Deciding Combinations of Theories

ROBERT E. SHOSTAK

SRI International, Menlo Park, California

Abstract. A method is given for deciding formulas in combinations of unquantified first-order theories. Rather than coupling separate decision procedures for the contributing theories, the method makes use of a single, uniform procedure that minimizes the code needed to accommodate each additional theory. It is applicable to theories whose semantics can be encoded within a certain class of purely equational canonical form theories that is closed under combination. Examples are given from the equational theories of integer and real arithmetic, a subtheory of monadic set theory, the theory of *cons*, *car*, and *cdr*, and others. A discussion of the speed performance of the procedure and a proof of the theorem that underlies its completeness are also given. The procedure has been used extensively as the deductive core of a system for program specification and verification.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Program Verification—*correctness proofs*; *verification*; F 3 1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*mechanical verification*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*mechanical theorem proving*

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Decision procedures

1. *Introduction*

Much attention has been given in the last few years to the problem of developing decision procedures for applications in program verification and mechanical proof of theorems in mathematics. It often happens, in these applications, that the formulas to be proved involve diverse semantic constructs. Verification conditions for programs, for example, often mix arithmetic with data structure semantics. Accordingly, a number of procedures have been developed (such as those of Jefferson [2] and the author [7]) for particular *mixes* of theories. One of the most general results along these lines is given by Nelson and Oppen [4] in their work on simplification using cooperating decision procedures. The central idea, here, is that procedures for diverse unquantified theories in a certain class can be hooked together so as collectively to decide a formula involving constructs from each. The coupled procedures communicate by passing back and forth information they individually derive about equality among terms.

The method described in this paper also addresses the problem of deciding combinations of theories. Rather than connecting separate procedures for the constituent theories, however, it is based on a single, uniform procedure that

This research was supported in part by the National Science Foundation under Grant No. MCS 79-04081 and by the Air Force Office of Scientific Research under Contract No. F49620-79-C-0099.

Author's address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025

localizes the mechanism particular to each constituent to a relatively small amount of code. As a result, new semantic constructs can be incorporated quite easily. The method also obviates the explicit computation of entailed equality information required by the distributed approach, and the redundant representation of this information in the separate procedures.

The method is applicable to theories whose semantics can be expressed inside a certain class of purely equational canonical form theories. The class has the property that the canonizing functions of arbitrary members can be combined to obtain a canonizing function for their union. The decision procedure extends the congruence closure methods [1, 5, 6] that have been used for the pure theory of equality. The extension depends upon a notion of *semantic signature* that generalizes the earlier concept of signature to accommodate the semantics of the canonical form theories.

The method is useful for a surprisingly rich class of constructs. We give examples from integer and real arithmetic, the theory of recursive data structures, a subtheory of monadic set theory, and the first-order theory of equality. The procedure has been used intensively in a major verification effort [8] over the course of several months, and has been found to give good speed performance.

The next two sections characterize the class of theories to which the method is applicable, and prove that it is closed under combination. Section 4 gives the decision procedure and exemplifies its operation. Sections 5 and 6 discuss performance issues and an extension of the method that treats nonconvex theories. The last section gives a proof of the theorem on which the procedure is based.

2. σ-*Theories and Algebraic Solvability*

The procedure described in this paper operates over a subclass of certain unquantified first-order equational theories called σ-theories. Such theories have both *interpreted* and *uninterpreted* function symbols. Terms whose outermost function symbol is interpreted are themselves said to be interpreted, and all other terms (including variables) are said to be uninterpreted. The distinguishing feature of σ-theories is that each has an effective canonizer, i.e., a computable function σ from terms to terms such that a purely interpreted (i.e., containing no uninterpreted function symbols) equation $t = u$ is valid in the theory iff $\sigma(t) = \sigma(u)$. The canonizer σ must act as an identity on uninterpreted terms, must be idempotent (i.e., $\sigma(\sigma(t)) = \sigma(t)$), and must have the property that the arguments of each interpreted canonical term must themselves be canonical.

