On Shostak's Decision Procedure for Combinations of Theories[⋆]

To be presented at CADE '96

David Cyrluk, Patrick Lincoln, and Natarajan Shankar

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
{cyrluk, lincoln, shankar}@csl.sri.com
Phone: +1 (415) 859-{2560, 5454, 5272} Fax: +1 (415) 859-2844

Abstract. Decision procedures are increasingly being employed for deciding or simplifying propositional combinations of ground equalities involving uninterpreted function symbols, linear arithmetic, arrays, and other theories. Two approaches for constructing decision procedures for combinations of ground theories were pioneered in the late seventies. In the approach of Nelson and Oppen, decision procedures for two disjoint theories are combined by introducing variables to name subterms and iteratively propagating any deduced equalities between variables from one theory to another. Shostak employs a different approach that works far more efficiently in practice. He uses an optimized implementation of the congruence closure procedure for ground equality over uninterpreted function symbols to combine theories that are *canonizable* and *algebraically solvable*. Many useful theories have these properties. Shostak's algorithm is subtle and complex and his description of this procedure is lacking in rigor. We present, for the first time, a careful development and clarification of Shostak's procedure that corrects several mistakes in Shostak's original presentation. Our analysis serves as a useful basis for the implementation, extension, and further optimization of Shostak's decision procedure.

1 Motivation

Consider the valid ground formula:

$$x = y \wedge f(f(f(x))) = f(x) \supset f(f(f(f(f(y))))) = f(x)$$

where the variables are assumed to be implicitly universally quantified and f is an uninterpreted function symbol. Formulas of this sort often arise as subgoals

[⋆] Supported by by NSF Grant CCR-930044, ONR Contract No. N00014-95-C-0168, NSF Grant CCR-9224858, and ARPA contract A721. We are indebted to Rob Shostak for the original ideas studied in this paper. We thank John Rushby, Sam Owre, Deepak Kapur, Harald Ruess, Oliver Möller, and Tomas Uribe, and the anonymous referees for valuable feedback and comments.

in theorem proving applied to program verification, particularly in systems such as Ehdm [2], Eves [4], Nqthm [1], Ontic [9], PVS [14], SDVS [3], and the Stanford Pascal Verifier [8]. In the mid-to-late seventies, Downey, Sethi, and Tarjan [6], Kozen [7], Nelson and Oppen [11], and Shostak [17] gave efficient algorithms for deciding such formulas by computing the *congruence closure* relation on the graph representing the terms in the formula.

Although necessary, it is not sufficient to have decision procedures for uninterpreted equality, since many subgoal formulas in typical proofs involve both uninterpreted function symbols (like f above) and interpreted function symbols such as addition, multiplication, recursive datatype operations, array operations, etc. As an example, consider the formula:

$$f(f(i-j)) = j \wedge i = 2*j \supset f(f(f(f(\mathit{select}(\mathit{update}(a, 2*j, j), i))))) = j,$$

where $\mathit{update}(a, j, 0)$ updates array a at index j to have the value 0, and $\mathit{select}(a, i)$ selects the array element from a at index i. Given decision procedures for the component theories (such as linear arithmetic and arrays), Nelson and Oppen [10] gave a technique for combining such decision procedures to decide the combination of these theories by simply propagating equalities between the different decision procedures. The Nelson-Oppen procedure is used in Eves [4], the Stanford Pascal Verifier [8], and SDVS [3].

Shostak [18] used a different approach that merges the simplifiers for individual theories into a single procedure based on congruence closure. Shostak's decision procedure is at the core of systems such as Ehdm [2] and PVS [13]. In practice, Shostak's procedure is more efficient than that of Nelson and Oppen (see Crocker [5]). Despite its significance, a rigorous analysis of Shostak's procedure has been lacking. Further, all published accounts of Shostak's method are flawed: the most often-cited paper [18] contains two important flaws in the description of the algorithm, while other sources, such as technical reports, contain the same as well as other flaws. In no published work we are aware of has an accurate description been given of Shostak's algorithm, and no accurate account exists of the limitations of Shostak's approach. The complex reasoning supporting the correctness of Shostak's procedure has also never been published.

This paper attempts to remedy the situation by providing the first correct description of Shostak's algorithm, along with the key invariants, lemmas, and theorems that demonstrate the correctness of Shostak's approach. We also accurately describe the limitations of Shostak's method. We start with a naive congruence closure procedure and show how this can be systematically optimized and augmented to yield a corrected version of Shostak's procedure.

The primary contribution of this paper is a rigorous understanding of Shostak's decision procedure where the flaws in his original description have been eliminated. Additionally, we sketch correctness arguments that can be used to construct proof objects from successful runs of the decision procedure. We hope that our analysis will make it easier for others to implement, adapt, and extend Shostak's ideas.

2 Congruence Closure

This section contains background material. The only new material here is the proof-theoretic justification for congruence closure based on a cut-elimination argument.

