(i(i(x,y),i(i(y,z),i(x,z))))     # Axiom 2.
```

* We have experimented with even more restrictive notions of proof checking that require, for example, generating proof clauses in a specified order and with specified ancestry. This work involves the use of special representations and special-purpose demodulators and is not reported in this article.

```
(EX1-3) P(i(i(i(x,y),y),i(i(y,x),x)))        # Axiom 3.
(EX1-4) P(i(i(i(x,y),i(y,x)),i(y,x)))        # Axiom 4.
(EX1-5) P(i(i(n(x),n(y)),i(y,x)))            # Axiom 5.
```

together with the inference rule condensed detachment are known to be a complete system for the many-valued sentential calculus [4, 16]. It is also known that condensed detachment can be applied by using hyperresolution and the following nucleus.

```
(EX1-6) -P(i(x,y)) | -P(x) | P(y)            # Condensed Detachment.
```

The problem is to show that Axiom 4 is dependent on the four remaining axioms, specifically, by showing that Axiom 4 can be derived from Axioms 1, 2, 3, and 5 with repeated applications of condensed detachment. For proof by contradiction, the input consists of Axioms 1, 2, 3, and 5 and the negation of Axiom 4.

```
(EX1-7) -P(i(i(i(a,b),i(b,a)),i(b,a)))       # Negation of Axiom 4.
```

We were given a proof of this fact that consists of 63 steps, representing 63 applications of condensed detachment; Step 63 is clause (EX1-4). Our task, for this example, is simply to verify that the 63 given steps are sufficient to prove the result. We do this by including each of the 63 steps as a hint clause and instructing the automated reasoning program to keep a generated clause if and only if it is equivalent to a hint. Within this set of clauses, the program is instructed to focus on clauses in the order that they are generated (i.e., to perform a breadth-first search of the search space).

It is no surprise, of course, that a proof is obtained immediately. Further, the breadth-first requirement increases the likelihood that the automated reasoning program will find the given proof rather than a different proof that happens to use a subset of the given clauses. Our original proof did, in fact, use each of the 63 given clauses. When we dropped the breadth-first requirement in a second experiment, we obtained a 58-step proof consisting of a proper subset of the original 63 clauses.

We note that there are subtleties associated even with this seemingly trivial use of automated reasoning. It is likely that a proof being checked was *not* generated by the system doing the proof checking. Automated reasoning programs are complex systems that are influenced by representation and by strategies for keeping information in a simplified or canonical form. Consider, for example, problems that involve commutative and/or associative operators. In order for an approach using hints to be effective, it is essential that hints and generated clauses be subject to the same strategies for maintaining information. This issue is addressed in Example 2.

Example 2, Checking a Proof for Which Representation Is an Issue

The $x^3 = x$ problem for associative rings ($x^3 = x$ implies commutativity) has been considered to be a benchmark problem for automated reasoning programs

[1]. We reported, in [10], a proof of the problem using the automated reasoning program AURA [8]. Since that time, the problem has been attacked with a variety of approaches [2, 9, 13]. T. C. Wang's proof is quite a bit different from other proofs we have seen, so we decided to try to verify it with OTTER.

Clauses corresponding to Wang's original input are as follows.

```
% (xy)z = x(yz)
(EX2-W1) EQ(ADD(ADD(x,y),z),ADD(x,ADD(y,z))).

% x - xxx = e
(EX2-W2) EQ(ADD(x,INV(MULT(x,MULT(x,x)))),e).

% xxy + xyx + xyy + yxx + yxy + yyx = e
(EX2-W3) EQ(ADD(MULT(x,MULT(x,y)),ADD(MULT(x,MULT(y,x)),
            ADD(MULT(x,MULT(y,y)),ADD(MULT(y,MULT(x,x)),
            ADD(MULT(y,MULT(x,y)),MULT(y,MULT(y,x)))))),e).