The semantics of σ-theories are defined with respect to σ-*interpretations*, i.e., Herbrand interpretations that respect σ in the sense that for each interpreted term $t = f(t_1, \ldots, t_n)$, $v(t) = \sigma(f(v(t_1), \ldots, v(t_n)))$, where $v(t)$ is the value of t in the interpretation. Note that these semantics say more than that two terms must be equal in all interpretations if they have identical canonical forms; they also say that certain terms (for example, distinct canonical constants) must be *unequal* in all interpretations.

A simple example of a σ-theory is the equational theory of addition and multiplication by constants over the reals. Each term in this theory can, of course, be reduced to a linear expression of the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n + c$ $(n \geq 0)$, where each a_i is a nonzero real constant, each x_i is a variable, and c is a real constant. By specifying some criterion for ordering the x_i's (alphabetic ordering, for example), removing unity coefficients, and discarding c if $c = 0$, a suitable σ can be defined. Note that the theory can trivially be enriched with uninterpreted function symbols.

The decision procedure we will describe operates on σ-theories with a special property we call *algebraic solvability* Such theories have a *solver*—a computable function that takes a purely interpreted equation (no uninterpreted symbols other than variables) e(say $t = u$) as input, and returns either *true*, *false*, or a conjunction of equations. The returned formula must be equivalent to e and must be either *true* or *false* if e has no variables. Moreover, the equations (if any) in the returned formula must all be of the form $x_i = t_i$, where the x_i's are distinct variables of t (or u if t has none) that do not occur in any of the t_i's.

Returning to the example of real linear arithmetic, a suitable solver is trivially provided by conventional algebraic manipulation; solving $3x + 2y = 2x + 4$, for example, gives $x = -2y + 4$.

At first glance, one might expect that algebraic solvability is such a strong criterion that there are no nontrivial theories other than, say, theories over fields, that satisfy it. Surprisingly, this is not so. As the following examples suggest, algebraic solvability is characteristic of many constructs one encounters in practice.

Example 1. *The convex theory of cons, car, and cdr.* First consider the theory of *cons*, *car*, and *cdr* with the following axiomitization:

$$\begin{aligned} car(cons(x, y)) &= x, \\ cdr(cons(x, y)) &= y, \\ cons(car(x), cdr(x)) &= x. \end{aligned}$$

Using the first two of these axioms as rewrite rules, one can always reduce a term in the theory to a canonical form in which *cons* is not an argument of either *car* or *cdr*.

A *solve* function for this theory is given by the algorithm shown in Figure 1. Note that the algorithm introduces new (existentially quantified) variables. The chief function of the main routine *solve* is to grind the left side of the given equation (represented as an ordered pair) down to a variable; the auxiliary function *solve*1 (which takes two terms and returns a term) takes over from there, and attempts to eliminate that variable from the right side. The routine *resolve*, called by *solve*, sees to it that no equations in the returned conjunction have the same left side.

Consider, for example, the action of the algorithm on the equation

$$cons(car(x), cdr(car(y))) = cdr(cons(y, x)).$$

The two sides are first canonized to obtain

$$cons(car(x), cdr(car(y))) = x.$$

This equation is then reduced to the two equations

$$car(x) = car(x),\ cdr(car(y)) = cdr(x).$$

The first of these reduces to *true* and is, in effect, discarded. The second becomes

$$car(y) = cons(a, cdr(x)),$$

where a is a new variable, and then (recursing again)

$$y = cons(cons(a, cdr(x)), d),$$

which is then returned.

It is not difficult to show that the algorithm satisfies the criteria for a solver. The reader might find it amusing to prove that it always terminates.

```
solve(⟨t, u⟩)
  t ← σ(t);
  u ← σ(u);
  if t = u return true;
  if t contains no variable swap t and u;
  if t still contains no variable return false;
  let a and d be fresh variables (new on each call);
  if t is of the form:
    car(t1) then return solve(⟨t1, cons(u, d)⟩);
    cdr(t1) then return solve(⟨t1, cons(a, u)⟩);
    cons(t1, t2) then
      return resolve(solve(⟨t1, car(u)⟩) ∧ solve(⟨t2, cdr(u)⟩));
  else return ⟨t, solve1(t, u)⟩;
solve1(x, u)
  u ← σ(u);
  if x = u or x does not occur in u return u;
  if x occurs in u as an argument of cons or if u is not a cons
    return from solve with false;
let a, d be fresh variables (new on each call);
  replace car(x) by a and cdr(x) by d in u;
  return cons(solve1(a, car(u)), solve1(d, cdr(u));
resolve(c)
  until c contains no two equations of the form v = t1, v = t2 do
    remove v = t2 from c; c ← c ∧ solve(⟨t1, t2⟩);
  return c;
```

FIG. 1. Solver for *car*, *cdr*, and *cons*.

