Deconstructing Shostak*

Harald Rueß and Natarajan Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
{ruess,shankar}@cs1.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, EHD, PVS, STeX, 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 type checking, verification conditions generated from Hoare triples, and inductive theorem proving. Shostak’s procedure is at the core of several verification systems including STP [SAMS82], EHD [EHD93], PVS [ORS92], STeX [MT96, BJ99], and SVC [BDL96]. The soundness of Shostak’s algorithm is reasonably straightforward, but its comple-

*This work was supported by SRI International, and by NSF Grant CCR-0482560, DARPA/AFRL Contract F33615-00-C-3043, and NASA Contract NAS1-0070.
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 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 \( y = x - 1, y - 1 = x - 1, y + 1 = x + 2, \) and so on. In order to avoid processing all these variations on the given equality, Shostak restricts his attention to \emph{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 \emph{canonizable}, then there is a canonizer \( \sigma \) 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,

\[
f(x + 1) - 1 = x + 1, \quad f(y) + 1 = y - 1, \quad y + 1 = x \vdash \text{false},
\]

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

\[
f(v) = v, \quad f(u) = u + 1, \quad u = v \vdash \text{false}.
\]

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 Cylwik, 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 [Bjo99] 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 $u, v, x, y, z$ 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 $\text{vars}(a)$, $\text{vars}(a = b)$, and $\text{vars}(T)$ 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 $\tau$, 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 $e$. $\text{solvables}(a)$ denotes the set of outermost non-interpreted subterms of $a$, i.e., those that do not occur uninterpreted in $a$.

$$\text{solvables}(f(a_1, \ldots, a_n)) = \bigcup_i \text{solvables}(a_i)$$

$\text{solvables}(a) = \{a\}$, otherwise

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

1. Axion: $\overline{T \vdash a = b}$, for $a = b \in T$.

2. Reflexivity: $\overline{T \vdash a = a}$

3. Symmetry: $\overline{T \vdash a = b \quad T \vdash b = a}$

4. Transitivity: $\overline{T \vdash a = b \quad T \vdash b = c \quad T \vdash a = c}$

5. Congruence: $\overline{T \vdash a_1 = b_1 \quad \ldots \quad T \vdash a_n = b_n \quad T \vdash f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)}$

The semantics for terms is given by a model $M$ over a domain $D$ and an assignment $\rho$ for the variables so that $M[\tau]_{\rho} = \rho(x)$ 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 $\rho$ over $\text{vars}(a = b)$. 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 $\tau$ [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 $\tau$. The canonizer and solver are first described for pure $\tau$-terms, i.e., without any uninterpreted function symbols, and then extended to uninterpreted terms by regarding these as variables.

Definition 3.1 A theory $\tau$ is canonizable if there is a canonizer $\sigma$ such that
1. \( \models a = b \text{ iff } \sigma(a) \equiv \sigma(b) \).

2. \( \sigma(x) \equiv x \).

3. \( \text{vars}(\sigma(a)) \subseteq \text{vars}(a) \).

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 \( \sigma \) 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 \( \sigma \)-model if \( M \models a = \sigma(a) \) for any term \( a \), and \( M \neq a = b \) for distinct canonical, variable-free terms \( a \) and \( b \).

**Definition 3.3** A set of equalities \( S \) and \( a = b \) are \( \sigma \)-equivalent iff for all \( \sigma \)-models \( M \) and assignments \( \rho \) over the variables in \( a \) and \( b \), \( M, \rho = a = b \) iff there is an assignment \( \rho' \) extending \( \rho \), 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\((a = b) = \perp \) if \( a = b \) is unsatisfiable in any \( \sigma \)-model, or \( S = \text{solve}(a = b) \) 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 \( i, 0 < i \leq n \),
   
   \( a \) \( x_i \in \text{vars}(a = b) \).
   
   \( b \) \( x_i \in \text{vars}(e_j) \), for \( j, 0 < j \leq n \).
   
   \( c \) \( x_i \neq x_j \) for \( i \neq j \) and \( 0 < j \leq n \).
   
   \( d \) \( \sigma(e_i) \equiv e_i \).