% xyz + xzy + yxz + yzx + zxy + zyx = e
(EX2-W4) EQ(ADD(MULT(x,MULT(y,z)),ADD(MULT(x,MULT(z,y)),
            ADD(MULT(y,MULT(x,z)),ADD(MULT(y,MULT(z,x)),
            ADD(MULT(z,MULT(x,y)),MULT(z,MULT(y,x)))))),e).
```

Wang's proof has two basic steps. The first step consists of generating ground equations by taking ground instances of the input axioms and by multiplying on the right and on the left by ground monomials. The second step consists of adding and subtracting six of the resulting equations to conclude that $ab - ba = e$, for arbitrary a and b.

Our attempt to verify Wang's proof required three sets of clauses: clauses to systematically generate new ground monomials, clauses to ground Wang's input axioms and to multiply by the ground monomials, and a clause to add and subtract equations. In addition, we used a set of input demodulators to help keep generated clauses in a reduced, or canonical, form [12].

The following clauses are used to generate ground monomials (for constants a and b).

```
% a and b are monomials.
(EX2-M1) MONO(a).
(EX2-M2) MONO(b).

% If x and y are monomials, then so are INV(x) and MULT(x,y).
(EX2-M3) -MONO(x) | MONO(INV(x)).
(EX2-M4) -MONO(x) | -MONO(y) | MONO(MULT(x,y)).
```

The following clauses are used to generate Wang's equations by instantiating axioms (EX2-W2), (EX2-W3), and (EX2-W4) and by multiplying on the right and left by ground monomials. Since we use demodulation to right associate expressions with respect to the ADD predicate, we do not include the clauses

corresponding to axiom (EX2-W1); such clauses would not participate in any significant applications of resolution.

```
% instantiate only

(EX2-E1) -MONO(x) | EQ(ADD(x,INV(MULT(x,MULT(x,x)))),e).

(EX2-E2) -MONO(x) | -MONO(y) |
             EQ(ADD(MULT(x,MULT(x,y)),ADD(MULT(x,MULT(y,x)),
                ADD(MULT(x,MULT(y,y)),ADD(MULT(y,MULT(x,x)),
                ADD(MULT(y,MULT(x,y)),MULT(y,MULT(y,x)))))),e).

(EX2-E3) -MONO(x) | -MONO(y) | -MONO(z) |
             EQ(ADD(MULT(x,MULT(y,z)),ADD(MULT(x,MULT(z,y)),
                ADD(MULT(y,MULT(x,z)),ADD(MULT(y,MULT(z,x)),
                ADD(MULT(z,MULT(x,y)),MULT(z,MULT(y,x)))))),e).

% instantiate and multiply on the right

(EX2-E4) -MONO(w) | -MONO(x) |
             EQ(ADD(MULT(x,w),INV(MULT(x,MULT(x,MULT(x,w))))),e).

(EX2-E5) -MONO(w) | -MONO(x) | -MONO(y) |
             EQ(ADD(MULT(x,MULT(x,MULT(y,w))),
                ADD(MULT(x,MULT(y,MULT(x,w))),
                ADD(MULT(x,MULT(y,MULT(y,w))),
                ADD(MULT(y,MULT(x,MULT(x,w))),
                ADD(MULT(y,MULT(x,MULT(y,w))),
                    MULT(y,MULT(y,MULT(x,w)))))))),e).

(EX2-E6) -MONO(w) | -MONO(x) | -MONO(y) | -MONO(z) |
             EQ(ADD(MULT(x,MULT(y,MULT(z,w))),
                ADD(MULT(x,MULT(z,MULT(y,w))),
                ADD(MULT(y,MULT(x,MULT(z,w))),
                ADD(MULT(y,MULT(z,MULT(x,w))),
                ADD(MULT(z,MULT(x,MULT(y,w))),
                    MULT(z,MULT(y,MULT(x,w)))))))),e).

% instantiate and multiply on the left

(EX2-E7) -MONO(w) | -MONO(x) |
            EQ(ADD(MULT(w,x),INV(MULT(w,MULT(x,MULT(x,x))))),e).

