Solving Open Questions and Other Challenge Problems Using Proof Sketches *

Robert Veroff
University of New Mexico, Albuquerque, NM 87131, U.S.A.

Abstract. In this article, we describe a set of procedures and strategies for searching for proofs in logical systems based on the inference rule condensed detachment. The procedures and strategies rely on the derivation of proof sketches—sequences of formulas that are used as hints to guide the search for sound proofs. In the simplest case, a proof sketch consists of a subproof—key lemmas to prove, for example—and the proof is completed by filling in the missing steps. In the more general case, a proof sketch consists of a sequence of formulas *sufficient* to find a proof, but it may include formulas that are not provable in the current theory. We find that even in this more general case, proof sketches can provide valuable guidance in finding sound proofs. Proof sketches have been used successfully for numerous problems coming from a variety of problem areas. We have, for example, used proof sketches to find several new two-axiom systems for Boolean algebra using the Sheffer stroke.

Keywords: proof sketch, condensed detachment, hints, linked UR-resolution

1. Introduction

This special issue of the *Journal of Automated Reasoning* is devoted to logical calculi based on the inference rule condensed detachment (CD):

i(s,t)	(major premise)
r	(minor premise)

t_σ, where σ is a most general unifier for terms r and s.

In particular, the focus of the articles is on proving theorems in these logical calculi using resolution-style automated reasoning programs such as OTTER [6]. The questions considered include determining the equivalence of various axiom systems, finding proofs to specific theorems, and finding proofs with certain specified properties.

In this article, we consider how the generation and use of *proof sketches*, together with the sophisticated strategies and procedures supported by an automated reasoning program such as OTTER, can be used to find proofs to challenging theorems, including open questions. Consider, for a simple example, the problem of finding a proof

* This work was support in part by National Science Foundation grant no. CDA-9503064.

of the "$x^3 = x$ implies commutativity" theorem from ring theory [12]. A proof sketch for this theorem might consist of the following sequence of equations.

1. $6x = 0$
2. $3xy + 3yx = 0$
3. $4xy + 2yx = 0$
4. $xy = yx$

Although this may be a reasonable outline of a proof, we don't necessarily want to commit to finding a proof based on this particular outline. First, we may not know, in general, whether a proposed outline actually leads to a valid proof. Second, there may be other proofs that are easier to find or have other desirable properties. To commit to a particular proof sketch—for example, to check a "proof" in a paper submitted for publication—we could simply try to prove each of the steps of the sketch in turn. The focus of this article is on the use of proof sketches for proof finding rather than proof checking.

If we do have a proof sketch for a target theorem, we can put a bias toward the steps of the sketch in our general search for a proof. That is, if we do happen to generate any of these steps as lemmas, we want to note the event and take advantage of it as we can. Furthermore, there often are several different proofs of a theorem, and it suffices, in general, to find any one of them. If we have multiple proof sketches for a target theorem, we may consider each to provide potentially valuable guidance and can consider them simultaneously. Relevant procedures and strategies are described in Section 2 of this article.

Where do proof sketches come from? If we have some expertise in an area, we may conjecture how a proof would or could go. For especially difficult problems, we might try to gain insight by looking at variations of the original problem. For example, in order to prove a theorem about nonassociative rings, we may first prove the same theorem for associative rings and use the resulting proof for guidance. In this article, we present a number of techniques for generating proof sketches that are amenable to automation and that have proved useful for numerous challenging problems.

The general idea behind the use of proof sketches is directly related to topics that have been well studied in the literature. These include abstraction (see, for example, [4, 8]), analogy (see, for example, [2, 3]), and planning (see, for example, [1, 10]). The contribution of this article is a suite of procedures and strategies that have proven to be especially effective for the types of logic problems featured in this special issue.

The remainder of this article is organized as follows. Section 2 is a background section that describes the general context and perspective of our work. This includes overviews of some of the key automated reasoning features—most notably hints [14] and linked inference rules [13, 20]—on which the work depends. Sections 3 through 5 describe procedures and strategies that have been especially effective for generating useful proof sketches. In particular, Section 3 describes uses of paramodulation and demodulation; Section 4 describes a strategy for using goal-directed reasoning; and Section 5 describes the use of proofs found in one logic as proof sketches for related, but more restricted, logics. Section 6 contains a summary of several relevant case studies and results. Finally, Section 7 contains a brief summary and plans for future work.

2. Background

In this section, we describe the context and general objectives of the material presented in the remainder of the article. This description includes a characterization of the problems we are trying to solve and the general approach taken. We also include in this section brief overviews of two of the key features of automated reasoning programs, hints and linked inference rules, on which much of the material in the remainder of the article depends.