The familiar nonconvex theory of *cons*, *car*, and *cdr*, with the distinguished element *nil* and the additional axiom

$$nil \neq cons(x, y)$$

can also be treated, as discussed in Section 6.

The more general theory of recursive data structures can be handled using a straightforward generalization of σ and the *solve* algorithm just given. □

Example 2. *Integer linear arithmetic.* The σ described earlier for real linear arithmetic can be used here as well. A *solve* function is provided by a method for reducing linear Diophantine equations that is based on the Euclidean algorithm. For example, the equation

$$17x - 49y = 30$$

is solved by ths method to obtain

$$x = 49z - 4, \qquad y = 17z - 2,$$

where z is a new variable. The theory of linear congruences can be treated in a similar way. □

Example 3. *"Venn diagram" monadic set theory.* Formulas in this theory are constructed from set variables and the empty set under set inclusion, union, intersection, and complement. As is well known, such formulas can be encoded in a purely equational theory with only set variables, the empty set, and the union operator +. The encoding represents each set variable in the given formula by the union of its disjoint constituent pieces in the Venn diagram corresponding to the variables of the formula. Referring to Figure 2, for example,

$$A \cap B = B \cup C$$

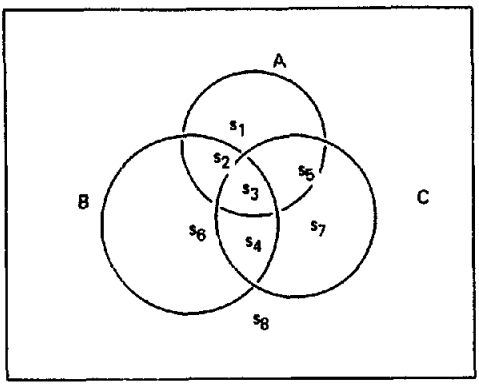

FIG. 2. Example from "Venn diagram" theory.

becomes

$$s_2 + s_3 = (s_2 + s_3 + s_4 + s_6) + (s_3 + s_4 + s_5 + s_7).$$

Similarly, $A \subseteq B$ becomes $s_1 + s_5 = \Phi$.

It is easy to see that terms in the encoding theory can be canonized simply by ordering the set variables in some way and eliminating redundancies.

A solver for the encoding theory is also easy to construct. One simply replaces each side of an equation with its σ-form and cancels variables occurring on both sides. If the two sides thereby become identical, *true* is returned; otherwise, the conjunction

$$\bigwedge v_i = \Phi.$$

is returned, where the v_i's are the remaining variables. □

Example 4. *The unquantified first-order theory of equality.* This theory has no interpreted function symbols at all. σ is therefore just the identity function. The *solve* function merely returns *true* if the given equation is an identity, and acts as an identity otherwise. □

Until now we have been concerned with solve functions for equations with only purely interpreted terms. In the next section and in the procedure itself it will be necessary to deal with equations that may have uninterpreted terms other than variables. The notion of solver is extended to deal with such equations in the following way.

We will say that a *solver* for a σ-theory T is a computable function that takes an arbitrary equation e(say $t = u$) over T as input and returns either *true*, *false*, or a conjunction of equations. The returned formula must be equivalent to e and must be one of *true* or *false* if e contains no uninterpreted terms. Moreover, the equations (if any) in the returned formula must all be of the form $u_i = t_i$, where the u_i's are distinct uninterpreted subterms of t(or u if t has none) that do not occur in the t_i's *except possibly as proper subterms of uninterpreted terms.*

Note that all the solvers we have talked about so far can be trivially extended to satisfy the new definition simply by treating the maximally uninterpreted subterms of a submitted equation as if they were distinct variables. For example, the equation

$$f(x) = g(x + 2, f(x)) = 0,$$

when solved in linear arithmetic, produces

$$f(x) = -1 \times g(x + 2, f(x)).$$

Note that although $f(x)$ occurs on the right-hand side of the result, it occurs only as a proper subterm of an uninterpreted term.