(EX2-E8) -MONO(w) | -MONO(x) | -MONO(y) |
             EQ(ADD(MULT(w,MULT(x,MULT(x,y))),
                ADD(MULT(w,MULT(x,MULT(y,x))),
                ADD(MULT(w,MULT(x,MULT(y,y))),
                ADD(MULT(w,MULT(y,MULT(x,x))),
                ADD(MULT(w,MULT(y,MULT(x,y))),
                    MULT(w,MULT(y,MULT(y,x)))))))),e).
```

```
(EX2-E9) -MONO(w) | -MONO(x) | -MONO(y) | -MONO(z) |
            EQ(ADD(MULT(w,MULT(x,MULT(y,z))),
               ADD(MULT(w,MULT(x,MULT(z,y))),
               ADD(MULT(w,MULT(y,MULT(x,z))),
               ADD(MULT(w,MULT(y,MULT(z,x))),
               ADD(MULT(w,MULT(z,MULT(x,y))),
                   MULT(w,MULT(z,MULT(y,x)))))))),e).
```

The following clause can be used to derive the combination of six equations specified in Wang's proof. The first three negative literals correspond to equations to be subtracted; the next three negative literals correspond to equations to be added.

```
% template for adding and subtracting 6 equations
(EX2-S1) -EQ(x1,e) | -EQ(x2,e) | -EQ(x3,e) |
         -EQ(x4,e) | -EQ(x5,e) | -EQ(x6,e) |
             EQ(ADD(INV(x1),ADD(INV(x2),ADD(INV(x3),
                ADD(x4,ADD(x5,x6))))),e).
```

To ensure the cancellation of like terms, a general subtraction template would include qualifications to force unification between equations. This is not necessary for this problem because the six key equations are known to be ground. Without *any* qualifications, however, there are 6^6 ways to match the six key equations with the six negative literals of template (EX2-S1). The following template is designed to reduce the number of undesirable and redundant combinations. Specifically, it ensures that six *distinct* equations will be combined, that any three equations will match the first three negative literals in at most one way, and that any three equations will match the next three negative literals in at most one way.

```
% modified template for adding and subtracting 6 equations
(EX2-S2) -EQ(x1,e) | -EQ(x2,e) | -EQ(x3,e) |
            -$LLT(x1,x2) | -$LLT(x2,x3) |
            -EQ(y1,e) | -EQ(y2,e) | -EQ(y3,e) |
            -$LLT(y1,y2) | -$LLT(y2,y3) |
             $ID(x1,y1) | $ID(x1,y2) | $ID(x1,y3) |
             $ID(x2,y1) | $ID(x2,y2) | $ID(x2,y3) |
             $ID(x3,y1) | $ID(x3,y2) | $ID(x3,y3) |
                EQ(ADD(INV(x1),ADD(INV(x2),ADD(INV(x3),
                   ADD(y1,ADD(y2,y3))))),e).
```

The $LLT and $ID predicates refer to OTTER's underlying lexical ordering of terms. The positive $ID literals ensure that the three equations being added will be distinct from the three equations being subtracted. The negative $LLT literals ensure that only one ordering within each set of three equations will be considered. Because much of the work associated with combining equations is in the processing (e.g., demodulation) of the positive unit resolvent, using (EX2-S2) instead of (EX2-S1) results in substantially better performance.

Our task, for this example, is to simulate the steps of Wang's proof. We do this by including Wang's six key equations as hints and by instructing the automated reasoning program to keep a generated clause if and only if it is an instance of predicate MONO (i.e., a new ground monomial) or it is equivalent to a hint.

The hint clauses are as follows.

```
% ba - bbba = e
(EX2-H1) EQ(ADD(MULT(b,a),INV(MULT(b,MULT(b,MULT(b,a))))),e).

% ab - abbb = e
(EX2-H2) EQ(ADD(MULT(a,b),INV(MULT(a,MULT(b,MULT(b,b))))),e).

% aabb + abab + abbb + baab + babb + bbab = e
(EX2-H3) EQ(ADD(MULT(b,MULT(a,MULT(a,b))),
             ADD(MULT(b,MULT(a,MULT(b,a))),
             ADD(MULT(b,MULT(a,MULT(b,b))),
             ADD(MULT(b,MULT(b,MULT(a,a))),
             ADD(MULT(b,MULT(b,MULT(a,b))),
                 MULT(b,MULT(b,MULT(b,a)))))))),e).

% aabb + abab + abbb + baab + babb + bbab = e
(EX2-H4) EQ(ADD(MULT(a,MULT(a,MULT(b,b))),
             ADD(MULT(a,MULT(b,MULT(a,b))),
             ADD(MULT(a,MULT(b,MULT(b,b))),
             ADD(MULT(b,MULT(a,MULT(a,b))),
             ADD(MULT(b,MULT(a,MULT(b,b))),
                 MULT(b,MULT(b,MULT(a,b)))))))),e).

