Deconstructing Shostak*

Appears in Proc. of IEEE LICS 2001 ©IEEE Press

Harald Rueß and Natarajan Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
{ruess,shankar}@csl.sri.com
Phone: (650)859-5272; Fax: (650)859-2844

Abstract

Decision procedures for equality in a combination of theories are at the core of a number of verification systems. Shostak's decision procedure for equality in the combination of solvable and canonizable theories has been around for nearly two decades. Variations of this decision procedure have been implemented in a number of systems including STP, EHDM*, PVS, STeP, and SVC. The algorithm is quite subtle and a correctness argument for it has remained elusive. Shostak's algorithm and all previously published variants of it yield incomplete decision procedures. We describe a variant of Shostak's algorithm along with proofs of termination, soundness, and completeness.*

1 Introduction

In 1984, Shostak [Sho84] published a decision procedure for the quantifier-free theory of equality over uninterpreted functions combined with other theories that are canonizable and solvable. Such algorithms decide statements of the form $T \vdash a = b$, where T is a collection of equalities, and T, a, and b contain a mixture of interpreted and uninterpreted function symbols. This class of statements includes a large fraction of the proof obligations that arise in verification including those involving extended typechecking, verification conditions generated from Hoare triples, and inductive theorem proving. Shostak's procedure is at the core of several verification systems including STP [SSMS82], EHDM [EHD93], PVS [ORS92], STeP [MT96, Bjø99], and SVC [BDL96]. The soundness of Shostak's algorithm is reasonably straightforward, but its completeness has steadfastly resisted proof. The proof given by Shostak [Sho84] is seriously flawed. Despite its significance and popularity, Shostak's original algorithm and its subsequent variations [CLS96, BDL96, Bjø99] are all incomplete and potentially nonterminating. We explain the ideas underlying Shostak's decision procedure by presenting a correct version of the algorithm along with rigorous proofs for its correctness.

If the terms in a conjecture of the form $T \vdash a = b$ are constructed solely from variables and uninterpreted function symbols, then congruence closure [NO80, Sho78, DST80, CLS96, Kap97, BRRT99] can be used to partition the subterms into equivalence classes respecting T and congruence. For example, when congruence closure is applied to

$$f^3(x) = f(x) \vdash f^5(x) = f(x),$$

the equivalence classes generated by the antecedent equality are $\{x\}, \{f(x), f^3(x), f^5(x)\}$, and $\{f^2(x), f^4(x)\}$. This partition clearly validates the conclusion $f^5(x) = f(x)$.

In practice, a conjecture $T \vdash a = b$ usually contains a mixture of uninterpreted and interpreted function symbols. Semantically, uninterpreted functions are unconstrained, whereas interpreted function are constrained by a *theory*, i.e., a closure condition with respect to consequence on a set of equalities. An example of such an assertion is

$$f(x-1)-1 = x+1,\ f(y)+1 = y-1,\ y+1 = x \vdash \mathit{false},$$

where $+$, $-$, and the numerals are from the theory of linear arithmetic, *false* is an abbreviation for $0 = 1$, and f is an uninterpreted function symbol. The contradiction here cannot be derived solely by congruence closure or linear arithmetic. Linear arithmetic is used to show that $x - 1 = y$ so that $f(x-1) = f(y)$ follows by congruence. Linear arithmetic can then be used to show that $x + 2 = y - 2$ which contradicts $y + 1 = x$.

*This work was supported by SRI International, and by NSF Grant CCR-0082560, DARPA/AFRL Contract F33615-00-C-3043, and NASA Contract NAS1-0079.

Nelson and Oppen [NO79] showed how decision procedures for disjoint equational theories could be combined. Since linear arithmetic and uninterpreted equality are disjoint, this method can be applied to the above example. First, *variable abstraction* is used to obtain a theory-wise partition of the *term universe*, i.e., the subterms of T, a, and b, in a conjecture $T \vdash a = b$. The uninterpreted equality theory Q then consists of the terms $\{f(u), f(y), w, z\}$ and the equalities $\{w = f(u), z = f(y)\}$, and the linear arithmetic theory L consists of the terms $\{u, x, y, x - 1, w - 1, x + 1, z + 1, y - 1, y + 1\}$ and the equalities $\{u = x - 1, w - 1 = x + 1, z + 1 = y - 1, y + 1 = x\}$. The key observation is that once the terms and equalities have been partitioned using variable abstraction, the two theories L and Q need exchange only equalities between variables. Thus, linear arithmetic can be used to derive the equality $u = y$, from which congruence closure derives $w = z$, and the contradiction then follows from linear arithmetic. Since the term universe is fixed in advance, there are only a bounded number of equalities between variables so that the propagation of information between the decision procedures must ultimately converge.

The Nelson-Oppen combination procedure has some disadvantages. The individual decision procedures must carry out their own equality propagation and the communication of equalities between decision procedures can be expensive. The number of equalities is quadratic in the size of the term universe, and each closure operation can itself be linear in the size of the term universe.

Shostak's algorithm tries to gain efficiency by maintaining and propagating equalities within a single congruence closure data structure. Equalities involving interpreted symbols contain more information than uninterpreted equalities. For example, the equality $y + 1 = x$ cannot be processed by merely placing $y + 1$ and x in the same equivalence class. This equality also implies that $y = x - 1$, $y - x = -1$, $x - y = 1$, $y + 3 = x + 2$, and so on. In order to avoid processing all these variations on the given equality, Shostak restricts his attention to *solvable* theories where an equality of the form $y + 1 = x$ can be solved for x to yield the solution $x = y + 1$. If the theories considered are also *canonizable*, then there is a canonizer σ such that whenever an equality $a = b$ is valid, then $\sigma(a) \equiv \sigma(b)$, where $\equiv$ represents syntactic equality. A canonizer for linear arithmetic can be defined to place terms into an ordered sum-of-monomials form. Once a solved form such as $x = y + 1$ has been obtained, all the other consequences $a = b$ of this equality can be obtained by $\sigma(a') = \sigma(b')$ where a' and b' are the results of substituting the solution for x into a and b, respectively. For example, substituting the solution into $y = x - 1$ yields $y = y + 1 - 1$, and the subsequent canonization step yields $y = y$.