The use of *congruence closure* for deciding quantifier-free or ground equational theories plays a central role in Shostak's combined decision procedure. As an example, consider the theory T given by the equations $\{w = g(f(a)), f(a) = a\}$. To show that $w = g(a)$ follows in theory T, we can construct a congruence closure graph whose nodes correspond to the terms and subterms of the term set $\{w, g(f(a)), g(a), a\}$ such that the node corresponding to $g(f(a))$ has label g and the successor node corresponding to $f(a)$. The congruence closure relation relates (by merging into the same equivalence class) those nodes corresponding to the given equations so that w and $g(f(a))$ are merged, as are a and $f(a)$. Furthermore, any two nodes with identical labels whose corresponding successor nodes are in the same equivalence class are merged. The merging of a and $f(a)$ thus induces the merging of $g(a)$ and $g(f(a))$. As a result, w and $g(a)$ are in the same equivalence class.

More formally, let T be a collection of equations between terms formed from variables x_i and n-ary function symbols f_i^n. A model $M = \langle D, |.|\rangle$ consists of a domain D, a mapping $|.|_\rho$ from terms to D such that $|x_i|_\rho = \rho(x_i) \in D$, and $|f_i^n| \in D^n \to D$. Let $|f_i^n(a_1, \ldots, a_n)|$ abbreviate $|f_i^n|(|a_1|, \ldots, |a_n|)$. M is a model for T if for each $a_i = b_i$ in T, $|a_i|_\rho = |b_i|_\rho$.

A complete set of proof rules for equivalence can be given by the rules of reflexivity, symmetry, and transitivity. A complete set of proof rules for ground equational logic can be obtained by adding the Leibniz rule or substitutivity: derive $f_i^n(a_1, \ldots, a_n) = f_i^n(b_1, \ldots, b_n)$ from $a_1 = b_1, \ldots, a_n = b_n$. We say that T deduces $a = b$ or $T \vdash a = b$ if there is a proof from the equations in T using the rules of reflexivity, symmetry, transitivity, and substitutivity.

Following the terminology of Nelson and Oppen [11], let $G = (V, E)$ be a labelled directed graph where $\lambda(v)$ and $\delta(v)$ give the label and out-degree, respectively, of vertex v in V. Let $v[i]$ (the i'th successor of v) be the vertex u such that (v, u) is the ith edge with source v. If α is a set of vertices, let $use(\alpha)$ be the set $\{v \in V | (\exists i{:} v[i] \in \alpha)\}$. Given a binary relation R on V, the equivalence closure of R (represented by R^*) is the reflexive, symmetric, and transitive closure of R. The congruence closure $\hat{R}$ of a binary relation R is the least extension of R^* such that for any u, v where $\lambda(u) = \lambda(v)$, $\delta(u) = \delta(v)$, and for each i, $1 \leq i \leq \delta(u)$, $u[i]\ \hat{R}\ v[i]$, we have $u\hat{R}v$.

Now, let S be a set of terms that is closed under subterms. We can represent S by a graph $G = (V, E)$ such that each term x_i is a leaf node (i.e., has no successors), and each term $f_i^n(a_1, \ldots, a_n)$ is represented by a node v such that $\lambda(v) = f_i$, $\delta(v) = n$, and the nodes $v[1], \ldots, v[n]$ represent the terms $a_1, \ldots, a_n$, respectively. If G is a graph representing S, and a is a term in S, let $\nu(a)$ be the vertex in G representing a. If T is a ground equational theory of the form

$a_1 = b_1, \ldots, a_n = b_n$, such that each a_i and each b_i is in S, then let R_T be the binary relation on S containing the pairs $(\nu(a_i), \nu(b_i))$ and closed under reflexivity, symmetry, and transitivity.

Theorem 1. *Let T be a ground equational theory, S be a set of terms closed under subterms and containing the terms in T, and let $G = (V, E)$ represent S. Then for terms a, b in S, we have $T \models a = b$ iff $\nu(a)\ \hat{R}_T\ \nu(b)$.*

Proof. The 'only-if' direction is easy since the collection of equivalence classes given by the equivalence relation $\hat{R}_T$ forms a model satisfying T and hence $a = b$. For the 'if' direction, if $T \not\models a = b$ then there is some model satisfying T but not $a = b$. Equality in this model induces a congruence-closed relation on the terms in S, which by minimality must contain $\hat{R}_T$. Thus, it must be the case that $\nu(a)\ \hat{R}_T\ \nu(b)$ also does not hold. ∎

Since we are interested, at least in principle, in constructing proof objects corresponding to successful proofs, we would like to also give a proof-theoretic argument for soundness and completeness. This proof-theoretic argument requires the following cut-elimination result which we state without proof.

Theorem 2. *If there is a proof of $a = b$ from theory T using reflexivity, symmetry, transitivity, and substitutivity, then there is a proof of $a = b$ from T where all applications of transitivity are restricted to the case where the cut term is a left or right-hand side of an equality in T.*

Theorem 3 below is the proof-theoretic analogue of Theorem 1.