2.1. Problem Characterization

CD problems are easily represented for resolution theorem provers. If we let `P(t)` represent the assertion that `t` is a theorem, then applications of condensed detachment can be implemented by using hyperresolution and the following clause.

```
-P(i(x,y)) | -P(x) | P(y).
```

A CD derivation in Otter consists of a sequence of input clauses representing axioms and hyperresolvents representing applications of condensed detachment. A CD proof of a theorem T consists of a CD derivation of a clause that conflicts with a clause representing the negation of T.

The central theme of this article is the generation and use of *proof sketches* to help find CD proofs. For our purposes, a proof sketch for a theorem T is any sequence of clauses such that if we have a CD derivation of each of the clauses in the sequence, then we necessarily have a CD derivation of T.[1] Of course, any sequence that contains T is a sketch in this weak sense. The intention is to find proof sketches having steps that provide valuable guidance and direction in finding a proof. A proof sketch, for example, might consist of the key lemmas needed to prove a theorem. It is the thesis of this article that the strategic generation of proof sketches, together with strategies for effectively turning proof sketches into CD proofs, significantly enhances our ability to prove theorems.

We note that for some logical systems there are meta-level inference rules—for example, term-level substitutions—that are known to be sound in the theory. Although proofs using such inference rules may suffice for some applications, our objective in this work always is to produce a proof that relies solely on applications of condensed detachment. It should be no surprise, however, that meta-theoretical proofs can serve well as proof sketches, and, in fact, mapping such sketches to CD proofs characterizes an interesting class of problems.[2]

2.2. Hints

A proof sketch for a theorem T can be thought of as a set of conditions (lemmas to prove) sufficient for proving T. In the ideal case, a proof sketch already is a proof, and the conditions are met trivially. In the more general case, the clauses of a proof sketch identify notable milestones on the way to finding a proof. ¿From a strategic standpoint, it is desirable to recognize when we have achieved such milestones and to adapt the continued search for a proof accordingly. In particular, we wish to focus our attention on such milestone results and pursue their consequences sooner rather than later. The weighting strategy [5], in general, and the hints strategy [14], in particular, provide the desired control over the search.

The weighting strategy provides a way for the user to impose intuition, special knowledge, or preferences on a search. Weighting works by assigning a heuristic weight value to a clause; clauses with lower weight values tend to be considered earlier in the search. A clause's weight value is determined by a set of user-defined weighting patterns. The hints strategy is an enhancement to the weighting strategy. In the hints strategy, the weight of a clause is adjusted—according to

[1] Strictly speaking, the proof of T may be a proper subset of the combined CD derivations.

[2] In some cases, there may be known procedures for mapping a meta-theoretical proof to a CD proof. Our interest, however, is in the more general use of proof sketches as part of a search strategy.

user preferences—if it subsumes or is subsumed by any of a set of user-supplied hint clauses.

The hints strategy provides a natural and effective way to take full advantage of a proof sketch in the search for a proof. For example, including each clause from the proof sketch as a hint clause, and making the OTTER assignment

```
% decrease by 100 the weight of any derived
% clause that back subsumes a hint clause
assign(bsub_hint_add_wt, -100).
```

virtually ensures that when a clause is derived that back subsumes a hint clause—in particular, one of the key milestone clauses of a proof sketch—the newly generated clause will become the focus of attention (that is, chosen as the "given" clause) as soon as possible.

The use of hints is additive in the sense that hints from multiple proof sketches or from sketches for different parts of a proof can all be included at the same time. For this reason, hints are particularly valuable for "gluing" subproofs together into a single coherent OTTER proof, even when wildly different search strategies are used to find the individual subproofs.

2.3. Linked Inference Rules

Inference rules are used to deduce new information from a database of axioms and previously derived formulas. One objective of the study of inference rules is to develop new rules that take "larger" steps that are better directed. Inference rules such as hyperresolution [9] and UR-resolution [5] take larger steps than binary resolution, but even these steps are still relatively small, and they are defined solely in terms of syntactic criteria. Our research has focused on a class of inference rules, called *linked inference rules* [13, 20], that can take arbitrarily large steps and allow steps to be chosen based on semantic rather than syntactic criteria. For example, low-level deductions, although technically required in a proof, may be a significant distraction to an overall proof strategy. Linking provides a way to combine several low-level steps into a single higher-level step, avoiding the explicit consideration of uninteresting intermediate results and allowing the proof search strategy to focus on the most interesting and relevant information.