The notion of a solvable and canonizable theory is extended to equalities involving a mix of interpreted and uninterpreted symbols by treating uninterpreted terms as variables. For the conjecture,

$$f(x-1)-1 = x+1,\ f(y)+1 = y-1,\ y+1 = x \vdash \mathit{false},$$

Shostak's algorithm would solve the equality $f(x-1)-1 = x+1$ as $f(x-1) = x+2$, the equality $f(y)+1 = y-1$ as $f(y) = y-2$, and $y+1 = x$ as $x = y+1$. Now, $f(x-1)$ and $f(y)$ are congruent because the canonical form for $x - 1$ obtained after substituting the solution $x = y + 1$ is y. By congruence closure, the equivalence classes of $f(x-1)$ and $f(y)$ have to be merged. In Shostak's original algorithm the current representatives of these equivalence classes, namely $x+2$ and $y - 2$ are merged. The resulting equality $x + 2 = y - 2$ is first solved to yield $x = y - 4$. This is incorrect because we already have a solution for x as $x = y+1$ and x should therefore have been eliminated. The new solution $x = y-4$ contradicts the earlier one, but this contradiction goes undetected by Shostak's algorithm. This example can be easily adapted to show nontermination. Consider

$$f(v) = v, f(u) = u - 1, u = v \vdash \mathit{false}.$$

The merging of u and v here leads to the detection of the congruence between $f(u)$ and $f(v)$. This leads to solving of $u - 1 = v$ as $u = v + 1$. Now, the algorithm merges v and $v+1$. Since v occurs in $v+1$, this causes $v + 1$ to be merged with $v + 2$, and so on.

An earlier paper by Cyrluk, Lincoln, and Shankar [CLS96] gave an explanation (with minor corrections) of Shostak's algorithm for congruence closure and its extension to interpreted theories. Though proofs of correctness for the combination algorithm are briefly sketched, the algorithm presented there is both incomplete and nonterminating. Other published variants of Shostak's algorithm used in SVC [BDL96] and STeP [Bjø99] inherit these problems.

In this paper, we present an algorithm that fixes the incompleteness and nontermination in earlier versions of Shostak's algorithms. In the above example, the incompleteness is fixed by substituting the solution for x into the terms representing the different equivalence classes. Thus, when $f(x-1)$ and $f(y)$ are detected to be congruent, their equivalence classes are represented by $y+3$ and $y-2$, respectively. The resulting equality $y+3 = y-2$ easily yields a contradiction. The nontermination is fixed by ensuring that no new mergeable

terms, such as $v+2$, are created during the processing of an axiom in T. Our algorithm is presented as a system of transformations on a set of equalities in order to capture the key insights underlying its correctness. We outline rigorous proofs for the termination, soundness, and completeness of this procedure. The algorithm as presented here emphasizes logical clarity over efficiency, but with suitable optimizations and data structures, it can serve as the basis for an efficient implementation. SRI's ICS (Integrated Canonizer/Solver) decision procedure package [FORS01] is directly based on the algorithm studied here.

Section 2 introduces the theory of equality, which is augmented in Section 3 with function symbols from a canonizable and solvable theory. Section 3 also introduces the basic building blocks for the decision procedure. The algorithm itself is described in Section 4 along with some example hand-simulations. The proofs of termination, soundness, and completeness are outlined in Section 5.

2 Background

With respect to a *signature* consisting of a set of function symbols F and a set of variables V, a term is either a variable x from V or an application $f(a_1, \ldots, a_n)$ of an n-ary function symbol f from F to n terms $a_1, \ldots, a_n$, where $0 \leq n$. The metavariable conventions are that u, v, x, y, and z range over variables, and a, b, c, d, and e range over terms. The metavariables R, S, and T, range over sets of equalities. The metatheoretic assertion $a \equiv b$ indicates that a and b are syntactically identical terms. Let $vars(a)$, $vars(a = b)$, and $vars(T)$ return the variables occurring in a term a, an equality $a = b$, and a set of equalities T, respectively. The operation $[\![a]\!]$ is defined to return the set of all subterms of a.

Some of the function symbols are *interpreted*, i.e., they have a specific interpretation in some given theory τ, while the remaining function symbols are uninterpreted, i.e., can be assigned arbitrary interpretations. A term $f(a_1, \ldots, a_n)$ is interpreted (uninterpreted) if f is interpreted (uninterpreted). A term e is *non-interpreted* if it is either a variable or an uninterpreted term. We say that a term a *occurs interpreted* in a term e if there is an occurrence of a in e that is not properly within an uninterpreted subterm of e. Likewise, a *occurs uninterpreted* in e if a is a proper subterm of an uninterpreted subterm of e. $solvables(a)$ denotes the set of outermost non-interpreted subterms of a, i.e., those that do not occur uninterpreted in a.

$$\begin{aligned} solvables(f(a_1, \ldots, a_n)) &= \bigcup_i solvables(a_i), \\ &\quad \text{if } f \text{ is interpreted} \\ solvables(a) &= \{a\}, \text{ otherwise} \end{aligned}$$

The theory of equality deals with sequents of the form $T \vdash a = b$. We will insist that these sequents be such that $vars(a = b) \subseteq vars(T)$. The proof theory for equality is given by the following inference rules.

1. Axiom: $\dfrac{}{T \vdash a = b}$, for $a = b \in T$.
2. Reflexivity: $\dfrac{}{T \vdash a = a}$.
3. Symmetry: $\dfrac{T \vdash a = b}{T \vdash b = a}$.
4. Transitivity: $\dfrac{T \vdash a = b \qquad T \vdash b = c}{T \vdash a = c}$.
5. Congruence: $\dfrac{T \vdash a_1 = b_1 \quad \ldots \quad T \vdash a_n = b_n}{T \vdash f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)}$.

The semantics for terms is given by a model M over a domain D and an assignment ρ for the variables so that $M[\![x]\!]_\rho = \rho(x)$ and $M[\![f(a_1, \ldots, a_n)]\!]_\rho = M(f)(M[\![a_1]\!]_\rho, \ldots, M[\![a_n]\!]_\rho)$, and $M[\![a]\!]_\rho \in D$ for all a. We say that $M, \rho \models a = b$ iff $M[\![a]\!]_\rho = M[\![b]\!]_\rho$, and $M \models a = b$ iff $M, \rho \models a = b$ for all assignments ρ over $vars(a = b)$. We write $M, \rho \models S$ when $\forall a, b : a = b \in S \supset M, \rho \models a = b$, and $M, \rho \models T \vdash a = b$ when $(M, \rho \models T) \supset (M, \rho \models a = b)$.