Theorem 3. *Let T be a ground equational theory, S be a set of terms closed under subterms and containing the terms in T, and let $G = (V, E)$ represent S. Then for terms a, b in S, we have $T \vdash a = b$ iff $\nu(a)\ \hat{R}_T\ \nu(b)$.*

Proof. The 'if' direction is an easy induction on the rules used to show that $\nu a\ \hat{R}_T\ \nu b$, since these rule applications correspond directly to proof steps in the ground equational logic.

The 'only if' direction is somewhat harder since we need to show that no equality proof of terms in S need use cut terms in transitivity steps that fall outside of S. We can use the cut elimination result of Theorem 2 to restrict our attention to proofs where the all the cut terms appear as left or right-hand side equalities in T and hence in S. The result then follows by a straightforward induction on the given proof of $a = b$ from T and the definition of congruence closure. ∎

The pseudocode for a naive congruence closure procedure due to Nelson and Oppen is shown below. The *Makuse* operation is not defined: it initializes the *use* table to reflect the terms in T. The main loop *NO* processes a list of equations T by applying the procedure *Merge* which uses Tarjan's union/find primitives to maintain equivalence classes so that *union* merges two equivalence classes and *find* returns the canonical representative of an equivalence class. The workhorse of the procedure is the *Merge* operation which merges two equivalence classes and propagates the merging to any parent nodes of these two equivalence classes that become congruent as a result.

```
Merge(u, v) =
  LET $P_u = \bigcup\{use(u') | find(u') = find(u)\}$,
      $P_v = \bigcup\{use(v') | find(v') = find(v)\}$
    IN  union(u, v);
        FOR x IN $P_u$ DO
          FOR y IN $P_v$ DO
            IF find(x) $\neq$ find(y) AND Congruent(x, y)
              THEN Merge(x, y)
            ENDIF


Congruent(u, v) =
(     delta(u) = delta(v)
  AND (FORALL i: 1 <= i <= delta(u): find(u[i]) = find(v[i])))


NO(T) =
  Makeuse(T);
  CASES T OF
    nil : RETURN
    [a = b : T′] : NO(T′);
                   IF find(a) = find(b)
                      THEN RETURN
                      ELSE Merge(a, b)
                   ENDIF;
  ENDCASES
```

We state the theorem but omit the proof that *NO* constructs a congruence closed collection of equivalence classes of terms (see Nelson-Oppen [10]). Let $find_T$ represent the $find$ operation following $NO(T)$.

Theorem 4. *(correctness of NO) Given a theory T, a term universe S that is closed under subterms and contains all the terms in T, then for any terms $a, b \in S$, $find_T(a) = find_T(b)$ iff $\nu(a)\ \hat{R}_T\ \nu(b)$.*

It is easy to see that the procedure terminates since the number of equivalence classes decreases with each call to *Merge*.

3 Optimized Congruence Closure

Shostak's version of the congruence closure algorithm optimizes the naive algorithm in three ways. The first two optimizations shown below are not significant and are fairly standard.

1. The computation of the sets of the terms Pu and Pv in *Merge* is optimized by preserving the invariant $use(u) \subseteq use(find(u))$. This is a somewhat trivial optimization but it has the useful consequence that any invocation of $Merge(x, y)$ in the congruence closure procedure above can be replaced with $Merge(find(x), find(y))$. We assume that the above naive procedure has already been thus optimized.
2. The computation of *Congruent* is optimized by maintaining a data-structure $sig(u)$ for each term u that preserves the invariant that $sig(f(u_1, \ldots, u_n)) = f(find(u_1), \ldots, find(u_n))$. It is easy to check that the Shostak procedure shown below preserves this invariant.
3. The last optimization is to eliminate the term universe S and the precomputation of the *use* structure. This means that the term universe is not predetermined and the algorithm can handle equalities in an "on-the-fly" manner. This optimization is significant. Since the term universe is open-ended, it is not the case that $find_{Sh}(u) = find_{NO}(u)$ when both procedures are run in parallel. To compensate for this lack of foreknowledge regarding the term universe, Shostak introduced an operation called *canon* which returns a canonical form of a newly presented term using the canonical forms of the subterms given by *find*.

The pseudocode for Shostak's procedure is shown below. The procedure $Sh(T)$ processes each equality in the given theory T using *Merge* and builds up a congruence closure structure in terms of the data structures *find*, *use*, and *sig*. The procedure applies *canon* to both sides of an equality to obtain their normal forms based on the known equalities. In the case of a previously processed term t, *canonsig* ensures that its normal form is just $find(t)$. This is because a previously processed term t given as argument to *canonsig* is either atomic, in which case $find(t)$ is returned even if t is previously unprocessed, or t is of the form $f(t_1, \ldots, t_n)$, and equal to the $sig(u)$ (since each t_i has been normalized) for some u in $use(t_1)$.[2] *Merge* is then invoked on the resulting normal forms to merge the equivalence classes of the two terms, and then successively merge equivalence classes of any congruent parent nodes where the *sig* table is used give a fast test for congruence.

```
Sh(T) =
  CASES T OF
    nil : RETURN,
    [a = b, T′] : Sh(T′);
                  Merge( canon(a), canon(b))
    ENDCASES


Merge(a, b) =
  UNLESS a = b DO
    union(a, b);
    FOR u IN use(a) DO
      replace a by b in the argument list of sig(u);
      FOR v IN use(b) WHEN sig(v) = sig(u) DO
```