For the problems of interest in this article, linked UR-resolution has the effect of combining multiple applications of condensed detachment into a single inference step. One immediate consequence is that the use of linking provides a local permutation of the search space (compared to the use of ordinary hyperresolution), which increases the likelihood of finding previously undiscovered derivations. More notably, linking permits the automated reasoning program to make use of high-weight clauses on the way to key results, clauses that otherwise might not be considered.

3. Using Paramodulation and Demodulation to Find Proof Sketches

In the calculus of ordinary propositional logic, a principle of substitution allows the replacement of any subformula with a logically equivalent formula; the resulting formula is necessarily logically equivalent to the original. A special case of this principle of substitution allows the replacement of any tautology with the constant `T`, where `T` is a special symbol that is assigned the logical value **true** under all interpretations. An analog for the logics we study with condensed detachment might allow us to replace with the constant symbol `T` any theorem that appears as a subterm of another theorem. Although this type of substitution may not be sound in general for an arbitrary logic, the resulting formula may be—and from our experience, often is—a theorem. It may require several CD steps to derive a theorem for which a single substitution sufficed, but our objective here is to generate useful proof sketches. Furthermore, a proof sketch that is found by using such substitutions may prove valuable even if some of the individual steps of the sketch are not provable in the current theory. It isn't necessarily the case that every step of a proof sketch will appear in a final proof.

We can implement this sketch generation strategy (actually, a slightly more general version that includes instantiation through unification) as a simple application of paramodulation and demodulation. To do this, we make a minor change in representation that consists of uniformly replacing every atom of the form `P(t)`, denoting that term `t` is a theorem, with `EQ(t,T)`, denoting that term `t` is equivalent to the constant `T`.

Example 3.1. Consider the following set of clauses representing axioms, a hyperresolution nucleus for condensed detachment, and the negation of a theorem.

```
% axioms
P(i(i(x,y),i(i(y,z),i(x,z))))        # label("L1").
P(i(i(n(x),x),x))                    # label("L2").
P(i(x,i(n(x),y)))                    # label("L3").
```

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

% negation of theorem
-P(i(a,a)).
```

The new representation is as follows.

```
% axioms
EQ(i(i(x,y),i(i(y,z),i(x,z))),T)    # label("L1").
EQ(i(i(n(x),x),x),T)                # label("L2").
EQ(i(x,i(n(x),y)),T)                # label("L3").

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

% negation of theorem
-EQ(i(a,a),T).
```

The change in representation, by itself, has no direct consequence; OTTER will operate exactly as it did before the change. The new representation, however, allows us to use paramodulation as an inference rule and to define meaningful demodulators. Strategically, this permits a term-oriented manipulation of clauses that is not possible using CD alone.

With the new representation, an application of paramodulation from the left side of the "from" clause results in an instance (due to unification) of a known theorem, but with a subterm of the instance replaced with the constant `T`. It is natural, then, to include the following two demodulators to further simplify the resulting formula.

```
EQ(i(x,T),T).
EQ(i(T,x),x).
```

These simplifications emulate properties of ordinary propositional logic, where the function symbol `i` denotes logical implication. Again, although applications of such rewrite rules are not necessarily sound in all possible logics, they have proven to be part of an effective strategy for generating proof sketches (see Section 6).

The use of paramodulation just described is one part of suite of strategies and procedures for generating proof sketches and finding proofs. In order to ensure that no other operations "accidentally" corrupt the representation—for example, unintentional consequences of other theorem proving strategies we may employ—we eliminate any deduced clause that is not in the form EQ(t,T). In OTTER, we can do this by using the following weight templates for purging clauses.[3]

```
% keep an atom if it is in the proper form
weight(EQ($(1),T),1).

% else reject it
weight(EQ($(1),$(1)),9999).
```

In an extended version of the described use of paramodulation, we occasionally have used equivalences more general than those in the specific form EQ(t,T). Further, we can have OTTER generate and recognize such equivalences dynamically. Consider, for example, the clause