3 Canonizable and Solvable Theories

Shostak's algorithm goes beyond congruence closure by deciding equality in the presence of function symbols that are *interpreted* in a theory τ [Sho84, CLS96]. The algorithm is targeted at canonizable and solvable theories, i.e., theories that are equipped with solvers and canonizers as outlined below. We write $\models_\tau a = b$ to indicate that $a = b$ is valid in theory τ. The canonizer and solver are first described for pure τ-terms, i.e., without any uninterpreted function symbols, and then extended to uninterpreted terms by regarding these as variables.

Definition 3.1 *A theory τ is* canonizable *if there is a canonizer σ such that*

1. $\models_\tau a = b$ *iff* $\sigma(a) \equiv \sigma(b)$.
2. $\sigma(x) \equiv x$.
3. $vars(\sigma(a)) \subseteq vars(a)$.
4. $\sigma(\sigma(a)) \equiv \sigma(a)$.
5. *If* $\sigma(a) \equiv f(b_1, \ldots, b_n)$, *then* $\sigma(b_i) \equiv b_i$ *for* $1 \leq i \leq n$.

For example, a canonizer σ for the theory of linear arithmetic can be defined to transform expressions into an ordered-sum-of-monomials normal form. A term a is said to be *canonical* if $\sigma(a) \equiv a$.

Definition 3.2 *A model M is a* σ-model *if $M \models a = \sigma(a)$ for any term a, and $M \not\models a = b$ for distinct canonical, variable-free terms a and b.*

Definition 3.3 *A set of equalities S and $a = b$ are* σ-equivalent *iff for all σ-models M and assignments ρ over the variables in a and b, $M, \rho \models a = b$ iff there is an assignment ρ' extending ρ, over the variables in S, a, and b, such that $M, \rho' \models S$.*

Definition 3.4 *A canonizable theory is* solvable *if there is an operation solve such that $solve(a = b) = \bot$ if $a = b$ is unsatisfiable in any σ-model, or $S = solve(a = b)$ for a set of equalities S such that*

1. *S is a set of n equalities of the form $x_i = e_i$ for $0 \leq n$ where for each i, $0 < i \leq n$,*
   (a) $x_i \in vars(a = b)$.
   (b) $x_i \notin vars(e_j)$, *for* j, $0 < j \leq n$.
   (c) $x_i \not\equiv x_j$, *for* $i \neq j$ *and* $0 < j \leq n$.
   (d) $\sigma(e_i) \equiv e_i$.
2. *S and $a = b$ are σ-equivalent.*

A solver for linear arithmetic, for example, takes an equation of the form

$$c + a_1x_1 + \ldots + a_nx_n = d + b_1x_1 + \ldots + b_nx_n,$$

where $a_1 \neq b_1$, and returns

$$\begin{array}{rcl} x_1 = \sigma(& & (d-c)/(a_1-b_1) \\ & + & ((b_2-a_2)/(a_1-b_1)) * x_2 \\ & + & \ldots \\ & + & ((b_n-a_n)/(a_1-b_1)) * x_n). \end{array}$$

In general, $solve(a = b)$ may contain variables that do not occur in $a = b$, and vice-versa.

There are a number of interesting canonizable and solvable theories including linear arithmetic, the theory of tuples and projections, algebraic datatypes like lists, set algebra, and the theory of fixed-sized bitvectors. In many cases, the canonizability and solvability of the union of theories (with disjoint signatures) follows from the canonizability and solvability of its constituent theories.[1] We do not address modularity issues here but instead assume that we already have a canonizer and solver for a single combined theory.

The solvers and canonizers characterized above are intended to work in the absence of uninterpreted function symbols. They are adapted to terms containing uninterpreted subterms by treating these subterms as variables. Canonizers are applied to terms containing uninterpreted subterms by renaming distinct uninterpreted subterms with distinct new variables. For a given term a, let γ be a bijective mapping between a set of variables X that do not appear in a and the uninterpreted subterms of a. The application of a substitution γ to a term a, written $\gamma[a]$, is defined so that $\gamma[a] = f(\gamma[a_1], \ldots, \gamma[a_n])$ if $a \equiv f(a_1, \ldots, a_n)$, where f is interpreted. If a is in the domain of γ, then $\gamma[a] = \gamma(a)$, and otherwise, $\gamma[a] = a$. Then $\sigma(a)$ is $\gamma[\sigma(\gamma^{-1}[a])]$.

For solving equalities containing uninterpreted terms, we introduce, as with σ, a bijective map γ between a set of variables X not occurring in a or b, and the uninterpreted subterms of a and b, such that

$$solve(a = b) = \gamma[solve(\gamma^{-1}[a] = \gamma^{-1}[b])] \ .$$

When uninterpreted terms are handled as above, the conditions in Definitions 3.1 and 3.4 must be suitably adapted by using $solvables(a)$ instead of $vars(a)$.

The proof theory for equality is augmented for canonizable, solvable theories by the proof rules:

1. Canonization: $\dfrac{}{T \vdash a = \sigma(a)}$, for any term a.
2. Solve: $\dfrac{T \vdash a = b \quad T \cup S \vdash c = d}{T \vdash c = d}$ if $S = solve(a = b) \neq \bot$ and $vars(c = d) \subseteq vars(T)$.
3. Solve-$\bot$: $\dfrac{T \vdash a = b}{T \vdash \mathit{false}}$, if $solve(a = b) = \bot$.

A sequent $T \vdash c = d$ is derivable if there is a proof of $T \vdash c = d$ using one of the inference rules: axiom, reflexivity, symmetry, transitivity, congruence, canonization, solve, or solve-$\bot$. We say that $T \vdash S$ is derivable if $T \vdash c = d$ is derivable for every $c = d$ in S. The sequent $T, S \vdash c = d$ is just $T \cup S \vdash c = d$. The *weakening* and *cut* lemmas below are easily verified.

[1] The general result on combining solvers claimed by Shostak [Sho84] is incorrect, but there are some restricted results on combining equational unifiers [BS96] that can be applied here.