2. \( S \) and \( a = b \) are \( \sigma \)-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

\[
x_1 = \sigma( (d-c)/(a_1-b_1) ) + ((b_2-a_2)/(a_1-b_1)) \cdot x_2 + \ldots + ((b_n-a_n)/(a_1-b_1)) \cdot x_n).
\]

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 \( \gamma \) 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 \( \gamma \) 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 \( \gamma \), 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 \( \sigma \), a bijective map \( \gamma \) 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 \( \text{solveables}(a) \) instead of \( \text{vars}(a) \).

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

1. Canonization: \( T \vdash a = \sigma(a) \), for any term \( a \).

2. Solve: \( T \vdash a = b \) \( T \cup S \vdash c = d \) if \( S = \text{solve}(a = b) \neq \perp \) and \( \text{vars}(c = d) \subseteq \text{vars}(T) \).

3. Solve-\( \perp \): \( T \vdash a = b \) \( T \vdash \text{false} \), if \( \text{solve}(a = b) = \perp \).

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-\( \perp \). 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 strengthening 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 [B896] 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 \( \sigma \)-model \( M \) and assignment \( \rho \) over \( \text{vars}(T) \), \( M, \rho \models T \vdash a = b \).

Proof. By induction on the derivation of \( T \vdash a = b \). The soundness of the \textit{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 \textit{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., \( \text{dom}(S) \).

\[
S(a) = \begin{cases} 
  b & \text{if } a = b \in S \\
  a & \text{otherwise}
\end{cases}
\]

\[
\text{dom}(S) = \{ a \mid \exists b. a = b \in S \}.
\]

The operation \( a \xrightarrow{\hat{\circ}} b \) 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 \( \text{dom}(S) \) such that \( a \xrightarrow{\hat{\circ}} b \), we have \( S(a) \equiv S(b) \).

\( S[a] \) replaces a subterm \( b \) in \( S(b) \), where \( b \in \text{solvable}(a) \).

\[
S[f(a_1, \ldots, a_n)] = f(S[a_1], \ldots, S[a_n]),
\]

if \( f \) is interpreted

\[
S[a] = S(a), \text{ otherwise}
\]

\( \text{norm}(S)(a) \) is a normal form for \( a \) with respect to \( S \) and is defined as \( \sigma(S[a]) \). The operation \( \text{norm} \) 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 \( \text{norm}(S)(a) \).

\[
\text{norm}(S)(a) = \sigma(S[a]).
\]

Lemma 3.8 If \( \text{solve}(a = b) = S \not\equiv \perp \), then \( \text{norm}(S)(a) \equiv \text{norm}(S)(b) \).

Proof. By definitions 3.3 and 3.4(2), for any \( \sigma \)-model \( M \) and assignment \( \rho \), 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 \( \sigma \)-model, by Definition 3.2, it must be the case that \( \sigma(a') \equiv \sigma(b') \), and therefore \( \text{norm}(S)(a) \equiv \text{norm}(S)(b) \).

The definition of the \textit{lookup} operation uses Hilbert’s epsilon operator, indicated by the keyword \textit{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 \( \text{lookup}(S)(a) \) returns \( a \) itself. We show later that the \textit{lookup} operation is used only when the results of this choice are deterministic.

\[
\text{lookup}(S)(f(a_1, \ldots, a_n)) = S(f(b_1, \ldots, b_n)),
\]

when \( b_1, \ldots, b_n : \)

\[
f(b_1, \ldots, b_n) \in \text{dom}(S),
\]

and \( a_i \equiv S(b_i) \), for \( 1 \leq i \leq n \)

\[
\text{lookup}(S)(a) = a, \text{ otherwise}
\]

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

\[
\text{can}(S)(f(a_1, \ldots, a_n)) = \text{lookup}(S)(f(\pi_1, \ldots, \pi_n)),
\]

if \( f \) is uninterpreted

\[
\text{can}(S)(f(a_1, \ldots, a_n)) = \sigma(f(\pi_1, \ldots, \pi_n)),
\]

if \( f \) is interpreted

\[
\text{can}(S)(a) = S(a), \text{ otherwise}
\]

Lemma 3.9 (\( \sigma \)-normal) If \( S \) is functional, then \( \text{norm}(S)(\sigma(a)) \equiv \hat{a} \) and \( \text{can}(S)(\sigma(a)) \equiv \pi \).