% abab + abba + baab + 2(baba) + bbaa = e
(EX2-H5) EQ(ADD(MULT(a,MULT(b,MULT(a,b))),
             ADD(MULT(a,MULT(b,MULT(b,a))),
             ADD(MULT(b,MULT(a,MULT(a,b))),
             ADD(H(MULT(b,MULT(a,MULT(b,a))),2),
                 MULT(b,MULT(b,MULT(a,a)))))),e).

% aabb + 2(abab) + abba + baab + baba = e
(EX2-H6) EQ(ADD(MULT(a,MULT(a,MULT(b,b))),
             ADD(H(MULT(a,MULT(b,MULT(a,b))),2),
             ADD(MULT(a,MULT(b,MULT(b,a))),
             ADD(MULT(b,MULT(a,MULT(a,b))),
                 MULT(b,MULT(a,MULT(b,a)))))),e).
```

These are Wang's six equations, but they have been mapped into our representation, and they have been demodulated by the input demodulators. We note that our representation includes integer coefficients for polynomial terms. The term H(MULT(a,MULT(b,MULT(a,b))),2) in clause (EX2-H6), for example, represents Wang's term $2(abab)$.

The automated reasoning program got the proof on the first attempt.

Example 3, Finding and Correcting an Error in a Published Proof

Because of the easy success with Wang's proof of the $x^3 = x$ problem, we decided to try to check his proof of the $x^4 = x$ problem, also reported in [13]. We used the same strategy described in Example 2, except we separated the proof into two steps, one to generate the twelve key equations given in Wang's proof, and a second to combine the equations exactly as specified in the proof. The exact combination is easy to specify in a separate experiment by giving each of the twelve key equations a distinct predicate name and modifying the analog of template (EX2-S1) accordingly.

Using this strategy, we got the twelve key equations of Wang's proof in the first attempt. We failed, however, to confirm Wang's combination of the twelve equations to complete the proof. A third experiment was designed to compare the result of the specified combination of equations with the twelve hint clauses. The result showed an extra two copies of one of the hint equations (the eleventh), indicating that the sign for this equation is incorrect in the published proof. In other words, the equation should be subtracted instead of added in the combination. The corrected combination is easily verified.

It is interesting to note, we believe, that the hardest and most time consuming part of this exercise was to get Wang's equations into a clause format suitable for the automated reasoning program. We did this with a substantial amount of editing and by using the automated reasoning program as a term-rewriting system.

3.2. Proof Completion

In this section we look at problems for which we have some, but not all, of the clauses for a proof. The objective is to complete the proof.

Example 4, Remembering a Forgotten Proof

While working with Wang's proof of $x^3 = x$, we realized that we had pretty much forgotten the details of the proof we had obtained ourselves previously, although we did remember the key significant steps. We decided to use Otter to try to "remember" the proof. In the first experiment, we included the following as hints,

```
% 6x = e
(EX4-1) EQ(H(x,6),e).

% 3x + 3xx = e
(EX4-2) EQ(ADD(H(x,3),H(MULT(x,x),3)),e).

% 3xy + 3yx = e
(EX4-3) EQ(ADD(H(MULT(x,y),3),H(MULT(y,x),3)),e).
```

```
% 2xxy + 2xyx + 2yxx = e
(EX4-4) EQ(ADD(H(MULT(x,MULT(x,y)),2),
           ADD(H(MULT(x,MULT(y,x)),2),
               H(MULT(y,MULT(x,x)),2))),e).

% 2xxyz + 2xyzx + 2yzxx = e
(EX4-5) EQ(ADD(H(MULT(x,MULT(x,MULT(y,z))),2),
           ADD(H(MULT(x,MULT(y,MULT(z,x))),2),
               H(MULT(y,MULT(z,MULT(x,x))),2))),e).