Lemma 3.5 (weakening) *If $T \subseteq T'$ and $T \vdash a = b$ is derivable, then $T' \vdash a = b$ is derivable.*

Lemma 3.6 (cut) *If $T' \vdash T$ and $T \vdash a = b$ is derivable, then $T' \vdash a = b$ is derivable.*

Theorem 3.7 (proof soundness) *If $T \vdash a = b$ is derivable, then for any σ-model M and assignment ρ over $vars(T)$, $M, \rho \models T \vdash a = b$.*

Proof. By induction on the derivation of $T \vdash a = b$. The soundness of the *solve* rules follows from the conditions in Definition 3.4. ∎

A set of equalities S is said to be *functional* (in a left-to-right reading of the equality) if whenever $a = b \in S$ and $a = b' \in S$, $b \equiv b'$. For example, the solution set returned by *solve* is functional. A functional set of equalities can be treated as a substitution and the associated operations are defined below. $S(a)$ returns the solution for a if it exists in S, and a itself, otherwise. If $a = b$ is in S for some b, then a is in the domain of S, i.e., $dom(S)$.

$$\begin{aligned} S(a) &= \begin{cases} b & \text{if } a = b \in S \\ a & \text{otherwise} \end{cases} \\ dom(S) &= \{a \mid \exists b.\ a = b \in S\}. \end{aligned}$$

The operation $a \overset{S}{\sim} b$ checks if a is congruent to b in S, i.e., $a \equiv f(a_1, \ldots, a_n)$, $b \equiv f(b_1, \ldots, b_n)$, and $S(a_i) \equiv S(b_i)$ for $1 \leq i \leq n$. A set of equalities S is said to be *congruence-closed* when for any terms a and b in $dom(S)$ such that $a \overset{S}{\sim} b$, we have $S(a) \equiv S(b)$.

$S[a]$ replaces a subterm b in a by $S(b)$, where $b \in solvables(a)$.

$$\begin{aligned} S[f(a_1, \ldots, a_n)] &= f(S[a_1], \ldots, S[a_n]), \\ &\quad \text{if } f \text{ is interpreted} \\ S[a] &= S(a), \text{ otherwise.} \end{aligned}$$

$norm(S)(a)$ is a normal form for a with respect to S and is defined as $\sigma(S[a])$. The operation *norm* does not appear in Shostak's algorithm and is the key element of our algorithm and its proof. With S fixed, we use $\hat{a}$ as a syntactic abbreviation for $norm(S)(a)$.

$$norm(S)(a) = \sigma(S[a]).$$

Lemma 3.8 *If $solve(a = b) = S \neq \bot$, then $norm(S)(a) \equiv norm(S)(b)$.*

Proof. By definitions 3.3 and 3.4(2), for any σ-model M and assignment ρ', we have $M, \rho' \models S \iff M, \rho' \models a = b$. Let $a' \equiv S[a]$ and $b' \equiv S[b]$. By induction on a, $M, \rho' \models a = a'$, and similarly $M, \rho' \models b = b'$. Hence, $M, \rho' \models a' = b'$. Then, since M is a σ-model, by Definition 3.2, it must be the case that $\sigma(a') \equiv \sigma(b')$, and therefore $norm(S)(a) \equiv norm(S)(b)$. ∎

The definition of the *lookup* operation uses Hilbert's epsilon operator, indicated by the keyword *when*, to return $S(f(b_1, \ldots, b_n))$ when $b_1, \ldots, b_n$ satisfying the listed conditions can be found. If no such $b_1, \ldots, b_n$ can be found, then $lookup(S)(a)$ returns a itself. We show later that the *lookup* operation is used only when the results of this choice are deterministic.

$$\begin{aligned} lookup(S)(f(a_1, \ldots, a_n)) &= S(f(b_1, \ldots, b_n)), \\ &\quad when\ b_1, \ldots, b_n : \\ &\quad f(b_1, \ldots, b_n) \in dom(S), \\ &\quad \text{and } a_i \equiv S(b_i), \\ &\quad \text{for } 1 \leq i \leq n \\ lookup(S)(a) &= a, \text{ otherwise.} \end{aligned}$$

$can(S)(a)$ is a canonical form in which any uninterpreted subterm e that is congruent to a known left-hand side e' in S is replaced by $S(e')$. It is analogous to the *canon* operation in Shostak's algorithm. We use $\overline{a}$ as a syntactic abbreviation for $can(S)(a)$.

$$\begin{aligned} can(S)(f(a_1, \ldots, a_n)) &= lookup(S)(f(\overline{a_1}, \ldots, \overline{a_n})), \\ &\quad \text{if } f \text{ is uninterpreted} \\ can(S)(f(a_1, \ldots, a_n)) &= \sigma(f(\overline{a_1}, \ldots, \overline{a_n})), \\ &\quad \text{if } f \text{ is interpreted} \\ can(S)(a) &= S(a), \text{ otherwise.} \end{aligned}$$

Lemma 3.9 (σ-norm) *If S is functional, then $norm(S)(\sigma(a)) \equiv \hat{a}$ and $can(S)(\sigma(a)) \equiv \overline{a}$.*

Proof. We know that $\vdash \sigma(a) = a$. Then for $b' = S[\sigma(a)]$ and $b = S[a]$, the equality $b' = b$ is valid in every σ-model. Then by Definition 3.2, $\sigma(S[\sigma(a)]) \equiv \sigma(S[a])$, and hence the first part of the theorem.

The reasoning in the second part is similar. If we let $R = \{b = \overline{b} \mid b \in [\![a]\!]\}$, then $can(S)(a) \equiv norm(R)(a)$. We can therefore use the first part of the theorem to establish the second part. ∎

We next introduce a composition operation for merging the results of a *solve* operation into an existing solution set. When $R \circ S$ is used, S must be functional, and the result contains $a = \hat{b}$ for each equality $a = b$ in R in addition to the equalities in S.

$$R \circ S = \{a = \hat{b} \mid a = b \in R\} \cup S.$$

The following lemmas about composition are given without proof.

Lemma 3.10 (norm decomposition) *If $R \cup S$ is functional, then*

$$norm(R \circ S)(a) \equiv norm(S)(norm(R)(a)).$$

$$
\begin{array}{rcl}
process(\{a=b, T\}) & = & assert(a=b, process(T)) \\
process(\emptyset) & = & \emptyset. \\
\\
assert(a=b, \bot) & = & \bot \\
assert(a=b, S) & = & cc(merge(\overline{a}, \overline{b}, S^+)), \text{ where,} \\
 & & S^+ = expand(S, \overline{a}, \overline{b}). \\
\\
expand(S, a, b) & = & S \cup \{e = e \mid e \in new(S, a, b)\}. \\
\\
new(S, a, b) & = & [\![a=b]\!] - dom(S). \\
\\
merge(a, b, S) & = & \bot, \text{if } solve(a=b) = \bot \\
merge(a, b, S) & = & S \circ solve(a=b), \text{ otherwise.} \\
\\
cc(\bot) & = & \bot \\
cc(S) & = & cc(merge(S(a), S(b), S)), \\
 & & when\ a, b: \\
 & & a, b \in dom(S) \\
 & & a \overset{S}{\sim} b, \text{ and } S(a) \not\equiv S(b) \\
cc(S) & = & S, \text{ otherwise.}
\end{array}
$$