Proof. We know that \( \models \sigma(a) = a \). Then for \( a' = S[\sigma(a)] \) and \( b = S[a] \), the equality \( a' = b \) is valid in every \( \sigma \)-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 \( R = \{ b = \pi \mid b \in [a] \} \), then \( \text{can}(S)(a) \equiv \text{norm}(R)(a) \). 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 \textit{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 (\( \sigma \)-normal decomposition) If \( R \cup S \) is functional, then

\[
\text{norm}(R \circ S)(a) \equiv \text{norm}(S)(\text{norm}(R)(a)).
\]
\[
\text{process}\{\{a = b, T\}\} = \text{assert}(a = b, \text{process}(T))
\]

\[
\text{process}(\emptyset) = \emptyset.
\]

\[
\text{assert}(a = b, \perp) = \perp
\]

\[
\text{assert}(a = b, S) = \sum(\text{merge}(\pi, \overline{\pi}, S^+)), \text{ where, } S^+ = \text{expand}(S, \pi, \overline{\pi}).
\]

\[
\text{expand}(S, a, b) = S \cup \{e = e \mid e \in \text{new}(S, a, b)\}.
\]

\[
\text{new}(S, a, b) = [a = b] - \text{dom}(S).
\]

\[
\text{merge}(a, b, S) = \perp, \text{ if } \text{solve}(a = b) = \perp
\]

\[
\text{merge}(a, b, S) = S \circ \text{solve}(a = b), \text{ otherwise.}
\]

\[
\sum(\perp) = \perp
\]

\[
\sum(S) = \sum(\text{merge}(S(a), S(b), S)), \text{ when } a, b:
\]

\[
a, b \in \text{dom}(S)
\]

\[
a \sim b, \text{ and } S(a) \neq S(b)
\]

\[
\sum(S) = S, \text{ otherwise.}
\]

**Figure 1:** Main Procedure: \text{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 \(\text{can}(S(c)) \equiv \text{can}(S(d))\), where \(S = \text{process}(T)\). The \text{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 \(\text{dom}(S)\) are in the same equivalence class according to \(S\) if \(S(a) \equiv S(b)\).

The operation \text{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 \text{assert}(a = b, S), where \(S\) is the state as returned by the recursive call of \text{process}. The invocation of \text{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 \(\overline{a}\) where \(e \in \text{dom}(S)\). This preprocessing step ensures that \(S\) contains entries corresponding to any terms that might be needed in the congruence closure phase in the operation \sum. The \text{merge} operation then solves the equality \(\pi = \overline{\pi}\) to get a solution \(S'\), 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 \text{cc}(S) computes the congruence closure of \(S\) by repeatedly picking a pair of congruent terms \(a\) and \(b\) from \(\text{dom}(S)\) such that \(\sum(S(a)) \neq S(b)\) and merging them using \text{merge}(S(a), S(b), S). Eventually either a contradiction is found or all congruent left-hand sides in \(S\) are merged and the \sum operation terminates returning a congruence-closed solution set.

The above algorithm fixes the nontermination and incompleteness problems in Shostak’s algorithm by introducing the \text{norm} operation and the composition operator \(R \circ S\) to fold in a solution. The \text{norm} operation ensures that no new uninterpreted terms are introduced during congruence closure in the function \sum, 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 \(f^3(x) = x, f^5(x) = x \vdash f(x) = x\). The value of \(S\) after the base case is \(\emptyset\). After the preprocessing of \(f^3(x) = x\) in \text{assert}, \(S\) is \(\{x = x, f(x) = f(x), f^3(x) = f^3(x), f^5(x) = f^5(x)\}\). After merging \(f^3(x)\) and \(x\), \(S\) is \(\{x = x, f(x) = f(x), f^3(x) = f^3(x), f^5(x) = f^5(x)\}\). When \(f^5(x) = x\) is preprocessed in \text{assert}, \(\text{can}(S(f^5(x)))\) yields \(f^2(x)\) since \(S(f^5(x)) \equiv x\), 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 \(f(x) \not\sim f^2(x)\) and hence \(f(x)\) and \(x\) are merged so that \(S\) is now \(\{x = x, f(x) = x, f^3(x) = x\}\).
The conclusion \( f(x) = x \) easily follows since \( \text{can}(S)(f(x)) \equiv x \equiv \text{can}(S)(x) \).