% 2xy - 2yx = e
(EX4-6) EQ(ADD(H(MULT(x,y),2),H(INV(MULT(y,x)),2)),e).
```

and instructed the reasoning program to focus on clauses that are equivalent to hints. It is no surprise, of course, that the proof was obtained immediately. In further experiments, we gave increasingly vague versions of our memory of the proof. For example, in one experiment we included the following as hints,

```
% identify clauses that are subsumed by
%    c1*xy + c2*yx = e
% for arbitrary coefficients c1 and c2
(EX4-7) EQ(ADD(H(MULT(x,y),vc1),H(MULT(y,x),vc2)),e).
(EX4-8) EQ(ADD(H(MULT(x,y),vc1),H(INV(MULT(y,x)),vc2)),e).

% identify clauses that are subsumed by
%    c*term1 + c*term2 + c*term3 = e
% for arbitrary coefficient c
(EX4-9) EQ(ADD(H(x,vc),ADD(H(y,vc),H(z,vc))),e).
```

and instructed the reasoning program to favor instances of these general hints. Again, we obtained a proof, using no special weighting templates and no other special strategies. In another experiment, we determined that the single hint (EX4-9) suffices.

Example 5, Finding a More Detailed Proof

The following are the axioms for a Robbins algebra. For all x, y, and z,

```
(EX5-1) o(x,y) = o(y,x)                        (commutativity)
(EX5-2) o(o(x,y),z) = o(x,o(y,z))              (associativity)
(EX5-3) n(o(n(o(x,y)),n(o(x,n(y))))) = x       (the "Robbins axiom")
```

It is, at least at the time of this writing, an open question whether all Robbins algebras are Boolean algebras – that is, whether the axioms for a Boolean algebra follow logically from the Robbins axioms. (The implied interpretation is that "o" and "n" correspond respectively to "or" and "not" in the Boolean algebra.) In [14], Steve Winker shows several conditions that, when taken together with the Robbins axioms, are sufficient to imply the Boolean algebra. Several of the

proofs of these results appear to be out of reach of current automated reasoning programs.

One of the results, that the axiom

```
(EX5-4) (exists x, y (o(x,y) = x))
```

suffices as an added condition, was proved by Bill McCune using an automated reasoning program with associative-commutative (AC) unification built in [7]. McCune's proof is based on another known (and easily proved) result, that the axiom

```
(EX5-5) (exists x (o(x,x) = x))
```

itself suffices. That is, McCune shows that the three axioms for Robbins algebra together with (EX5-4) logically implies (EX5-5), which in turn implies the axioms for a Boolean algebra. McCune's proof, in his representation, is as follows.