```
-EQ(i(x1,x2),T) | -EQ(i(x2,x1),T) | EQUIV(x1,x2).
```

This is an analog of the classical implication

$$((E_1 \to E_2) \wedge (E_2 \to E_1)) \Longrightarrow (E_1 \leftrightarrow E_2),$$

which holds for arbitrary formulas E_1 and E_2. The use of a new predicate symbol, EQUIV, in the clause allows us to distinguish context and allows us to continue to use the weighting template describe previously for deleting clauses containing undesired EQ atoms. Derived EQUIV unit clauses remain available for future applications of paramodulation.

More Aggressive Uses of Demodulation

Using the new representation, we have full access to the equality-oriented features supported by OTTER. These include, for example, designating any or all of the axioms of the current theory as input demodulators—either as simplifiers or as part of a restriction strategy—and the ability to specify the selection of certain generated theorems as new (dynamic) demodulators.

Using axioms as simplifiers is a natural extension of the paramodulation strategy. As an inference rule, applications of paramodulation generate new clauses and are under the control of the search strategy. If, on the other hand, a clause of the form EQ(t,T) is designated as a demodulator, then all instances of the term t will automatically be rewritten to the constant T. This approach can greatly simplify the set

[3] In OTTER, weighting can be used both as a search strategy and as a deletion strategy.

of clauses being managed by the automated reasoning program and can have a significant impact on the search for a proof.

Example 3.2. Axioms $L1$, $L2$, and $L3$ from Example 3.1 can be designated as input demodulators by including the following clauses on the appropriate list.

```
EQ(i(i(x,y),i(i(y,z),i(x,z))),T).   % L1
EQ(i(i(n(x),x),x),T).               % L2
EQ(i(x,i(n(x),y)),T).               % L3
```

Alternatively, rather than using input axioms as part of a somewhat aggressive simplification strategy, we can use them as part of a restriction strategy, for example, by deleting any clause that has any one of a number of selected theorems as a proper subterm.

Example 3.3. Analogous to the previous example, we can specify demodulators

```
EQ(i(i(x,y),i(i(y,z),i(x,z))),junk).   % L1
EQ(i(i(n(x),x),x),junk).               % L2
EQ(i(x,i(n(x),y)),junk).               % L3
```

along with a weighting template to delete any clauses containing the constant symbol `junk`.

We note that demodulation must be used carefully when used as part of a deletion strategy. For example, consider the demodulators

```
f(x,x) = c.
g(a,x) = junk.
g(x,b) = junk.
```

and the term `f(g(a,x),g(y,b))`. This term will be demodulated, incorrectly, to the constant `c` rather than being identified as "junk" and deleted. Adding context to the junk terms protects against this type of side effect. For example, if the demodulators

```
f(x,x) = c.
g(a,x) = junk(C1,[x]).
g(x,b) = junk(C2,[x]).
```

are used instead of those first presented, the term `f(g(a,x),g(y,b))` will be demodulated to `f(junk(C1,[x]),junk(C2,[y]))`, as it should.

We can go even further with our use of demodulation by including additional input demodulators that are designed specifically to simplify terms for the sake of facilitating the search for good proof sketches and proofs. In particular, we have found it useful to use simplifications that emulate properties of ordinary propositional logic. Perhaps the most notable example is

```
EQ(n(n(x)),x).
```

which participated in the derivation of numerous useful proof sketches.

Proof sketches that result from multiple applications of demodulation may have fairly large gaps, making it all that much more difficult to use the proof sketch to find a CD proof, even with the hints strategy. We can mitigate this difficulty significantly by including in the proof sketch every intermediate demodulant relevant to the proof sketch. That is, if a clause `D` participates in a proof, and `D` results from the generation of a clause `C`, followed by a sequence of demodulation steps resulting in the intermediate clauses $\mathtt{C} = \mathtt{C}_1$, $\mathtt{C}_2$, ..., $\mathtt{C}_n = \mathtt{D}$, then we include all of the $\mathtt{C}_i$ in the resulting proof sketch. This can be done quite easily with the `print_intermediate_demods` flag that we have implemented in our own version of OTTER.

4. Backing Up from the Denial

Strategically, it often is desirable to have the automated reasoning program reason about the denial of a theorem rather than simply using the denial as a stopping condition for a strictly forward proof. In the case of condensed detachment, the denial very often takes the form of a negative ground unit clause. Reasoning about the denial could, for example, produce new negative unit clauses that serve as new sufficient conditions for proving the original target theorem. Unfortunately, effectively backing up denials for condensed detachment problems is not done naturally. Consider, once again, the clause for condensed detachment

```
-P(i(x,y)) | -P(x) | P(y).
```

and a unit clause of the form `-P(t)` representing the negation of a theorem to be proved. Resolving the negative unit clause with the positive literal of condensed detachment results in the two-literal clause

```
-P(i(x,t)) | -P(x).
```

which we will refer to as $invCD$. There are two scenarios for generating a new negative unit clause from this clause.

Scenario 1. We can resolve the second literal of *invCD* with *any* theorem `P(t')`, resulting in the clause