3. *Combining Theories*

Algebraically solvable σ-theories have the elegant and useful property that they can be combined (provided they are compatible in a certain sense) without giving up algebraic solvability.

One needs to be careful, of course, about what it *means* to combine theories. Simply to say that the interpreted symbols of the combined theory are the union of those of the constituents does not suffice. The semantics of mixed terms (terms with symbols from different constituents) must be explicated, and conflicts that may arise when the constituents share interpreted symbols must be resolved.

The most natural resolution (and the one we espouse) of the issue of mixed terms is simply to assume that interpreted symbols from different theories have no semantic interaction in the combined theory; if a function interpreted in T_1 but not in T_2 is given an argument that is interpreted in T_2 but not in T_1, the effect is to treat the argument *as if it were uninterpreted.* For example, in the combination of real linear arithmetic with the theory of *cons*, *car*, and *cdr*, we will have

$$\sigma(car(cons(1 + 1, nil))) = 1 + 1.$$

We will consider that + is *uninterpreted with respect to cons, car, and cdr,* (*and vice versa*) in the combined theory, and that $1 + 1$ is therefore an uninterpreted subterm of $car(cons(1 + 1, nil))$.

It still remains to resolve the question of what happens if the constituent theories share interpreted symbols. Here we will take the conservative stand of requiring that the shared symbols have exactly the same semantics. More precisely, we will say that T_1 and T_2 are *compatible* if for each term t in their combination whose function symbol is interpreted in both, $\sigma_1(t) = \sigma_2(t)$. We will consider only combinations of compatible theories.

Note that as a consequence of the view that uninterpretedness is relative, the purely interpreted terms of the combined theory are exactly the union of those from the constituent theories; the action of the combined σ on each such term is the action of the σ of the appropriate constituent. It is easy to check that the combined σ thus defined satisfies the requirements for a canonizer.

Given solvers for each of the constituent theories, one can be constructed for the combination in the following way. Let T be the combination of $T_1, \ldots, T_r$, and let $t = u$ be an equation of T to be solved. If neither t nor u contains uninterpreted terms, we return *true* if they have identical canonical forms and *false* if not. If t has no uninterpreted subterms, t and u are exchanged. $solve_i$ is now applied to $t = u$, where T_i is a constituent theory from which the function symbol of t derives. Each equality in the resulting conjunction is then recursively replaced by its solution until no left side of an equality occurs in a right side except as a proper subterm of an uninterpreted term.

The idea is easily understood in the context of an example. Suppose the equation to be solved is

$$5 + car(x + 1) = cdr(x + 2) + 3.$$

First solving in linear arithmetic, we obtain

$$car(x + 1) = cdr(x + 2) - 2.$$

Note that whereas $car(x + 1)$ is (relatively) uninterpreted in the original equation, it is interpreted here. The solver for *car*, *cdr*, and *cons* is now applied to obtain

$$x + 1 = cons(cdr(x + 2) - 2, d),$$

where d is a new variable. Solving once more in linear arithmetic, we have

$$x = cons(cdr(x + 2) - 2, d) - 1.$$

As the left side x is an uninterpreted term that does not occur in the right side except as a proper subterm of an uninterpreted term $(x + 2)$, we are done.

4. *The Procedure*

We now give a refutation procedure for conjunctions of equalities and disequalities in an algebraically solvable σ-theory T. Formulas with arbitrary propositional structure can be decided for validity by placing their negations into disjunctive normal form and applying the refutation procedure to each conjunction; the original formula is valid iff all conjunctions are unsatisfiable.