```
(EX5-P01) -(x+x=x)                                  # Negation of (EX5-5).
(EX5-P02) C+D=C                                     # (EX5-4).
(EX5-P03) n(n(n(x)+y)+n(x+y))=y                     # Robbins axiom.
(EX5-P04) n(n(C)+n(D+n(C)))=D.
(EX5-P05) n(n(C+x+y)+n(D+n(C+x)+y))=D+y.
(EX5-P06) n(D+n(C+n(D+n(C))))=n(D+n(C)).
(EX5-P07) n(n(n(n(x)+y)+n(x+y)+z)+n(y+z))=z.
(EX5-P08) n(n(n(n(x)+y)+x+y)+y)=n(n(x)+y).
(EX5-P09) n(n(C)+n(D+n(C+n(x))+n(C+x)))=D.
(EX5-P10) n(n(D+n(C+n(D+n(C)))+x)+n(n(D+n(C))+x))=x.
(EX5-P11) n(n(C+n(D+n(C)))+n(D+n(C)))=D.
(EX5-P12) n(n(C+n(C+n(D+n(C))))+n(C+n(D+n(C))))=C.
(EX5-P13) n(D+n(D+n(C)+n(C+n(D+n(C)))))=n(C+n(D+n(C))).
(EX5-P14) n(D+n(n(C)+n(n(D+n(C))+n(x))+n(n(D+n(C))+x)))=n(C).
(EX5-P15) n(n(n(n(n(x)+y)+x+y)+y+z)+n(n(n(x)+y)+z))=z.
(EX5-P16) n(C+n(D+n(C)))=n(C).
(EX5-P17) n(n(C)+n(C+n(C)))=C.
(EX5-P18) n(n(C+x)+n(n(C)+n(C+n(C))+x))=x.
(EX5-P19) n(n(C+C+n(C+n(C)))+n(C+n(C)))=C.
(EX5-P20) n(C+n(C+n(C)+n(C+C+n(C+n(C)))))=n(C+C+n(C+n(C))).
(EX5-P21) n(C+C+n(C+n(C)))=n(C).
(EX5-P22) D+n(C+n(C))=D.
(EX5-P23) C+n(C+n(C))=C.
(EX5-P24) n(C+n(C))+x=x                          # Contradicts (EX5-P01).
```

Axioms (EX5-1) and (EX5-2) are not present explicitly, since they are already built into the AC unification.

Using the steps of McCune's AC-proof (mapped into our representation) as hints, we were able to get a proof of the same result *without* the use of AC unification. Our initial thought was simply to fill in the AC proof with the relevant

applications of commutativity and associativity. In fact, the new 45-step non-AC proof uses only eight of the generated clauses from McCune's 24-step AC proof.

3.3. PROOF FINDING

In [16] and [17], Wos describes the *resonance strategy* and its applicability to finding shorter proofs and to using the proof of one problem to help find a proof for a related problem. Resonance is a strategy that guides a user's choice for weight templates. A special class of weight templates, called *resonators*, is defined that has been shown to be effective for a variety of problems. Similar to the hints strategy reported in this article, the focus of the resonance strategy is on the selection of clauses rather than the weighting of terms. Being an application of weighting, however, resonance is subject to weighting's limited notion of matching. We believe that the subsumption-based matching criteria of the hints strategy (e.g., when a newly generated clause subsumes a hint clause) is especially significant.

We performed experiments with several of the same problems reported in [16]. The hints strategy performed as well or better than the resonance strategy in all of the problems tried.

Example 6, Finding a Shorter Proof

As reported in [16], Dana Scott presented Larry Wos with a set of challenge problems in sentential calculus. The original problem consisted of three theses (Theses 1, 2, and 3), known to be a complete set of axioms for the two-valued sentential calculus [3] and sixty-eight theses to be proved (Theses 4 through 71). Many of the examples in [16] are based on this set of problems, including derivations from various subsets of the seventy-one theses.

In this example, we describe our experience using the hints strategy to find a shorter proof for one of the problems discussed by Wos. The problem is to show that Thesis 21 can be derived from Theses 18 and 35 with repeated applications of condensed detachment.

```
(EX6-1) P(i(x,i(y,x)))                          # Thesis 18.
(EX6-2) P(i(i(x,i(y,z)),i(y,i(x,z))))           # Thesis 21.
(EX6-3) P(i(i(x,i(y,z)),i(i(x,y),i(x,z))))      # Thesis 35.
```

For proof by contradiction, the input consists of Theses 18 and 35 and the negation of Thesis 21.

```
(EX6-4) -P(i(i(a,i(b,c)),i(b,i(a,c))))          # Negation of Thesis 21.
```

Our approach to finding a shorter proof consisted of a sequence of three experiments. In the first experiment, we proved Thesis 21 from Theses 18 and

35, using all seventy-one theses as hints. The resulting proof, which required thirteen applications of condensed detachment, contained nine clauses *not* from the original set of theses. In the second experiment, we again proved Thesis 21 from Theses 18 and 35, but we used the clauses from the first proof as hints. Rather than using ordinary hyperresolution, however, we used linked-UR resolution as the inference rule [11]. Linked UR-resolution has the effect of combining multiple applications of condensed detachment into a single inference step. The significance to this problem is that we are looking for new derivations of clauses known to be useful in proving the result; linking provides a local permutation of the search space, which increases the likelihood of finding new derivations. The proof resulting from the second experiment consisted of exactly nine applications of condensed detachment. In the final experiment, we returned to ordinary hyperresolution, but we used the clauses from the most recent proof as hints. Using this approach, we were able to find a proof with exactly eight applications of condensed detachment. In each of the three experiments, we instructed the automated reasoning program to focus on clauses that subsume hints.

Example 7, Blocking the Participation of Clauses in a Proof

Occasionally we are interested in knowing whether a particular theorem can be proved *without* using certain lemmas as intermediate results. For example, each of the following axioms is known to be a single axiom for the equivalential calculus [15]. Specifically, any one of the following axioms can be used, with the inference rule condensed detachment, to derive all valid formulas of the theory.