```
-P(i(t',t)).
```

We note that the resolution step in this case involves a single trivial unification and that the two terms `t` and `t'` in the resulting clause necessarily have no variables in common.

Scenario 2. We can resolve the first literal of *invCD* with any theorem of the form

```
P(i(r,s)).
```

such that term `s` unifies with `t`. The result is

```
-P(r').
```

where `r'` is the appropriate instance of `r`.[4]

Although there may be some benefits to expanding the space of negative unit clauses as described in the first scenario, the exhaustive nature of the expansion is prohibitive. Furthermore, the absence of any nontrivial unification makes it unlikely that the results will be semantically interesting.

The second scenario is more effective at deriving conditions—that is, new negative unit clauses—sufficient for proving the original theorem, but it also has a tendency to derive clauses that can't possibly be provable in the current theory. Consider, for example, a negative unit clause of the form

```
-P(i(r,a)).
```

where `a` is a constant. Resolving the corresponding instance of *invCD*

```
-P(i(x,i(r,a))) | -P(x).
```

with a theorem

```
P(i(x,i(y,x)).
```

results in

```
-P(a).
```

[4] When function symbol i denotes the logical implication operator, condensed detachment is identical to the inference rule modus ponens. Under this same interpretation, the inference rule defined by Scenario 2 is identical to *modus tollens.*

Although it is easy to see that proving theorem `a` suffices to prove the original theorem, it also should be easy to see that `a` is not likely to be provable!

We observe from experience that there tends to be a proliferation of such trivial consequences of scenario two—clauses that express sufficient conditions for proving the theorem under study but do not themselves represent theorems in the current theory. Ideally, what we want is to filter the results of the second scenario. That is, we wish to keep precisely those clauses that represent theorems provable in the current theory. Of course, being able to determine this in general is as difficult as proving the original theorem itself.

Rather than giving up on the second scenario entirely, we choose to define a strategic filter, one that can be used to eliminate as many of the unprovable negative unit clauses as possible, while keeping those clauses that are potentially useful termination conditions for the current search. Our first attempt at defining such a filter is based on a simple evaluation. Specifically, we keep a clause if and only if it represents a tautology in ordinary propositional logic.

We can implement this filter with a somewhat messy but effective set of demodulators that generates the truth table for each candidate. First, we need to deduce the candidate negative unit clauses. This is easily done with the linked UR-resolution nucleus,

```
$NUCLEUS([2]) | -P(i(x,y)) | -Q(x) | P(y).
```

where `Q` is a new predicate symbol. By designating the second literal as the target (output) literal for applications of linked UR-resolution, this clause can be used to generate clauses of the form

```
-Q(t).
```

where `t` is a candidate condition for proving the original theorem.

It now remains to replace the candidate clause with

```
-P(t).
```

if and only if term `t` is accepted by the current filter. That is, we keep the result of the application of *invCD* provided our heuristic filter suggests there is a reasonably good chance that the result corresponds to a theorem in the current theory.

The first part of the filter is based on the conditional demodulator

```
G(TTeval(x)) -> Q(x) = P(x).
```

where `TTeval` is a procedure—implemented with demodulators—that takes a formula as input and returns the formula's truth table (a single column) as output, and `G` is a predicate used to accept or reject the truth table result. If the result is accepted—specifically, if the `G` predicate evaluates to the OTTER constant `$T`—then the demodulation from the candidate `Q` form to the `P` form takes place. Otherwise, the demodulation does not take place, and the `Q` form remains as the fully demodulated result. We can eliminate such results from further consideration by using the purge weight template

```
weight(Q($(1)),9999).
```

All that remains is the implementation of `TTeval` and `G`. These implementations rely on the following new function symbols.

`col(x1,...,xn)`	– represents a single column of a truth table
`COLi(x,y)`	– implication operator for truth table columns
`COLn(x)`	– negation operator for truth table columns
`I(x,y)`	– implication operator for truth values
`N(x)`	– negation operator for truth values

We illustrate the implementation of this strategy with an example defined for a denial expressed in terms of constants `p`, `q`, and `r`.

Example 4.1. Demodulators for constants `p`, `q`, and `r` appearing in the denial.

```
% implement operations in the formula as operations
% on the columns of a truth table
TTeval(i(x,y)) = COLi(TTeval(x),TTeval(y)).
TTeval(n(x)) = COLn(TTeval(x)).

% define evaluation by truth table

% truth tables for the constants
TTeval(p) = col(0,0,0,0,1,1,1,1).
TTeval(q) = col(0,0,1,1,0,0,1,1).
TTeval(r) = col(0,1,0,1,0,1,0,1).