The procedure extends the congruence closure methods of Nelson and Oppen [5], Downey et al. [1], and the author [6] for deciding purely uninterpreted formulas. These methods all depend on computing a congruence relation that provides a canonical form for each term represented in the relation. The procedure we describe can be viewed as combining this canonial form for purely uninterpreted terms with the σ-form for interpreted terms to obtain a universal canonical form for all terms. Crucial to the extension is a generalization of the notion of *signature* that permits the two kinds of canonical forms to be merged.

The main algorithms of the procedure are *process* and *canon*, shown in Figure 3. Algorithm *process* is called on the set of equalities (each represented by an ordered pair) occurring in the conjunction S whose satisfiability is to be determined; if *process* returns *unsatisfiable*, the procedure halts with that conclusion. Otherwise, for each disequality $t \neq u$ of S, $canon(t)$ and $canon(u)$ are computed. If $canon(t) = canon(u)$ for some such disequality, *unsatisfiable* is returned; otherwise, *satisfiable* is returned.

As in the procedure described by Nelson and Oppen, a congruence relation is developed (by *process*) using Tarjan's *union* and *find* primitives [9]. The *canon* routine returns the canonical representation, in this relation, of a term to which it is applied. Note from the specification of *canon* that two terms have the same *canon* if they have the same signature. If a term is already represented in the relation (i.e., *canon* has previously been called on a term with the same signature), the existing canonical form is returned; otherwise, the signature of the term is returned as its *canon*.

The *solve* routine is assumed to return a set (representing a conjunction). In accordance with the definition given in the last section, the set must either be a singleton containing one of *true*, *false*, or a set of equations whose left-hand sides are uninterpreted. The right-hand sides are assumed to be reduced with respect to σ. Note that *process* uses the convention that e_1 refers to the left side of an equation e, and that e_2 refers to the right side. The order of sides is important.

Note also that the *union* function (called in *merge*) is assumed always to change *find* of its first argument. Initially, $find(t)$ is assumed to be t for each term t.

Aside from the tree structure used to implement *union* and *find*, two global data structures are used: *use* and *sig*. For each canonical term t, $use(t)$ is a list of terms whose signatures have t as an argument. The list $use(t)$ is maintained so that the

```
process(eqset)
  while eqset ≠ ∅ do
    select and delete e from eqset;
    s ← solve(canon(e1) = canon(e2));   /* e1 refers to the left side of e, e2 to the right side */
    while s ≠ ∅ do
      select and delete e from s;
      if e = true continue while;
      if e = false return unsatisfiable;
      merge(e1, e2),
merge(t1, t2)
  t1 ← find(t1); t2 ← find(t2);
  if t1 = t2 return;
  union(t1, t2)   /* find(t1) is now t2 */
  for u ∈ use(t1) do
    if u is uninterpreted
      replace t1 with t2 in the argument list of sig(u);
      for v ∈ use(t2) when sig(v) = sig(u) do
        remove v from use(t2);
        s ← s ∪ solve(find(u) = find(v));
      add u to use(t2);
    else if find(u) ≠ u
      obtain u' from u by replacing t1 with t2 in the argument list;
      add u = canonsig(σ(u')) to s;
canon(t)
  return(canonsig(signature(t)));
canonsig(w)
  if w is a constant return find(w);
  else
    say w = f(w1, ..., wn);
    if w is interpreted replace each wi with canonsig(wi);
    if w = sig(u) for some u ∈ use(w1) return find(u);
    else
      for i from 1 to n do add w to use(wi);
      sig(w) ← w;
      use(w) ← ∅;
      return(w);
signature(t)
  if t is a constant return t;
  else return σ(f(canon(t1), ..., canon(tn))) where t = f(t1, ..., tn);
```

FIG 3. Main routines of procedure.

signatures of its members are distinct and include the signatures of all represented terms whose signatures have t as an argument. For each u in $use(t)$, $sig(u)$ gives the current signature of u.

Figure 4 shows various stages in the application of the procedure to the conjunction

$$z = f(x - y) \wedge x = z + y \wedge -y \neq -(x - f(f(z)))$$

in the combination of the theory of real linear arithmetic and the pure theory of equality. The solid arrows in each diagram represent the *union–find* structure (for example, $find(z) = f(x + -1 \times y)$ in Figure 4a. The dotted arrows represent the *use* structure; a dotted arrow from t to u indicates that u is a member of $use(t)$.