```
(EX7-01) P(e(e(x,y),e(e(z,y),e(x,z))))      # Axiom YQL.
(EX7-02) P(e(e(x,y),e(e(x,z),e(z,y))))      # Axiom YQF.
(EX7-03) P(e(e(x,y),e(e(z,x),e(y,z))))      # Axiom YQJ.
(EX7-04) P(e(e(e(x,y),z),e(y,e(z,x))))      # Axiom UM.
(EX7-05) P(e(x,e(e(y,e(x,z)),e(z,y))))      # Axiom XGF.
(EX7-06) P(e(e(x,e(y,z)),e(z,e(x,y))))      # Axiom WN.
(EX7-07) P(e(e(x,y),e(z,e(e(y,z),x))))      # Axiom YRM.
(EX7-08) P(e(e(x,y),e(z,e(e(z,y),x))))      # Axiom YRO.
(EX7-09) P(e(e(e(x,e(y,z)),z),e(y,x)))      # Axiom PYO.
(EX7-10) P(e(e(e(x,e(y,z)),y),e(z,x)))      # Axiom PYM.
(EX7-11) P(e(x,e(e(y,e(z,x)),e(z,y))))      # Axiom XGK.
(EX7-12) P(e(x,e(e(y,z),e(e(x,z),y))))      # Axiom XHK.
(EX7-13) P(e(x,e(e(y,z),e(e(z,x),y))))      # Axiom XHN.
```

Given an ordered pair (A,B) of single axioms from the list of thirteen, we may be interested in knowing whether B can be derived from A without first deriving any of the eleven *remaining* single axioms as an intermediate result.

For this reasoning task we simply include the eleven remaining single axioms as hints and instruct the reasoning program *not* to focus on any generated clause

that subsumes a hint. Using this simple strategy, we have been able to find 90 such proofs out of the 156 (13 * 12) candidate pairs for the thirteen given axioms. This study is still in progress.

Example 8, Increasing the Goal-oriented Nature of the Search

When the denial of a theorem consists of a multiliteral clause, it makes good sense to instruct the automated reasoning program to focus as soon as possible on unit clauses that can resolve with the literals of the denial. For example, when the theorem is universal (i.e., contains no existentially quantified variables) and the proof search is forward in nature, it is useful to include as hints unit clauses corresponding to the positive statement of the theorem. By instructing the automated reasoning program to focus on any generated clause that subsumes a hint, we help ensure that the automated reasoning program will recognize a contradiction as soon as possible. This is a somewhat obvious application of the hints strategy, but it does highlight the significance of using subsumption – specifically, the subsumption of a hint clause by a generated clause – as a selection criterion.

Using hints to focus on the literals of a multiliteral denial clause has been useful for proving that one axiom system logically implies another (and, ultimately, for showing that two axiom systems are equivalent). For example, we can show that the system

```
(EX8-1) P(i(i(i(x,y),z),i(y,z)))                  # Thesis 19.
(EX8-2) P(i(i(i(x,y),z),i(n(x),z)))               # Thesis 37.
(EX8-3) P(i(i(n(x),y),i(i(z,y),i(i(x,z),y))))     # Thesis 59.
```

(from Scott's challenge) with condensed detachment is complete for the sentential calculus by deriving the system

```
(EX8-4) P(i(i(x,y),i(i(y,z),i(x,z))))             # Thesis  1.
(EX8-5) P(i(i(n(x),x),x))                         # Thesis  2.
(EX8-6) P(i(x,i(n(x),y)))                         # Thesis  3.
```

which is known to be complete. For proof by contradiction, we include the following denial clause.