% define operations on truth table columns

COLi(col(x1,x2,x3,x4,x5,x6,x7,x8),
     col(y1,y2,y3,y4,y5,y6,y7,y8))
   =
```

```
       col(I(x1,y1),I(x2,y2),I(x3,y3),I(x4,y4),
           I(x5,y5),I(x6,y6),I(x7,y7),I(x8,y8)).

COLn(col(x1,x2,x3,x4,x5,x6,x7,x8))
   =
       col(N(x1),N(x2),N(x3),N(x4),
           N(x5),N(x6),N(x7),N(x8)).

% define operations on truth values

I(0,0) = 1.
I(0,1) = 1.
I(1,0) = 0.
I(1,1) = 1.

N(0) = 1.
N(1) = 0.

% acceptance condition
G(col(1,1,1,1,1,1,1,1)) = $T.
```

The example filter just given accepts only negative ground unit clauses in the constants `p`, `q`, and `r`. We can define instead a slightly looser filter that permits some variables to appear in the resulting negative unit clause. For example, the conditional demodulator

```
$VAR(x) -> TTeval(x) = col(U,U,U,U,U,U,U,U).
```

permits a variable occurrence in the candidate term to be considered by the truth table evaluation. The following demodulators then permit these variables—represented by the constant `U`, meaning "unspecified"—to be evaluated in a way that is consistent with the requirement of accepting only tautologies.

```
I(0,U) = 1.
I(U,1) = 1.
```

We note that although we have a fair amount of experience, including several notable successes, with the first filter, our experience with the second filter is limited. It is easy to believe that the looser filter admits more opportunities for finding proofs, but it also is easy to believe that it could have a detrimental effect on a proof search by sending the search down too many fruitless paths. The full impact of the second filter needs to be studied further.

Once we have a proof P that relies on backed up versions of the denial, we still must convert the proof to a CD proof. This is relatively easy to do using the steps of P to define an appropriate set of hint clauses. Specifically, we include as hints all of the positive clauses of P along with the positive versions of the negative clauses appearing P. For example, if the clause

```
-P(i(a,i(b,a))).
```

appears in P, we include

```
P(i(x,i(y,x))).
```

as a hint.

5. Proving a Theorem in a Related Logic

The logics we have been studying are fairly expressive in that there often are many different proofs for the same theorem. Given this observation and the observation that some of the logics are very closely related, it seems reasonable to consider using the proof of a theorem T in one logic as a sketch for proving the same theorem in another logic.

Say a proof P of theorem T in logic L consist of the steps

$$A_1, A_2, ..., A_k, T_1, T_2, ..., T_m = T,$$

where $A_1, A_2, ..., A_k$ are axioms in L, and $T_1, T_2, ..., T_m$ are derived theorems. In what way can these proof steps, the derived steps in particular, be relevant to a proof of T in a different logic L'? For one thing, the individual T_i may or may not be theorems in L'. Further, those that are theorems in L' may or may not participate in any proof of T in L', and those that do participate in some proof may not appear together in the same proof. To summarize the possibilities:

- Proof P might map directly to a proof in L'. That is, each T_i may follow directly from the axioms of L' and the steps $T_1, T_2, ..., T_{i-1}$. This, of course, is the most desirable outcome. Such a proof would quickly be found using the theorems in P as hints.

- Some subset of the T_i may participate in a proof in L', but the proof of each relevant T_i may have to go outside of the steps given. This probably is the most typical scenario for turning a proof sketch into a proof. The ability to complete such a proof depends, of course, on the ability to fill in the gaps of the sketch. But in any case, the large proof has been reduced to a set of smaller proofs.

- Some T_i may be such that T_i itself is not a theorem in L', but a proper instance T_i' of T_i *is* a theorem in L', and T_i' suffices to prove theorem T. We can attempt to account for this possibility by using as hints not the steps of P, but the instances of these steps that suffice for P. This information is easily extracted from an OTTER derivation of P.

- None of the T_i participates in a proof of T in L', but some of the T_i are derivable in L'. This is the worst-case scenario, since the search may be thrown off track by the hints strategy if it does derive any of the T_i.

Implementing a sketch generation strategy based on these observations is straightforward. Consider, for example, the two logics TV and MV as defined by the following axioms.

Axioms for two-valued logic (TV):

```
P(i(i(x,y),i(i(y,z),i(x,z)))) # label("L1").
P(i(i(n(x),x),x))             # label("L2").
P(i(x,i(n(x),y)))             # label("L3").
```

Axioms for multiple-valued logic (MV):