[2] There is nothing sacred about t_1 and any other t_i would work equally well.

```
        Merge(find(u), find(v));
      use(b) := use(b) $\cup$ {u}

canon(t) =
  CASES t OF
    f(t$_1$,...,t$_n$): canonsig(f(canon(t$_1$),...,canon(t$_n$))),
    ELSE: canonsig(t)
  ENDCASES

canonsig(t) =
  CASES t OF
    f(t$_1$,...,t$_n$):
        IF (FORSOME u IN use(t$_1$): t = sig(u))
           THEN RETURN find(u)
           ELSE FOR i FROM 1 to n DO use(t$_i$):= use(t$_i$) $\cup$ {t};
                sig(t) := t;
                use(t) := {};
                RETURN t
        ENDIF
    ELSE: find(t);
  ENDCASES;
```

To illustrate the difference between the naive congruence closure procedure and that of Shostak, consider a term universe S of the form $\{x, y, z, f(x), f(y)\}$ and a list of equations of the form $[x = y, z = f(x)]$. In both variants, once the equality $z = f(x)$ has been processed, we have $find_{NO}(z) = find_{Sh}(z) = f(x)$. In the naive version, after $x = y$ has been processed, $find_{NO}(x) = y$ and $find_{NO}(f(x)) = f(y)$, whereas in Shostak's version, $find_{Sh}(f(x)) = f(x)$ since $f(y)$ is not explicitly present in any of the equalities processed thus far. In Shostak's variant, the *canon* operation can cope with previously unobserved terms so that $canon(f(x)) = f(y)$. This does not mean that $canon(t) = find_{NO}(t)$ since when the second equality is processed, $canon(z) = find_{Sh}(z) = f(x)$, whereas $find_{NO}(z) = f(y)$. Conversely, $canon(f(z)) = f(f(x))$, whereas $find_{NO}(f(z)) = f(z)$ since $f(z)$ is not in the given term universe. It is however the case that for terms in the term universe S, *canon* and $find_{NO}$ maintain the same equivalence classes, so that $find_{NO}(a) = find_{NO}(b)$ iff $canon(a) = canon(b)$.

We assert the crucial invariants on the main loops of the procedures NO and Sh that relate these two procedures so that the correctness of Sh follows from that of NO. Let $congruent_{Sh}(u, v)$ be the same as the naive version of *congruent* but defined to use $find_{Sh}$ instead of $find_{NO}$. Let $find^T_{Sh}(t)$, $use^T(t)$, $canonsig^T(t)$, $canon^T(t)$ represent the result of $find(t)$, $use(t)$, $canonsig(t)$, $canon(t)$, respectively, following $Sh(T)$, and similarly for $find^T_{NO}(t)$. Recall that the naive version of congruence closure procedure has already been optimized so that $use(t) \subseteq use(find(t))$.

Invariant 1 $sig(f(t_1, \ldots, t_n)) = f(find(t_1), \ldots, find(t_n))$

Invariant 2 $sig(a) = sig(b) \supset Congruent_{Sh}(a, b)$.

Invariant 3 *$use(a) \subseteq use(find(a))$.*

Let *canon∗* be the same as *canon* but without the side-effects of updating the *sig* and *use* data structures.

Invariant 4 *canon∗ is idempotent: $canon{*}(canon{*}(t)) = canon{*}(t)$.*

Invariant 5 *If $f(a_1, \ldots, a_n) \in use^T(t)$ for some t, then*

1. *$canon{*}^T(f(a_1, \ldots, a_n)) = find^T_{Sh}(f(a_1, \ldots, a_n))$*
2. *there is a term $f(b_1, \ldots, b_n)$ in T such that $canon{*}^T(b_i) = canon{*}^T(a_i)$ for $1 \le i \le n$.*

Invariant 6

$$canon{*}^T(f(t_1, \ldots, t_n)) = canon{*}^T(f(canon{*}^T(t_1), \ldots, canon{*}^T(t_n)))$$

Invariant 7 *For any terms a, b in the term universe S of NO, $canon{*}^T(a) = canon{*}^T(b)$ iff $find^T_{NO}(a) = find^T_{NO}(b)$.*

Proof. The proof is by induction on the length of T. In the base case when T is empty, $canon{*}(a) = a = find_{NO}(a)$ and $canon{*}(b) = b = find_{NO}(b)$.

If the induction hypothesis holds of T' then for any terms a, b in S, $canon{*}^{T'}(a) = canon{*}^{T'}(b)$ iff $find^{T'}_{NO}(a) = find^{T'}_{NO}(b)$.