**Example 4.2** Consider \( y + 1 = x \), \( f(y) + 1 = y - 1 \), \( f(x - 1) - 1 = x + 1 \) is 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

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

The congruence close operation \( \text{cc} \) detects the congruence \( f(1 - y) \sim f(x) \) and invokes \text{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 \text{merge} operation is clearly the workhorse of the procedure since it is invoked from within both \text{assert} and \text{cc}. Let \( U(X) \) represent the set \( \{ a \in X \mid \text{a uninterpreted} \} \) of uninterpreted terms in the set \( X \). Let \( A \) be \text{solvables}(a), \( B \) be \text{solvables}(b), and \( S' = \text{merge}(a, b, S) \), then assuming \( U(A \cup B) \subseteq \text{dom}(S) \) 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 \( a \in \text{dom}(S) \), \( [a] \subseteq \text{dom}(S) \).
3. **Range closure:** \( S \) is range-closed if for any \( a \in \text{dom}(S) \), \( \text{U(solvables}(S(a)) \subseteq \text{dom}(S) \), and for any \( c \in \text{solvables}(S(a)) \), \( S(c) \equiv c \).
4. **Norm closure:** \( S \) is norm-closed if \( S(a) \equiv \text{norm}(S)(a) \) for \( a \) in \( \text{dom}(S) \). This of course holds trivially for uninterpreted terms \( a \).

These properties can be easily established by inspection. Since whenever \text{merge}(a, b, S) \) is invoked in the algorithm, the arguments do satisfy the conditions \( U(A \cup B) \subseteq \text{dom}(S) \) and for all \( c \in A \cup B \), \( S(c) \equiv c \), it then follows that these properties are also preserved by \text{assert} and \text{cc}, and therefore hold of \text{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 \( A = \text{solvables}(a) \) and \( B \equiv \text{solvables}(b) \). Given that \( U(A \cup B) \subseteq \text{dom}(S) \) and for all \( c \in A \cup B \), \( S(c) \equiv c \), if \( S' = \text{merge}(a, b, S) \neq \perp \), then

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

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

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

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

\[
E(S)(a) = \{ b \in U(\text{dom}(S)) \mid S(b) \equiv S(a) \} \\
P(S) = \{ E(S)(a) \mid a \in U(\text{dom}(S)) \} \\
\#(S) = \#P(S)
\]
The definition of $cc(S)$ terminates because the measure $\#(S)$ decreases with each recursive call. If in the definition of $cc$, $merge(S(a), S(b), S) = ⊥$, then clearly $cc$ terminates. Otherwise, let $S' = merge(S(a), S(b), S) ≠ ⊥$, for $a$ and $b$ in $dom(S)$ such that $S(a) ≠ S(b)$ and $a ≈ b$. In this case $a$ and $b$ must be uninterpreted terms since for interpreted terms $a$ and $b$, if $a ≈ b$, then $S(a) ≡ S(b)$ by norm closure. By merge equivalence, $norm(S')(S(a)) ≡ norm(S')(S(b))$ and $U(dom(S')) = U(dom(S))$. By monotonicity, for any $c$ and $d$ such that $S(c) ≡ S(d)$, we have $S'(c) ≡ S(d)$, and therefore $\#(S') ≤ \#(S)$. However, by norm closure, $S'(a) ≡ S'(b)$ 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 $vars(a) ⊆ vars(T ∪ S)$, then $T, S ⊢ a = a'$ is derivable, for $a' ≡ S[a]$.

**Lemma 5.3 (norm soundness)**
If $vars(a) ⊆ vars(T ∪ S)$, then $T, S ⊢ a = ̄a$ is derivable.

**Lemma 5.4 (can soundness)**
If $vars(a) ⊆ vars(T ∪ S)$, then $T, S ⊢ a = ̄a$ is derivable.

**Lemma 5.5 (merge soundness)**
If $S' = merge(a, b, S) ≠ ⊥$, then if $T, S ⊢ a = b$, and $T, S' ⊢ c = d$ with $vars(c = d) ⊆ vars(T ∪ S)$, then $T, S ⊢ c = d$. Otherwise, $merge(a, b, S) = ⊥$, and $T, S ⊢ ⊥$.

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

If $merge(a, b, S) = ⊥$, then by similar reasoning using the solve-⊥ rule, $T, S ⊢ ⊥$ is derivable.

**Lemma 5.6 (cc soundness)**
If $S' = cc(S) ≠ ⊥$, $T, S' ⊢ a = b$ for $vars(a = b) ⊆ vars(T, S)$, then $T, S ⊢ a = b$ is derivable. Otherwise, $cc(S) = ⊥$, and $S ⊢ ⊥$ is derivable.