Figure 1: Main Procedure: *process*

Lemma 3.11 (associativity of composition) *If $Q \cup R \cup S$ is functional, then*

$$(Q \circ R) \circ S = Q \circ (R \circ S).$$

Lemma 3.12 (monotonicity) *If $R \cup S$ is functional, then if $R(a) \equiv R(b)$, then $(R \circ S)(a) \equiv (R \circ S)(b)$, for any a and b.*

4 An Algorithm for Deciding Equality in the Presence of Theories

We next present an algorithm for deciding $T \vdash c = d$ for terms containing uninterpreted function symbols and function symbols interpreted in a canonizable and solvable theory. The algorithm for verifying $T \vdash c = d$ checks that $can(S)(c) \equiv can(S)(d)$, where $S = process(T)$. The *process* procedure shown in Figure 1, is written as a functional program. It is a mathematical description of the algorithm and not an optimized implementation. The *state* of the algorithm consists of a set of equalities S which holds the solution set. We demonstrate as an invariant that S is functional. Two terms a and b in $dom(S)$ are in the same equivalence class according to S if $S(a) \equiv S(b)$.

The operation $process(T)$ returns a final solution set by starting with an empty solution set and successively processing each equality $a = b$ in T by invoking $assert(a = b, S)$, where S is the state as returned by the recursive call of *process*. The invocation of $assert(a = b, S)$ is executed by first reducing a and b to their respective canonical forms $\overline{a}$ and $\overline{b}$. Next, S is expanded to include $e = e$ for each subterm e of $\overline{a} = \overline{b}$ where $c \notin dom(S)$. This preprocessing step ensures that S contains entries corresponding to any terms that might be needed in the congruence closure phase in the operation cc.[2] The *merge* operation then solves the equality $\overline{a} = \overline{b}$ to get a solution[3] S', and returns $S \circ S'$ as the new value for the state S. As we will show, this new value affirms $a = b$, but it is not congruence-closed and hence does not contain all the consequences of the assertion $a = b$. The step $cc(S)$ computes the congruence closure of S by repeatedly picking a pair of congruent terms a and b from $dom(S)$ such that $S(a) \not\equiv S(b)$ and merging them using $merge(S(a), S(b), S)$. Eventually either a contradiction is found or all congruent left-hand sides in S are merged and the cc operation terminates returning a congruence-closed solution set.

The above algorithm fixes the nontermination and incompleteness problems in Shostak's algorithm by introducing the *norm* operation and the composition operator $R \circ S$ to fold in a solution. The *norm* operation ensures that no new uninterpreted terms are introduced during congruence closure in the function cc, as is needed to guarantee termination. The composition operator $R \circ S$ ensures that any newly generated solution S is immediately substituted into R and the algorithm never attempts to find a solution for an already solved non-interpreted term.

We first illustrate the algorithm on some examples. The first example contains no interpreted symbols.

Example 4.1 Consider the goal $f^5(x) = x, f^3(x) = x \vdash f(x) = x$. The value of S after the base case is $\emptyset$. After the preprocessing of $f^3(x) = x$ in *assert*, S is $\{x = x, f(x) = f(x), f^2(x) = f^2(x), f^3(x) = f^3(x)\}$. After merging $f^3(x)$ and x, S is $\{x = x, f(x) = f(x), f^2(x) = f^2(x), f^3(x) = x\}$. When $f^5(x) = x$ is preprocessed in *assert*, $can(S)(f^5(x))$ yields $f^2(x)$ since $S(f^3(x)) \equiv x$, and S is left unchanged. When $f^2(x)$ and x have been merged, S is $\{x = x, f(x) = f(x), f^2(x) = x, f^3(x) = x\}$. Now $f(x) \overset{S}{\sim} f^3(x)$ and hence $f(x)$ and x are merged so that S is now $\{x = x, f(x) = x, f^2(x) = x, f^3(x) = x\}$.

[2] Actually, the interpreted subterms of $\overline{a} = \overline{b}$ need not all be included in $dom(S)$. Only those that are immediate subterms of uninterpreted subterms in $\overline{a} = \overline{b}$ are needed.

[3] Any variables occurring in $solve(a = b)$ and not in $vars(a = b)$ must be fresh, i.e., they must not occur in the original conjecture or be generated by any other invocation of *solve*.

The conclusion $f(x) = x$ easily follows since $can(S)(f(x)) \equiv x \equiv can(S)(x)$.

Example 4.2 Consider $y + 1 = x$, $f(y) + 1 = y - 1$, $f(x-1) - 1 = x + 1 \vdash \mathit{false}$ which is a permutation of our earlier example. Starting with $S \equiv \emptyset$ in the base case, the preprocessing of $f(x-1) - 1 = x + 1$ causes the equation to be placed into canonical form as $-1 + f(-1 + x) = 1 + x$ and S is set to

$$\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x, \\ f(-1 + x) = f(-1 + x), 1 + x = 1 + x\}.$$

Solving $-1+f(-1+x) = 1+x$ yields $f(-1+x) = 2+x$, and S is set to

$$\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x, \\ f(-1 + x) = 2 + x, 1 + x = 1 + x\}.$$

No unmerged congruences are detected. Next, $f(y) + 1 = y - 1$ is asserted. Its canonical form is $1 + f(y) = -1 + y$, and once this equality is asserted, the value of S is

$$\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x, \\ f(-1 + x) = 2 + x, 1 + x = 1 + x, y = y, \\ f(y) = -2 + y, -1 + y = -1 + y, \\ 1 + f(y) = -1 + y\}.$$

Next $y + 1 = x$ is processed. Its canonical form is $1 + y = x$ and the equality $1 + y = 1 + y$ is added to S. Solving $y + 1 = x$ yields $x = 1 + y$, and S is reset to

$$\{\ 1 = 1, -1 = -1, x = 1 + y, -1 + x = y, \\ f(-1 + x) = 3 + y, 1 + x = 2 + y, y = y, \\ f(y) = -2 + y, -1 + y = -1 + y, \\ 1 + f(y) = -1 + y, 1 + y = 1 + y\}.$$

The congruence close operation *cc* detects the congruence $f(1 - y) \overset{S}{\sim} f(x)$ and invokes *merge* on $3 + y$ and $-2 + y$. Solving this equality $3 + y = -2 + y$ yields $\perp$ returning the desired contradiction.