```
P(i(x,i(y,x)))                # label("MV1").
P(i(i(x,y),i(i(y,z),i(x,z)))) # label("MV2").
P(i(i(i(x,y),y),i(i(y,x),x))) # label("MV3").
P(i(i(n(x),n(y)),i(y,x)))     # label("MV4").
P(i(i(i(x,y),i(y,x)),i(y,x))) # label("MV5").
```

It is known that TV implies MV. That is, every theorem in MV is a theorem in TV. When trying to prove a difficult theorem T in MV, we can turn to TV for some guidance. We can, for example, first try to prove T using one or more axiom systems for TV and then use any resulting proofs as sketches when trying to prove T in MV.

6. Summary of Results

We have tested and evaluated our proof-sketch procedures and strategies extensively with three types of condensed detachment problems:

- Prove a theorem T in a logic L defined by axioms $A_1, A_2, ..., A_n$.

- Given an axiom system $A_1, A_2, ..., A_n$ for a logic L, show that a different set of formulas, $A'_1, A'_2, ..., A'_m$ is an equivalent axiom system for L. This involves showing that each A'_i is a theorem in L and that each A_i is derivable from $A'_1, A'_2, ..., A'_m$.
- Given a meta-theoretical proof of a theorem T in a logic L—that is, a proof using inference rules in addition to condensed detachment—find a CD proof of T in L.

Many of the problems considered are from TV and MV—as defined in the preceding section—and a number of similar logics.

We have experimented with numerous problems of each type and have tried a number of different strategies for each. As a direct consequence of the proof sketch techniques described in this article, we now are able to prove theorems substantially more difficult than anything we had been able to prove previously. We also find that sets of related problems are now more likely to be amenable to an attack with a uniform strategy.

Example 6.1, Axiom Systems for TV

Several sets of formulas are known to be axiomatizations of TV. See [21] for examples and discussion. We have used the techniques described in this article to prove many of these systems from scratch. That is, starting with no prior knowledge about the proofs, we were able to generate proof sketches that in turn led to the discovery of proper CD proofs. Two of the proved systems

```
% Meredith's single axiom
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),
    i(i(v,x),i(u,x)))) # label("Meredith").

% Lukasiewicz's 23-letter single axiom
P(i(i(i(x,y),i(i(i(n(z),n(u)),v),z)),
    i(w,i(i(z,x),i(u,x))))) # label("Luka23").
```

are notable in that they appear to be especially challenging to prove with an automated reasoning system. In our work, finding proofs for the most difficult problems required the derivation of *several* proof sketches using a variety of sketch-generation strategies. The final proof in each case, however, consists of a strictly forward, CD-only proof.

Example 6.2, A Challenge Theorem from MV

The problem is to prove that

```
P(i(i(i(x,i(i(y,z),z)),i(i(y,z),w)),
    i(i(i(i(x,y),y),z),w))).
```

is a theorem in MV. This theorem is a generalization of