We show that when the new equation $u = v$ is processed, that any *Merge* in Sh corresponds to a *Merge* in NO. For the 'if' direction, we can then show by induction on the size of a that whenever $Merge_{NO}(find_{NO}(a), find_{NO}(b))$ is invoked, $canon{*}^T(a) = canon{*}^T(b)$. This certainly holds for atomic a since this is only possible in the initial call to $Merge_{NO}$. When a is of the form $f(a_1, \ldots, a_n)$, either this is the initial call to $Merge_{NO}$, in which case the conclusion follows easily, or b must be of the form $f(b_1, \ldots, b_n)$. Since $Congruent_{NO}(a, b)$, by the induction hypothesis and Invariant 6 above applied to a and b, we have $canon{*}(a) = canon{*}(b)$.

In the 'only if' direction, whenever $Merge_{Sh}(canon{*}(a), canon{*}(b))$ holds, we need to show that $find^T_{NO}(a) = find^T_{NO}(b)$. The argument is similar to the 'if' case but is not as straightforward since the analogue of assertion (6) for $find_{NO}$ may not hold even for terms in the universe S since $f(t_1, \ldots, t_n)$ can be in S but $f(find_{NO}(t_1), \ldots, find_{NO}(t_n))$ might not be in S. However, by Invariant 5.2, we can restrict our attention to a and b occurring in T. The case of the initial call to $Merge_{Sh}$ is direct. For the other (recursive) calls, we have by the induction hypothesis that since $canon{*}(a_i) = canon{*}(b_i)$ for $1 \le i \le n$, that $Congruent_{NO}(a, b)$ holds. The conclusion $find_{NO}(a) = find_{NO}(b)$ follows from the invariant of NO that for a, b in S

$$Congruent^T_{NO}(a, b) \supset find^T_{NO}(a) = find^T_{NO}(b).$$

■

4 Combinations of Theories

The Nelson-Oppen approach to combining two decision procedures is to have each theory rename any uninterpreted subterms by variables, and to iteratively propagate any newly deduced equalities between variables across theories [10]. The main efficiency drawback to Nelson-Oppen's approach is that each theory has much of the same notion of equality resulting in duplicated effort.

Shostak [18] takes a different approach by restricting his attention to canonizable and algebraically solvable theories. A theory is *canonizable* if there is a canonizer σ such that for any pure equality $a = b$ is provable in the theory (i.e., $\vdash a = b$) if and only if $\sigma(a) = \sigma(b)$. A theory is *algebraically solvable* if any equality $a = b$ can be rewritten in an equivalent *solved* form $\bigwedge_i x_i = t_i$, where each x_i occurs in $a = b$ but in none of the t_j. Shostak shows how real and integer linear arithmetic, the convex theory of lists, and monadic set theory are some examples of canonizable and algebraically solvable theories.

Shostak [18] also shows how the canonizers and solvers for several pairwise disjoint theories can be combined into a single canonizer and solver for the combined theory. In contrast, the Nelson-Oppen procedure yields a method for combining arbitrary decision procedures for disjoint equational theories whereas Shostak's procedure can merely decide the combination of disjoint canonizable and algebraically solvable equational theories by combining solvers and canonizers. This paper does not discuss the details of this combination but with the decision procedure given that such canonizers and solvers can be constructed.

Shostak's method centralizes equality reasoning in one procedure: the congruence closure algorithm described in Section 3. The interpreted theories communicate through a semantic canonizer and a solver. For the purpose of illustration, we will restrict our attention to the combination of the theories of equality over uninterpreted functions symbols and linear arithmetic equations over the reals.

Shostak's procedure operates over σ-theories, namely those theories for which there is a computable canonizer function σ from terms to terms such that the following conditions hold[3].

1. An equation $t = u$ in the theory is valid iff $\sigma(t) = \sigma(u)$
2. If t is a term not in the theory then $\sigma(t) = t$
3. $\sigma(\sigma(t)) = \sigma(t)$
4. If $\sigma(t) = f(t_1, \ldots, t_n)$ for a term, t, in the theory then $\sigma(t_i) = t_i$ for $1 \leq i \leq n$
5. $\sigma(t)$ does not contain any variables that are not already in t.

The role that σ has in the interpreted procedure is as the semantic or interpreted analog to $find$ in the uninterpreted congruence closure algorithm. That is, in the purely uninterpreted case $find$ returns the canonical representative of a term given the uninterpreted equalites already processed. In the purely interpreted case σ returns the canonical representative of a term in the terms

[3] Much of the theory and notation in this section comes from [18]

interpreted theory. Shostak's procedure is a method for combining these two canonical forms.

For example, a canonizer for linear arithmetic could be constructed by transforming such expressions (using associativity, commutativity, distributivity) into the form $a_1x_1 + \ldots + a_nx_n + c$ where each a_i is a nonzero constant and the summands are arranged in some canonical order. In the presence of uninterpreted function symbols, each arithmetic term obtained by replacing uninterpreted subterms (i.e., subterms where the topmost function symbol is uninterpreted) by variables, must be in canonical form.