```
% negation of Theses 1, 2, and 3
(EX8-7) -P(i(i(a,b),i(i(b,c),i(a,c)))) |
        -P(i(i(n(a),a),a)) |
        -P(i(a,i(n(a),b))).
```

To focus on this denial, we include clauses (EX8-4), (EX8-5), and (EX8-6) as hints as well as any other hints that may help us solve the problem. In this case, without any other insight, we include all of the original 71 theses as hints, and we get the proof using no weighting templates or any other special strategies.

The hints corresponding to the literals of the denial are especially significant to the final step of the proof.

Remark. We note that the Boyer–Moore theorem prover [2] has a "hints" feature for directing a proof. While the hints strategy presented in this article focuses on the evaluation and consideration of newly generated clauses, hints are used in the Boyer–Moore prover to force the use of specific instances of previously-proved theorems, definitions, and axioms and the use of specific rewrite rules. Althrough there are similarities when using hints for proof checking, the two approaches are somewhat different in spirit and intent.

4. Summary

We have described a new strategy for increasing the effectiveness of an automated reasoning program. Under the hints strategy, the value of a clause is determined, in part, by whether or not it subsumes or is subsumed by a user-supplied hint clause.

The hints strategy is applicable to a variety of reasoning tasks, including proof checking, proof completion, and proof finding. The application to proof checking and proof completion is natural and intuitive; the strategy ensures that the automated reasoning program will focus on clauses that are known to participate in a proof. The application to proof finding is less direct, providing a new mechanism for a user to supply intuition or knowledge to the automated reasoning program.

We have implemented the hints strategy in OTTER and have experimented with it extensively. We conclude that the use of hints can be an effective strategy for the automation of reasoning and that support for the hints strategy provides a valuable enhancement to an automated reasoning program.

References

1. Bledsoe, W. W.: Non-resolution theorem proving, *Artificial Intelligence* **9** (1977), 1–35.
2. Boyer, R. S., and Moore, J. S. *A Computational Logic Handbook,* Academic Press, San Diego (1988).
3. Kapur, D. and Zhang, H.: A case study of the completion procedure: Proving ring commutativity problems, in J.-L. Lassez and G. Plotkin (eds), *Computational Logic: Essays in Honor of Alan Robinson*, MIT Press, Cambridge, MA, 1991, pp. 360–394.
4. Lukasiewicz, J.: *Elements of Mathematical Logic,* Pergamon Press, Oxford, 1963.
5. Lukasiewicz, J.: Investigations into the sentential calculus, in L. Borkowski (ed.), *Jan Lukasiewicz: Selected Works*, North-Holland, Amsterdam, 1970, pp. 131–152.
6. McCharen, J., Overbeek, R. A. and Wos, L.: Complexity and related enhancements for automated theorem-proving programs, *Computers and Mathematics with Applications* **2** (1976), 1–16.
7. McCune, W. W.: OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.
8. McCune, W. W.: Private communication, 1994.
9. Smith, B. T.: Reference Manual for the Environmental Theorem Prover, An Incarnation of AURA, Technical Report ANL-88-2, Argonne National Laboratory, Argonne, IL, 1988.

10. Stickel, M. E.: A case study of theorem proving by the Knuth-Bendix method: Discovering that $x^3 = x$ implies ring commutativity, in *Proc. 7th Conf. on Automated Deduction*, Lecture Notes in Computer Science **170**, Springer-Verlag, New York, 1984, pp. 248–258.
11. Veroff, R. L.: Canonicalization and Demodulation, Technical Report ANL-81-6, Argonne National Laboratory, Argonne, IL, 1981.
12. Veroff, R. L. and Wos, L.: The linked inference principle, I: The formal treatment, *J. Automated Reasoning* **8** (1992), 213–274.
13. Veroff, R. L.: An Updated Demodulation Strategy for Ring Problems, Technical Report CS94-12, Department of Computer Science, University of New Mexico, 1994.
14. Wang, T. C.: Case studies of Z-module reasoning: Proving benchmark theorems from ring theory, *J. Automated Reasoning* **3** (1987), 437–451.
15. Winker, S.: Robbins algebra: conditions that make a near-Boolean algebra Boolean, *J. Automated Reasoning* **6** (1990), 465–489.
16. Wos, L.: Meeting the challenge of fifty years of logic, *J. Automated Reasoning* **6** (1990), 213–232.
17. Wos, L.: Automated reasoning and Bledsoe's dream for the field, in R. S. Boyer (ed.), *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Kluwer Academic Publishers, Dordrecht, 1991, pp. 297–345.
18. Wos, L.: The resonance strategy, *Computers and Mathematics with Applications* **29** (1995), 133–178.