```
P(i(i(i(x,i(i(y,z),z)),i(i(y,z),z)),
    i(i(i(i(x,y),y),z),z))).
```

which represents a type of associativity property.

We were able to prove this theorem using a sequence of proof sketches and the techniques—paramodulation, demodulation, and backing up the denial—described in this article. The basic approach was to first prove the theorem in TV and then, starting with the axioms of TV together with the axioms of MV, to successively eliminate (from the input) the axioms of TV until only the axioms of MV remained.

The value of proof sketches is not limited to condensed detachment.

Example 6.3, Open Questions from Boolean Algebra

In 1913, Henry Sheffer [11] presented a three-axiom basis for Boolean algebra in terms of the "Sheffer stroke".

$$(x \mid x) \mid (x \mid x) \;=\; x \qquad \text{(Sheffer 1)}$$
$$x \mid (y \mid (y \mid y)) \;=\; x \mid x \qquad \text{(Sheffer 2)}$$
$$(x \mid (y \mid z)) \mid (x \mid (y \mid z)) \;=\; ((y \mid y) \mid x) \mid ((z \mid z) \mid x) \qquad \text{(Sheffer 3)}$$

More recently, a number of simplifications ("abridgements") of Sheffer's system have been published, including, for example, five systems presented in Meredith [7]. Early in 2000, Stephen Wolfram proposed a study of 27 candidate axiom systems consisting of 25 single equations and two pairs of equations [18]. Wolfram's interest in these equations arose from his research project *A New Kind of Science* [19].

We have used OTTER to identify several new two-axiom systems from the Wolfram set. See [15, 16, 17] for a summary of these results. In each case, our approach relied heavily on the use of proof sketches. The general approach is to derive a known axiom system from some sufficient set of axioms—that is, the set of current interest plus others—and then to successively eliminate the extra axioms from the input set, using all previous proofs as hints. This approach has worked remarkably well on the problems considered.

7. Summary

In this article, we presented procedures and strategies for generating and using proof sketches to find proofs for open questions and other challenging problems. The approach relies heavily on the use of hints and includes special uses of paramodulation, demodulation, linked inference rules, and a new strategy for reasoning about denials. We have substantial anecdotal evidence that the procedures and strategies are effective for condensed detachment problems. Furthermore, the use of proof sketches has led directly to the discovery of new two-axiom systems for Boolean algebra using the Sheffer stroke.

Although initially somewhat ad hoc in nature, the applications of the new procedures and strategies have become increasingly regular and systematic. We have already begun to implement tools to facilitate the execution of appropriate *sequences* of OTTER experiments. One of our goals is to have a comprehensive strategy that is fully automated.

References

1. Barker-Plummer, D., "Gazing: An Approach to the Problem of Definition and Lemma Use," *J. Automated Reasoning* **8**(3), 1992, 311–344.
2. Bledsoe, W., The Use of Analogy in Proof Discovery, MCC Tech. Report AI-2158-86, Microelectronics and Computer Technology Corporation, Austin, Texas, 1986.
3. Brock, B., Cooper, S., and Pierce, W., "Analogical Reasoning and Proof Discovery," E. Lusk and R. Overbeek, (eds.), *Proc. of the 9th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 310, Springer-Verlag, 1988, 454–468.
4. Giunchiglia, F., and Walsh T., "A Theory of Abstraction," *Artificial Intelligence* **57**(2,3), 1992, 323–389.
5. McCharen, J., Overbeek, R., and Wos, L., "Complexity and Related Enhancements for Automated Theorem-proving Programs," *Computers and Mathematics with Applications* **2**, 1976, 1–16.
6. McCune, W., OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.
7. Meredith, C. "Equational Postulates for the Sheffer Stroke," *Notre Dame J. of Formal Logic*, vol. 10 (1969), pp. 266–270.
8. Plaisted, D., "Abstraction Mappings in Mechanical Theorem Proving," W. Bibel and R. Kowalski, (eds.), *Proc. of the 5th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 87, Springer-Verlag, 1980, 264–280.
9. Robinson, J., "Automatic Deduction with Hyper-resolution," *International J. of Computer Mathematics* **1**, 1965, 227–234.
10. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence* **5**, 1974, 115–135.

11. Sheffer, H., "A Set of Five Independent Postulates for Boolean Algebras, with Application to Logical Constants," *Trans. American Mathematical Society*, vol. 14 (1913), pp. 481–488.
12. Veroff, R., Canonicalization and Demodulation, Technical Report ANL-81/6, Argonne National Laboratory, Argonne, Illinois, 1981.
13. Veroff, R., and Wos, L., "The Linked Inference Principle, I: The Formal Treatment," *J. Automated Reasoning* **8**(2), 1992, 213–274.
14. Veroff, R., "Using Hints to Increase the Effectiveness of an Automated Reasoning Program: Case Studies," *J. Automated Reasoning* **16**(3), 1996, 223–239.
15. Veroff, R., Axiom Systems for Boolean Algebra Using the Sheffer Stroke, Technical Report TR-CS-20, Computer Science Department, University of New Mexico, Albuquerque, New Mexico, 2000.
16. Veroff, R., Short 2-Bases for Boolean Algebra in Terms of the Sheffer Stroke, Technical Report TR-CS-25, Computer Science Department, University of New Mexico, Albuquerque, New Mexico, 2000.
17. Veroff, R., `http://www.cs.unm.edu/~veroff/BA/`, 2000.
18. Wolfram, S., Correspondence by electronic mail, February 4, 2000.
19. Wolfram, S., A new kind of science. `http://wolframscience.com`, 2000.
20. Wos, L., Veroff, R., Smith, B., and McCune, W., "The Linked Inference Principle, II: The User's Viewpoint", in R. Shostak (ed.), *Proc. of the 7th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 170, Springer-Verlag, 1984, 316–332.
21. Wos, L., "Automated Reasoning and Bledsoe's Dream for the Field," in R. Boyer (ed.), *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Kluwer Academic Publishers, Dordrecht, 1991, 297–345.