To construct a decision procedure for equality in a σ-theory Shostak requires that a σ-theory have the additional property of algebraic solvability. A σ-theory is algebraically solvable if there exists a computable function *solve*, that takes an equation e and returns either **true**, **false**, or a conjunction of equations and has the following properties: (In the following let *solve*$(e) = E$.)

1. E and e are equivalent, i.e., any σ-model satisfying E satisfies e, and any σ-model of e can be extended to one satisfying E (since E can contain new variables not appearing in e).
2. $E \in$\{**true**, **false**\} or $E = \bigwedge_i x_i = t_i$
3. If e contains no variables then $E \in$\{**true**, **false**\}.
4. If $E = \bigwedge_i (x_i = t_i)$ then the following hold:
   (a) x_i occurs in e,
   (b) for all i, j, x_i does not occur in t_j,
   (c) for all $i \neq j$: $x_i \neq x_j$ and
   (d) $\sigma(t_i) = t_i$.

For example, a solver for real linear arithmetic takes an equation of the form $a_1 * x_1 + \ldots + a_n * x_n + c = b_1 * x_1 + \ldots b_n * x_n + d$ and returns $x_1 = ((b_2 - a_2)/(a_1 - b_1)) * x_2 + \ldots + ((b_n - a_n)/(a_1 - b_1)) * x_n + (d - c)$.

The purpose of *solve* is two-fold. First it detects unsatisfiability in a σ-theory by returning **false**. Second, it makes it possible to propagate equalities by augmenting the *canon* operation in Section 3 so that σ is used on any interpreted subterms and (as before) a canonical (interpreted or uninterpreted) representative of the equivalence class is used for uninterpreted subterms. For example, if we want to deduce $f(x - y) = f(f(z))$ from $y = f(z)$ and $x - 2 * y = 0$, then the solver when invoked on the latter two equations returns $y = f(z)$ and $x = 2 * y$ so that $canon(x - y) = \sigma(2 * f(z) - f(z)) = f(z)$, and $canon(f(x - y)) = f(f(z))$.

The set of equations will have a model iff *solve* never returns **false** for a processed equation. Thus whenever an equation is equivalent to **false** *solve* must report the contradiction.

We now give pseudocode for Shostak's procedure for combining decision procedures [18] with the flaws corrected. The main differences with the procedure in Section 3 are that:

1. In *Sh*, *Process1* is used to apply *Merge* to a list of pairs of terms returned by *solve*.
2. In *Process1*, equalities are solved using *solve* before *Merge* is invoked. The extra *canonsig* applied to the arguments of *Merge* is in order to update the *use/sig* structures following the *solve*, and to apply *find* to return the current canonical form. Indeed, *canonsig*(a) in *Process1* could be replaced by $find(a)$ since such an a is always either a variable or an already canonized uninterpreted term.
3. In $Merge(a, b)$, the uninterpreted case is the same as before except that *solve* is applied to $find(u) = find(v)$ before *Merge* is recursively invoked (since $find(u)$ for uninterpreted u can be interpreted).
   When u in *use*(a) is interpreted and $find(u) = u$ (i.e., u was canonical prior to the current *Merge*), then it is merged with the new canonical term *canonsig*$(\sigma(u'))$. The main reason for merging representative interpreted terms with their canonical forms is to propagate the merging to any uninterpreted superterms (to preserve Invariant 5 for uninterpreted terms). The merging of interpreted terms also preserves the invariant that uninterpreted terms u in *use*(t) appear in *use*$(find(t))$. We can thus observe that congruence closure is preserved for uninterpreted terms so that for any two uninterpreted terms a, b occurring in the *use* structure that are equal from the equations processed so far, $find(a) = find(b)$ holds. This fact is significant since *canon* uses $find$ on terms in the *use* structure to return the canonical representative of uninterpreted terms. Note that interpreted terms u in *use*(a) are not added to the *use*(b) since it is sufficient for *use*(b) to contain only the canonical term *canonsig*$(\sigma(u'))$ and this is ensured by *canonsig*.
4. The definition of *canon* is modified to apply σ to the input to *canonsig*.
5. The definition of *canonsig* is modified to recursively apply *canonsig* to the arguments when an interpreted term is given as input. This is because such a term could have been newly created by σ and the *sig* and *use* data structures must be updated to reflect this new term. In the end, the combination of *canon*, *signature*, and *canonsig* returns the canonical representative of a given term where the largest uninterpreted subterms t have been replaced by $find(t)$ and interpreted subterms t (in the original term or in replacement of uninterpreted s by $find(s)$) have been replaced by $\sigma(t)$, and the *sig* and *use* structures have been properly initialized. The use of the solved form for interpreted equalities thus makes it possible to unambiguously apply $find$ for interpreted terms and σ for interpreted terms.

```
Sh(T) =
  CASES T OF
    nil : RETURN,
    [a = b, T′] :
      Sh(T′);
      Process1(solve(canon(a) = canon(b)))
  ENDCASES
```

```
Process1(S) =
  FOR e in S DO
    CASES e OF
      false: RETURN unsatisfiable,
      a = b: Merge(canonsig(a), canonsig(b))
    ENDCASES

Merge(a, b) =
  UNLESS a = b DO
    union(a, b);
    FOR u IN use(a) DO
      IF u is uninterpreted
        THEN replace a with b in the argument list of sig(u);
             FOR v IN use(b) WHEN sig(v) = sig(u) DO
                 Process1(solve(find(u) = find(v)));
             use(b) := use(b) $\cup$ {u}
      ELSIF find(u) = u THEN
             u′ := u with b for a in its argument list;
             Merge(u, canonsig($\sigma$(u′)));
      ENDIF

canon(t) = canonsig(signature(t))

canonsig(w) =
  IF w is atomic
    THEN RETURN find($w$);
    ELSE
      LET $f(w_1, \ldots, w_n) = w$ IN
        IF w is interpreted, replace each $w_i$ with canonsig($w_i$);
        IF w = sig(u) for some u $\in$ use($w_1$)
          THEN RETURN find(u)
          ELSE
            FOR i FROM 1 TO n DO add $w$ to use($w_i$);
            sig(w) := w;
            use(w) := {};
            RETURN w
        ENDIF
  ENDIF

signature(t) =
  IF t is a constant
    THEN RETURN t
    ELSE RETURN $\sigma(f(canon(t_1), \ldots, canon(t_n)))$
          where $t = f(t_1, \ldots, t_n)$
  ENDIF
```