The procedure first calls *process* on the equalities of the conjunction. The equality $z = f(x - y)$ is selected, *canonized* (giving $z = f(x + -1 \times y)$), and submitted to *solve*. The *canonized* equality is its own solution, and so $merge(z, f(x + -1 \times y))$

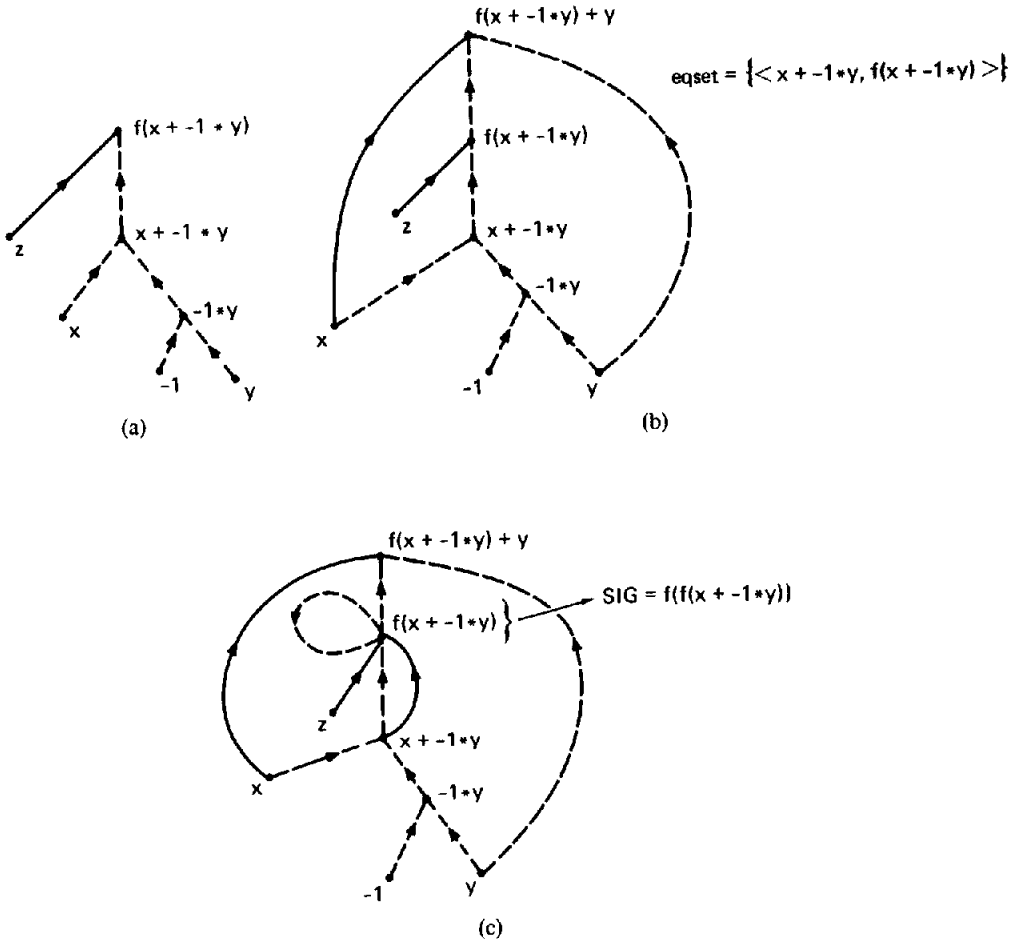

FIG. 4. Application of the procedure to $z = f(x - y) \wedge x = z + y \wedge -y \neq -(x - f(f(z)))$.

is invoked. Figure 4a shows the state after this call. The dotted structure results from the call of *canon* on $f(x - y)$.