5 Analysis

We describe the proofs of termination, soundness, and completeness, and also present a complexity analysis.

Key Invariants. The *merge* operation is clearly the workhorse of the procedure since it is invoked from within both *assert* and *cc*. Let $U(X)$ represent the set $\{a \in X \mid a \text{ uninterpreted}\}$ of uninterpreted terms in the set X. Let A be $\mathit{solvables}(a)$, B be $\mathit{solvables}(b)$, and $S' = \mathit{merge}(a, b, S)$, then assuming $U(A \cup B) \subseteq \mathit{dom}(S)$ and for all $c \in A \cup B$, $S(c) \equiv c$, the following properties hold of S' if they hold of S:

1. Functionality.
2. Subterm closure: S is *subterm-closed* if for any $a \in \mathit{dom}(S)$, $[\![a]\!] \subseteq \mathit{dom}(S)$.
3. Range closure: S is *range-closed* if for any $a \in \mathit{dom}(S)$, $U(\mathit{solvables}(S(a))) \subseteq \mathit{dom}(S)$, and for any $c \in \mathit{solvables}(S(a))$, $S(c) \equiv c$.
4. Norm closure: S is *norm-closed* if $S(a) \equiv \mathit{norm}(S)(a)$ for a in $\mathit{dom}(S)$. This of course holds trivially for uninterpreted terms a.
5. Idempotence: S is *idempotent* if $S[S(a)] \equiv S(a)$, $\mathit{norm}(S)(S(a)) \equiv S(a)$, and $\mathit{norm}(S)(\mathit{norm}(S)(a)) \equiv \mathit{norm}(S)(a)$.

These properties can be easily established by inspection. Since whenever $\mathit{merge}(a, b, S)$ is invoked in the algorithm, the arguments do satisfy the conditions $U(A \cup B) \subseteq \mathit{dom}(S)$ and for all $c \in A \cup B$, $S(c) \equiv c$, it then follows that these properties are also preserved by *assert* and *cc*, and therefore hold of $\mathit{process}(T)$. We assume below that these invariants hold of S whenever the metavariable S is used with or without subscripts or superscripts.

Lemma 5.1 (merge equivalence) *Let $A = \mathit{solvables}(a)$ and $B \equiv \mathit{solvables}(b)$. Given that $U(A \cup B) \subseteq \mathit{dom}(S)$ and for all $c \in A \cup B$, $S(c) \equiv c$, if $S' = \mathit{merge}(a, b, S) \neq \perp$, then*

1. $\mathit{norm}(S')(a) \equiv \mathit{norm}(S')(b)$.
2. $U(\mathit{dom}(S')) = U(\mathit{dom}(S))$.

Proof. Let $R \equiv \mathit{solve}(a = b)$. By definition, $\mathit{merge}(a, b, S) \equiv S \circ R$. By Lemma 3.8, $\mathit{norm}(R)(a) \equiv \mathit{norm}(R)(b)$. Since $S(c) \equiv c$ for $c \in A \cup B$, $\mathit{norm}(S)(a) \equiv a$ and $\mathit{norm}(S)(b) \equiv b$. Hence, by *norm decomposition*, we have $\mathit{norm}(S')(a) \equiv \mathit{norm}(S')(b)$.

By Definition 3.4, $\mathit{dom}(R) \subseteq A \cup B$, hence $U(\mathit{dom}(S')) = U(\mathit{dom}(S))$. ∎

Termination. We define $\#(S)$ to represent the number of distinct equivalence classes partitioning $U(\mathit{dom}(S))$ as given by $P(S)$.

$$\begin{aligned} E(S)(a) &= \{b \in U(\mathit{dom}(S)) \mid S(b) \equiv S(a)\} \\ P(S) &= \{E(S)(a) \mid a \in U(\mathit{dom}(S))\} \\ \#(S) &= |P(S)| \end{aligned}$$

The definition of $cc(S)$ terminates because the measure $\#(S)$ decreases with each recursive call. If in the definition of *cc*, $merge(S(a), S(b), S) = \bot$, then clearly *cc* terminates. Otherwise, let $S' = merge(S(a), S(b), S) \neq \bot$, for a and b in $dom(S)$ such that $S(a) \not\equiv S(b)$ and $a \overset{S}{\sim} b$. In this case a and b must be uninterpreted terms since for interpreted terms a and b, if $a \overset{S}{\sim} b$, then $S(a) \equiv S(b)$ by *norm closure*. By *merge equivalence*, $norm(S')(S(a)) \equiv norm(S')(S(b))$ and $U(dom(S')) = U(dom(S))$. By *monotonicity*, for any c and d such that $S(c) \equiv S(d)$, we have $S'(c) \equiv S(d)$, and therefore $\#(S') \leq \#(S)$. However, by *norm closure*, $S'(a) \equiv S'(b)$ so that $\#(S') < \#(S)$.

Soundness. The following lemmas establish the soundness of the operations *norm* and *can* with respect to S. *Substitution soundness* and *can soundness* are proved by a straightforward induction on a, and *norm soundness* is a simple consequence of *substitution soundness*.

Lemma 5.2 (substitution soundness)
If $vars(a) \subseteq vars(T \cup S)$, *then* $T, S \vdash a = a'$ *is derivable, for* $a' \equiv S[a]$.

Lemma 5.3 (norm soundness)
If $vars(a) \subseteq vars(T \cup S)$, *then* $T, S \vdash a = \hat{a}$ *is derivable.*

Lemma 5.4 (can soundness)
If $vars(a) \subseteq vars(T \cup S)$, *then* $T, S \vdash a = \bar{a}$ *is derivable.*

Lemma 5.5 (merge soundness)
If $S' = merge(a, b, S) \neq \bot$, *then if* $T, S \vdash a = b$, *and* $T, S' \vdash c = d$ *with* $vars(c = d) \subseteq vars(T \cup S)$, *then* $T, S \vdash c = d$. *Otherwise,* $merge(a, b, S) = \bot$, *and* $T, S \vdash \bot$.

Proof. If $S' = merge(a, b, S) \neq \bot$, then let $R = solve(a = b)$. By *norm soundness*, $S, R \vdash S'$, and hence by *cut*, $T, S, R \vdash c = d$ is derivable. By the *solve* rule, $T, S \vdash c = d$ is derivable.