To illustrate how the procedure works, let us consider a list of equations: $f(x) = 4, f(2*y-x) = 3, x = f(2*x-y), 4*x = 2*x+2*y$. We first process $4*x = 2*x+2*y$, and since both sides are already in canonical form, we solve to obtain $x = y$ and these two terms are therefore merged. This merging is (needlessly) propagated to the terms $2 * x$ and $4 * x$. We then process $x = f(2*x-y)$, which following canonization is $y = f(y)$. This is left unchanged by *solve* and hence y and $f(y)$ are merged. (Note that it is okay for the variable y to occur as a proper subterm of an uninterpreted term such as $f(y)$ on the right-hand side.) Now when $f(2 * y - x) = 3$ is processed, the canonical form on the left is $f(f(y))$ which is merged with 3. When $f(x) = 4$ is processed, the canonical form of the left-hand side is 3 because $canon(x)$ is $f(y)$, $sig(f(f(y)))$ is $f(f(y))$ so $canon(f(x))$ is 3 so that $solve(3 = 4)$ returns **false**.

For soundness, we need to show the following invariant.

Invariant 8 *(soundness)* $canon*^T(a) = canon*^T(b) \supset T \vdash a = b$.

Proof. The proof is by induction on the list of equations in T. The base case when T is empty is easy. Let T be of the form $[a = b, T']$. First note that by the induction hypothesis, *canon* (and hence *canon*∗) preserves equality in T'. The initial invocation of $Merge$ is on the solved form of the given equation $a = b$, and we have restricted *solve* to return equations that are conservative with respect to the input equation. Since $Merge$ is the only operation that affects *canon*∗, we show that the soundness property is preserved whenever $Merge$ is recursively invoked. We note the invariant that for uninterpreted a, b, it is the case that $sig(a) = sig(b)$ implies $Congruent(a, b)$, and hence by the induction hypothesis, $T \vdash a = b$.

When $Merge$ is invoked on a, b where a is interpreted, b is simply a canonized form of a so that $T \vdash a = b$ follows easily from the induction hypothesis. ∎

Invariant 9 *(completeness) If* $T \vdash a = b$*, then either* $Sh(T) =$ `unsatisfiable` *or* $canon*(a) = canon*(b)$.

Proof. Given a set of equations T, the algorithm simultaneously constructs the *solve* and congruence closure $\overline{T}$ of T by ensuring that if $\overline{T}$ contains $u = v$ for interpreted u, then it also includes $solve(u = v)$. Since *canon*∗ returns a canonical representative of the equivalence class in $\overline{T}$, thus for any equality $u = v$ in $\overline{T}$, it is the case that $canon*^T(u) = canon*^T(v)$. If $\overline{T}$ does not contain **false**, then it can be shown that proofs of $T \vdash a = b$ can be constructed from $\overline{T}$ using reflexivity, symmetry, transitivity, substitutivity, and the axiom scheme: $\vdash \sigma(t) = t$. The result then follows easily by induction on the derivation of $\overline{T} \vdash a = b$. To establish the base case for the axiom scheme $\vdash \sigma(t) = t$, note that $\sigma(f(t_1, \ldots, t_n)) = \sigma(f(\sigma(t_1), \ldots, \sigma(t_n)))$, and hence we have $canon*^T(\sigma(a)) = canon*^T(a)$. ∎

Discrepancies in Shostak's Original Procedure. There are two actual bugs in the pseudocode presented in the paper [18]. The most egregious is in the procedure *Merge*. The condition $find(u) \neq u$ should actually be $find(u) = u$. Shostak's earlier technical report [19] does not contain this bug but has other discrepancies. The second bug is that the output to *solve* needs to be canonized just in case *solve* creates new variables and new terms that need to be introduced into the *use* data structure.