The second equality $x = z + y$ is next selected from *eqset*, *canonized* to obtain $x = f(x + -1 \times y) + y$, and submitted to *solve*. Again *solve* has no effect, and *merge*$(x, f(x + -1 \times y) + y)$ is called. Note that inside the call, *canonsig* is invoked on $x + -1 \times y$, yielding (as the reader should check) $f(x + -1 \times y)$.

The equation

$$x + -1 \times y = f(x + -1 \times y)$$

is then added to s. Figure 4b depicts the situation after this call.

Now *merge*$(x + -1 \times y, f(x + -1 \times y))$ is invoked as a result of the addition to s. The effects of this call are shown in Figure 4c. Note that $f(x + -1 \times y)$ is added to its own *use* list, and that *sig*$(f(x + -1 \times y))$ is updated to reflect the new signature of $f(x + -1 \times y)$, namely $f(f(x + -1 \times y))$. No new pairs are added to s, so that call to *process* is complete.

The procedure next computes *canon* of $-y$ and $-(x - f(f(z)))$. Both calls produce $-1 \times y$, so the conjunction is unsatisfiable.

5. *Performance Issues*

The procedure has been implemented in INTERLISP-10, and forms the deductive core of a specification and verification system that provides mechanical support for an extension of multisorted first-order logic [8]. This system has been used intensively over the last year in a successful design proof effort for the SIFT [3,10] fault-tolerant operating system. The effort involved the proof of literally hundreds of formulas. We found that formulas occupying several pages of syntax would usually be proved in less than a minute (provided they were true—otherwise this could take substantially longer!).

The time complexity of the procedure naturally depends on the cost of invoking the algorithms for σ and *solve*. Since these costs typically tend to dominate, it is impossible to analyze the performance of the algorithm without making specific assumptions about the particular combination of theories to which it is applied. In the purely uninterpreted case, the procedure becomes very similar to the *E*-procedure mentioned earlier, whose worst-case time was determined by Nelson and Oppen to be $O(m^2)$, where m is the number of edges in the graph of the developed congruence relation. The differences are that *use* lists are not kept sorted in our algorithm and that we use a special data structure *sig* to store the current signatures of terms that represent signature classes. We have found that because *use* lists are usually quite short, the time required to maintain sorted versions is not worth the theoretical advantage in doing so. We have also found that the use of the *sig* data structure provides a significant practical improvement, because it eliminates having to compute signatures in the second **for** loop of the *merge* routine.

We should point out that analytic measures can be quite misleading indicators of performance in real applications. Our experience has been that such measures are often less informative than more precise and objective characterizations of performance, such as "blindingly fast."

6. *Extension to Nonconvex Theories*

The theories we have used to illustrate the procedure are all *convex* in the sense of [4]. The notion of algebraic solvability can be straightforwardly generalized to apply to nonconvex theories by permitting the equalities returned by a solver to be embedded in arbitrary propositional structure as opposed to strict conjunctions. The nonconvex theory of *cons*, *car*, *cdr*, and *nil*, for example, can be treated by modifying the *solve* of Figure 1 to return an expression of the form

$$\textbf{if } u = \mathit{nil} \textbf{ then } \mathit{false} \text{ else } \ldots$$

in the case where t is of the form $\mathit{cons}(t_1, t_2)$. The *process* procedure given in Figure 3 must be modified to break the propositional structure returned by the solver back down into disjunctive normal form.

7. *Correctness of the Procedure*

It is easy to check by inspection that the procedure is sound, i.e., that two terms with the same *canon* must be equal in any σ-interpretation for S. This section gives a proof of the theorem that underlies the procedure's completeness. The theorem extends the result given in [6] for the pure theory of equality to σ-theories.

Definition. We say that a congruence relation $\simeq$ over a set $\mathscr{R}$ of terms in a σ-theory T is a *σ-congruence relation* if each equivalence class $[t]$ in $\simeq$ has a

representative $\rho(t)$ such that

(*) For each interpreted term $t = f(t_1, \ldots, t_n)$ in T,

$$\rho(t) = \sigma(f(\rho(t_1), \ldots, \rho(t_n))).$$

Note that, interpreting the *canon* procedure as ρ and the set of represented terms as $\mathscr{R}$, the (*)-property just says that the **canon** of each interpreted term t must be the semantic signature of t.