If $merge(a, b, S) = \bot$, then by similar reasoning using the *solve*-$\bot$ rule, $T, S \vdash false$ is derivable. ∎

Lemma 5.6 (cc soundness) *If* $S' = cc(S) \neq \bot$, $T, S' \vdash a = b$ *for* $vars(a = b) \subseteq vars(T, S)$, *then* $T, S \vdash a = b$ *is derivable. Otherwise,* $cc(S) = \bot$, *and* $S \vdash false$ *is derivable.*

Proof. By computation induction on the definition of *cc* using *merge soundness*. ∎

Lemma 5.7 (process soundness)
If $S = process(T_1) \neq \bot$, $T_1 \subseteq T_2$, *and* $T_2, S \vdash c = d$ *for* $vars(c = d) \subseteq vars(T_2)$, *then* $T_2 \vdash c = d$ *is derivable. Otherwise,* $process(T_1) = \bot$, *and* $T_1 \vdash false$ *is derivable.*

Proof. By induction on the length of T_1. In the base case, S is empty and the theorem follows trivially. In the induction step, with $T_1 = \{a = b, T_1'\}$ and $S' = process(T_1')$, we have the induction hypothesis that $T_2 \vdash c = d$ is derivable if $T_2, S' \vdash c = d$ is derivable, for any c, d such that $vars(c = d) \subseteq vars(T_2)$. We know by *can soundness* that $T_2, S' \vdash \overline{a} = a$ and $T_2, S' \vdash \overline{b} = b$ are derivable. When S' is augmented with identities over subterms of $\overline{a}$ and $\overline{b}$ to get S'^{+}, we have the derivability of $T_2, S' \vdash S'^{+}$. By *cc soundness*, we then have the derivability of $T_2, S'^{+} \vdash c = d$ from that of $T_2, S \vdash c = d$. The derivability of $T_2, S' \vdash c = d$ then follows by *cut* from that of $T_2, S'^{+} \vdash c = d$, and we get the conclusion $T_2 \vdash c = d$ by the induction hypothesis.

A similar induction argument shows that when $process(T_1) = \bot$, then $T_2 \vdash false$. ∎

Theorem 5.8 (soundness) *If* $S = process(T) \neq \bot$, $vars(a = b) \subseteq vars(T)$, *and* $\overline{a} \equiv \overline{b}$, *then* $T \vdash a = b$ *is derivable. Otherwise,* $process(T) = \bot$, *and* $T \vdash false$ *is derivable.*

Proof. If $S = process(T) \neq \bot$, then by *can soundness*, $T, S \vdash a = \overline{a}$ and $T, S \vdash b = \overline{b}$ are derivable. Hence, by transitivity and symmetry, $T, S \vdash a = b$ is derivable. Therefore, by *process soundness*, $T \vdash a = b$ is derivable.

If $process(T) = \bot$, then already by *process soundness*, $T \vdash false$. ∎

Completeness. We show that when $S = process(T)$ then $can(S)$ is a σ-model satisfying T. When this is the case, completeness follows from *proof soundness*. In proving completeness, we exploit the property that the output of *process* is congruence-closed.

Lemma 5.9 (confluence)
If S *is congruence-closed and* $U(\llbracket a \rrbracket) \subseteq dom(S)$, *then* $can(S)(a) \equiv norm(S)(a)$.

Proof. The proof is by induction on a. In the base case, when a is a variable, $can(S)(a) \equiv S(a) \equiv norm(S)(a)$.

If a is uninterpreted and of the form $f(a_1, \ldots, a_n)$, then $can(S)(a) \equiv lookup(S)(f(\overline{a_1}, \ldots, \overline{a_n}))$. Since S is *subterm-closed*, by the induction hypothesis and *norm closure*, we have $\overline{a_i} \equiv \hat{a}_i \equiv S(a_i)$ for $0 < i \leq n$. Then

there must be some b of the form $f(b_1, \ldots, b_n)$ such that $S(b_i) \equiv S(a_i)$, for $0 < i \leq n$, since a itself is such a b. Then by *congruence closure* and *norm closure*, $\overline{a} \equiv S(b) \equiv S(a) \equiv \hat{a}$, since $a \overset{S}{\sim} b$.

If a is interpreted, by the induction hypothesis and *subterm closure*, $\overline{a} \equiv \sigma(f(\overline{a_1}, \ldots, \overline{a_n})) \equiv \sigma(f(\hat{a_1}, \ldots, \hat{a_n})) \equiv \hat{a}$. ∎

Lemma 5.10 (can composition) *If $S' = S \circ R$ and S' is congruence-closed, then $can(S')(can(S)(a)) \equiv can(S')(a)$.*

Proof. By induction on a. When a is a variable. $can(S)(a) \equiv S(a)$. If $a \notin dom(S)$, then $S(a) = a$, and hence the conclusion. Otherwise, by range-closure, $U(\llbracket S(a) \rrbracket) \subseteq dom(S) \subseteq dom(S')$. Then, by *confluence*, *norm decomposition*, and *idempotence*, $can(S')(S(a)) \equiv norm(S')(S(a)) \equiv norm(R)(norm(S)(S(a))) \equiv norm(R)(norm(S)(a)) \equiv norm(S')(a) \equiv can(S')(a)$.

In the induction step, let $a \equiv f(a_1, \ldots, a_n)$. If a is uninterpreted, then if

$$f(\overline{a_1}, \ldots, \overline{a_n}) \overset{S}{\sim} f(b_1, \ldots, b_n)$$

for some $f(b_1, \ldots, b_n) \in dom(S)$, then $\overline{a} \equiv S(f(b_1, \ldots, b_n))$. The reasoning used in the base case can then be repeated to derive the conclusion. Otherwise, $\overline{a} \equiv f(\overline{a_1}, \ldots, \overline{a_n})$ and by the induction hypothesis and the definition of *can*, $can(S')(\overline{a}) \equiv lookup(S')(f(can(S')(a_1), \ldots, can(S')(a_n))) \equiv can(S')(a)$.