Shostak [18] states that the *use* data structure is maintained so that the *sig*'s are unique. This is incorrect and such an "optimization" can result in an incompleteness. The solver for *cons*, *car*, and *cdr* given by Shostak is also buggy.

Performance Comparisons. Crocker [5] has carried out an empirical comparison of Shostak's algorithm as implemented by Shostak himself against Nelson and Oppen's implementation of their procedure. Crocker's results indicate a considerable efficiency advantage for Shostak's approach.

We have performed a preliminary study of the efficiency of Nelson-Oppen approach and the Shostak approach to cooperating decision procedures, and found the Shostak approach to be about an order of magnitude faster than Nelson-Oppen. This holds over a wide class of examples including propositional, arithmetic, and equality reasoning. These results match those obtained by Crocker [2, 5, 8].

5 Conclusion

The addition of decision procedures to automatic and semi-automatic theorem provers and proof checkers has clear advantages. Most of the major proof checking systems available today contain implementations of decision procedures for propositional logic, equality, and linear arithmetic. Although the efficiency of the individual procedures is important, the method of combination of procedures is more crucial to the overall efficiency of a system than the efficiency of any one component.

Shostak's approach to the combination of decision procedures is dramatically (10x) more efficient than the earlier Nelson-Oppen method. The order of magnitude difference stems from a tighter integration of the cooperating decision procedures and from improved algorithms which save work by considering smaller term universes. All comparisons we are aware of between Shostak's and other approaches have found a dramatic speed advantage for Shostak, but have been at a loss to explain them.

In future work, we will explore the detailed implementation tradeoffs for individual theories, and develop a combination of the Shostak and Nelson-Oppen procedures. The latter combination would use a Nelson-Oppen outermost loop with an entire Shostak-style combination of procedures as one element. The advantage of this combination of procedures is that some theories are not algebraically solvable, but are still of computational interest (e.g., the theory of potentially circular lists). Adding such theories to Nelson-Oppen is simple, but adding them to Shostak is impossible. Thus we intend to use the tighter Shostak approach for all theories which can provide efficient solvers and canonizers, and use the Nelson-Oppen approach for any remaining theories. We have also developed a solver for the important case of linear arithmetic inequalities. Shostak's original implementation treated these as an external decision procedure (in the Nelson-Oppen style) and the integration of linear arithmetic inequalities was therefore not clean. In future work we also hope to describe in some detail the myriad uses made of our decision procedures in the implementation of the state-of-the-art proof checker PVS [13].

References

1. R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Academic Press, New York, NY, 1988.
2. *User Guide for the* EHDM *Specification Language and Verification System, Version 6.1.* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. Three volumes.
3. Jeffrey V. Cook, Ivan V. Filippenko, Beth H. Levy, Leo G. Marcus, and Telis K. Menas. Formal computer verification in the state delta verification system (SDVS). In *AIAA Computing in Aerospace VIII*, pages 77–87, Baltimore, MD, October 1991. AIAA paper 91-3715.
4. Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Bill Pase, and Mark Saaltink. EVES: An overview. In Prehn and Toetenel [15], pages 389–405.
5. S. Crocker. Comparison of Shostak's and Oppen's solvers. Unpublished manuscript, 1988.
6. P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpressions problem. *Journal of the ACM*, 27(4):758–771, October 1980.
7. D. Kozen. Complexity of finitely represented algebras. In *Proc. 9th ACM STOC*, pages 164–177, 1988.
8. D. C. Luckham, S. M. German, F. W. von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis. Stanford Pascal Verifier user manual. CSD Report STAN-CS-79-731, Stanford University, Stanford, CA, March 1979.
9. David A. McAllester. *ONTIC: A Knowledge Representation System for Mathematics.* MIT Press, Cambridge, MA, 1989.
10. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.
11. G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.
12. Greg Nelson. Techniques for program verification. Technical Report csl-81-10, Xerox Palo Alto Research Center, Palo Alto, CA, June 1981.
13. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
14. Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

15. S. Prehn and W. J. Toetenel, editors. *VDM '91: Formal Software Development Methods*, volume 551 of *Lecture Notes in Computer Science*, Noordwijkerhout, The Netherlands, October 1991. Springer-Verlag. Volume 1: Conference Contributions.
16. John Rushby and Friedrich von Henke. Formal verification of the Interactive Convergence clock synchronization algorithm using EHDM. Technical Report SRI-CSL-89-3R, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1989 (Revised August 1991). Original version also available as NASA Contractor Report 4239, June 1989.
17. Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, July 1978.
18. Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.
19. Robert E. Shostak. Deciding combinations of theories. Technical Report 132, SRI-CSL, Menlo Park, CA, January 1984.