THEOREM. *Let S be a σ-unsatisfiable set of equalities and disequalities over a σ-theory T, and let $\simeq$ be a σ-congruence relation over a set $\mathscr{R}$ of terms in T such that $t \simeq u$ for each equation $t = u \in S$. Then there exists a disequality $t \neq u \in S$ such that $t \simeq u$.*

PROOF. We will suppose that there is no such disequality, and construct a σ-model for S. First, divide the Herbrand Universe of T into layers $H_0, H_1, \ldots$, where $H_0 = \mathscr{R}$, and

$$H_{i+1} = H_i \cup \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H_i\}$$

for each function symbol f. We say that the *$\mathscr{R}$-height* of a term t is the smallest i for which $t \in H_i$.

The model $\mathscr{M}$ is constructed inductively by height. For terms t of height 0, we let $\mathscr{M}(t) = \rho(t)$. For terms $t = f(t_1, \ldots, t_n)$, $n \geq 0$, of height $i > 0$, we let

$$\mathscr{M}(t) = \begin{cases} \mathscr{M}(v), & \text{if there exists a term } v = f(v_1, \ldots, v_n) \text{ of height } < i \text{ such that } \mathscr{M}(v_j) = \mathscr{M}(t_j), \\ & 1 \leq j \leq n. \\ \sigma(f(\mathscr{M}(t_1), \ldots, \mathscr{M}(t_n))) & \text{otherwise.} \end{cases}$$

It must be shown that $\mathscr{M}$ is functionally reflexive, that is,

$$\bigwedge \mathscr{M}(t_i) = \mathscr{M}(u_i) \supset \mathscr{M}(f(t_1, \ldots, t_n)) = \mathscr{M}(f(u_1, \ldots, u_n)),$$

and that $\mathscr{M}$ respects σ. Both these properties are straightforwardly proved by induction on $\mathscr{R}$-height. The first follows directly from the manner in which the model is constructed, the second from the (*)-property. □

ACKNOWLEDGMENTS. The author is grateful to D. Hare, P. M. Melliar-Smith and Richard Schwartz, all of whom helped to correct lacunae in the procedure.

REFERENCES

1. DOWNEY, P.J., SETHI, R., AND TARJAN, R. Variations on the common subexpression problem. *J. ACM 27*, 4 (Oct. 1980), 758–771.
2. JEFFERSON, D.R. Type reduction and program verification. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, Pa., Aug. 1978.
3. MELLIAR-SMITH, P.M., AND SCHWARTZ, R.L. Formal specification and mechanical verification of SIFT. A fault-tolerant flight control system. *IEEE Trans Comput 31*, 7 (July 1982), 616–630.
4. NELSON, G., AND OPPEN, D.C. Simplification by cooperating decision procedures. *ACM Trans. Prog Lang Syst 1*, 2 (Oct. 1979), 245–257.
5. NELSON, G., AND OPPEN, D C. Fast decision procedures based on congruence closure. *J. ACM, 27*, 2 (Apr 1980), 356–364.
6. SHOSTAK, R.E. An algorithm for reasoning about equality. *Commun. ACM 21*, 7 (July 1978), 583–585.

7. SHOSTAK, R.E. A practical decision procedure for arithmetic with function symbols. *J. ACM 26*, 2 (Apr. 1979), 351–360.
8. SHOSTAK, R.E., SCHWARTZ, R., AND MELLIAR-SMITH, P.M. STP: A mechanized logic for specification and verification. In *Proc. 6th Conf. on Automated Deduction*, Lecture Notes in Computer Science, 138, D.W. Loveland, Ed., Springer Verlag, New York, pp. 32–49.
9. TARJAN, R.E. Efficiency of a good but not linear set union algorithm. *J. ACM 22*, 2 (Apr. 1975), 215–225.
10. WENSLEY, J., et al., SIFT. Design and analysis of a fault-tolerant computer for aircraft control. *Proc IEEE 66*, 10 (Oct. 1978).

RECEIVED APRIL 1982; REVISED FEBRUARY 1983; ACCEPTED FEBRUARY 1983