When a is interpreted, by the induction hypothesis and the *σ-norm* lemma,

$$\begin{array}{ll} & can(S')(\overline{a}) \\ \equiv & can(S')(\sigma(f(\overline{a_1}, \ldots, \overline{a_n}))) \\ \equiv & \sigma(f(can(S')(\overline{a_1}), \ldots, can(S')(\overline{a_n}))) \\ \equiv & can(S')(a). \end{array}$$

∎

Lemma *can composition* with $\emptyset$ for R yields the idempotence of $can(S)$ for congruence-closed S so that we can define a σ-model M_S in terms of $can(S)$. The domain D of M_S consists of $\{a | can(S)(a) = a\}$. The mapping of functions is such that $M_S(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) = lookup(S)(f(\mathbf{a_1}, \ldots, \mathbf{a_n}))$, if f is uninterpreted. If f is interpreted $M_S(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) = \sigma(f(\mathbf{a_1}, \ldots, \mathbf{a_n}))$. If $\rho[x] = \rho(x)$ and $\rho[f(a_1, \ldots, a_n)] = f(\rho[a_1], \ldots, \rho[a_n])$, then by the idempotence of $can(S)$, $M_S\llbracket a \rrbracket_\rho$ is just $can(S)(\rho[a])$. Lemma σ-norm can then be used to show $M_S \models \sigma(a) = a$. M_S is therefore a σ-model. Correspondingly, for a given set of variables X, ρ_S^X is defined so that $\rho_S^X(x) = can(S)(x)$ for $x \in X$.

Lemma 5.11 (can σ-model) *If $S = process(T) \neq \bot$ and $X = vars(T)$, then $M_S, \rho_S^X \models a = b$ for any $a = b \in T$.*

Proof. Showing that $M_S, \rho_S^X \models a = b$ is the same as showing that $can(S)(a) \equiv can(S)(b)$. The proof is by induction on T. In the base case, T is empty. In the induction step, $T = \{a = b, T'\}$ with $X' = vars(T')$. Let $S' = process(T')$. By the induction hypothesis, $M_{S'}, \rho_{S'}^{X'} \models T'$. With $S'^{+} = expand(S, a', b')$ for $a' \equiv can(S')(a)$ and $b' \equiv can(S')(b)$, let $S_0 = merge(a, b, S'^{+})$, hence by *merge equivalence*, $norm(S_0)(a') \equiv norm(S_0)(b')$. By associativity of composition, it can be shown that there is an R such that $S = S_0 \circ R$ and an R' such that $S = S'^{+} \circ R'$. Hence by monotonicity, $norm(S)(a') \equiv norm(S)(b')$. Since S is congruence-closed, by *confluence*, $can(S)(a') \equiv norm(S)(a')$ and $can(S)(b') \equiv norm(S)(b')$. Hence, $can(S)(a') \equiv can(S)(b')$.

It can also be shown that $can(S'^{+})(a) \equiv can(S')(a)$, and similarly for b. Therefore, by *can composition*, we have $can(S)(a) \equiv can(S)(b)$, and hence $M_S, \rho_S^X \models a = b$. A similar argument shows that for $c = d \in T'$, since $can(S')(c) \equiv can(S')(d)$, we also have $can(S)(c) \equiv can(S)(d)$. ∎

When $T \vdash false$ is derivable, we know by *proof soundness* that there is no σ-model satisfying T and hence by the *can σ-model* lemma, $process(T)$ must be $\bot$.

Theorem 5.12 (completeness) *If $S = process(T) \neq \bot$ and $T \vdash a = b$, then $can(S)(a) \equiv can(S)(b)$.*

Proof. Since $M_S, \rho_S^X \models T$ by *can σ-model* for $X = vars(T)$, we have by *proof soundness* that $can(S)(a) \equiv can(S)(b)$. ∎

Complexity. We have already seen in the termination argument that the number of iterations of *cc* in *process* is bounded by the number of distinct equivalence classes of terms in $dom(S)$ which is no more than the number of distinct uninterpreted terms. We will assume that the *solve* operation is performed by an oracle and that there is some fixed bound on the size of the solution set returned by it. In the case of linear arithmetic, the solution set has at most one element. Let n represent the number of distinct terms appearing in T which is also a bound on $|S|$ and on the size of the largest term appearing in S. The composition operation can be implemented in linear time. Thus the entire algorithm has $O(n^2)$ steps assuming that the σ and *solve* operations are length-preserving and ignoring the time spent inside *solve*.

6 Conclusions

Shostak's decision procedure for equality in the presence of interpreted and uninterpreted functions is seriously flawed. It is both incomplete and nonterminating, and hence not a decision procedure. All subsequent variants of Shostak's algorithm have been similarly flawed. This is unfortunate because decision procedures based on Shostak's algorithm are at the core of a number of widely used verification systems. We have presented a correct algorithm that captures Shostak's key insights, and described proofs of termination, soundness, and completeness.

Acknowledgments: We are especially grateful to Clark Barrett for instigating this work and correcting several significant errors in earlier drafts, and to Jean-Christophe Filliâtre for his oCaml implementation which yielded useful feedback on the algorithm studied here. The presentation has been substantially improved thanks to the suggestions of the anonymous referees and those of Nikolaj Bjørner, David Cyrluk, Bruno Dutertre, Ravi Hosabettu, Pat Lincoln, Ursula Martin, David McAllester, Sam Owre, John Rushby, and Ashish Tiwari.

References

[BDL96] Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201, Palo Alto, CA, November 1996. Springer-Verlag.

[Bjø99] Nikolaj Bjørner. *Integrating Decision Procedures for Temporal Verification.* PhD thesis, Stanford University, 1999.

[BRRT99] L. Bachmair, C. R. Ramakrishnan, I.V. Ramakrishnan, and A. Tiwari. Normalization via rewrite closures. In *International Conference on Rewriting Techniques and Applications, RTA '99*, Berlin, 1999. Springer-Verlag.

[BS96] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symbolic Computation*, 21:211–243, 1996.

[CLS96] David Cyrluk, Patrick Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction—CADE-13*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 463–477, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[DST80] P.J. Downey, R. Sethi, and R.E. Tarjan. Variations on the common subexpressions problem. *Journal of the ACM*, 27(4):758–771, 1980.

[EHD93] Computer Science Laboratory, SRI International, Menlo Park, CA. *User Guide for the* EHDM *Specification Language and Verification System, Version 6.1*, February 1993. Three volumes.

[FORS01] J-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated canonizer and solver. In *CAV 01: Computer-Aided Verification*. Springer-Verlag, 2001. To appear.

[Kap97] Deepak Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *International Conference on Rewriting Techniques and Applications, RTA '97*, number 1232 in Lecture Notes in Computer Science, pages 23–37, Berlin, 1997. Springer-Verlag.

[MT96] Zohar Manna and The STeP Group. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[NO80] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[ORS92] 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.

[Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, July 1978.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[SSMS82] R. E. Shostak, R. Schwartz, and P. M. Melliar-Smith. STP: A mechanized logic for specification and verification. In D. Loveland, editor, *6th International Conference on Automated Deduction (CADE)*, volume 138 of *Lecture Notes in Computer Science*, New York, NY, 1982. Springer-Verlag.