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

**Lemma 5.7 (process soundness)**
If $S = process(T_i) ≠ ⊥$, $T_1 ⊆ T_2$, and $T_2, S ⊢ c = d$ for $vars(c = d) ⊆ vars(T_2)$, then $T_2 ⊢ c = d$ is derivable. Otherwise, $process(T_i) = ⊥$, and $T_1 ⊢ false$ is derivable.

**Proof.** By induction on the length of $T_i$. In the base case, $S$ is empty and the theorem follows trivially. In the induction step, with $T_i = \{a = b, T'_i\}$ and $S' = process(T'_i)$, we have the induction hypothesis that $T_2 ⊢ c = d$ is derivable if $T_2, S' ⊢ c = d$ is derivable, for any $c, d$ such that $vars(c = d) ⊆ vars(T_2)$. We know by can soundness that $T_2, S' ⊢ \overrightarrow{a} = \overrightarrow{b}$ and $T_2, S' ⊢ \overrightarrow{b} = \overrightarrow{a}$ are derivable. When $S'$ is augmented with identities over subterms of $\overrightarrow{a}$ and $\overrightarrow{b}$ to get $S'^+$, we have the derivability of $T_2, S'^+ ⊢ c = d$. By cc soundness, we then have the derivability of $T_2, S'^+ ⊢ c = d'$ from that of $T_2, S ⊢ c = d$. The derivability of $T_2, S' ⊢ c = d$ then follows by cut from that of $T_2, S'^+ ⊢ c = d$, and we get the conclusion $T_2 ⊢ c = d$ by the induction hypothesis.

A similar induction argument shows that when $process(T_i) = ⊥$, then $T_2 ⊢ false$.

**Theorem 5.8 (soundness)**
If $S = process(T) ≠ ⊥$, $vars(a = b) ⊆ vars(T)$, and $\overrightarrow{a} ⊆ \overrightarrow{b}$, then $T ⊢ a = b$ is derivable. Otherwise, $process(T) = ⊥$, and $T ⊢ false$ is derivable.

**Proof.** If $S = process(T) ≠ ⊥$, then by can soundness, $T, S ⊢ a = \overrightarrow{a}$ and $T, S ⊢ b = \overrightarrow{b}$ are derivable. Hence, by transitivity and symmetry, $T, S ⊢ a = b$ is derivable. Therefore, by process soundness, $T ⊢ a = b$ is derivable.

If $process(T) = ⊥$, then already by process soundness, $T ⊢ false$.

**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 $U([a]) ⊆ dom(S)$, then $can(S)(a) ⊆ norm(S)(a)$.

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

If $a$ is uninterpreted and of the form $f(a_1, ..., a_n)$, then $can(S)(a) ≡ lookup(S)(f(\overrightarrow{a}_1, ..., \overrightarrow{a}_n))$. Since $S$ is subterm-closed, by the induction hypothesis and norm closure, we have $\overrightarrow{a}_i ≡ \overrightarrow{a}_i ≡ S(a_i)$ for $0 < i ≤ n$. 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, $\sigma \equiv S(b) \equiv S(a) \equiv \hat{a}$, since $a \cong b$.

If $a$ is interpreted, by the induction hypothesis and substitution closure, $\sigma \equiv \sigma(f(\overline{a_1}, \ldots, \overline{a_n})) \equiv \sigma(f(\overline{\hat{a}_1}, \ldots, \overline{\hat{a}_n})) \equiv \hat{a}.

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

Proof. By induction on $a$. When $a$ is a variable, $\text{can}(S)(a) \equiv S(a)$. If $a \not\in \text{dom}(S)$, then $S(a) = a$, and hence the conclusion.

Otherwise, by range-closure, $\text{dom}(S) \subseteq \text{dom}(S')$. Then, by confluence, norm decomposition, and idempotence, $\text{can}(S')(S(a)) \equiv \text{norm}(S')\text{can}(S)(a) \equiv \text{norm}(R)\text{can}(S)(S(a)) \equiv \text{norm}(R)\text{can}(S')(a) \equiv \text{can}(S')(a)$.

In the induction step, let $a \equiv f(a_1, \ldots, a_n)$. If $a$ is uninterpreted, then if $f(b_1, \ldots, b_n) \in \text{dom}(S)$, then $\sigma \equiv S'(f(b_1, \ldots, b_n))$. The reasoning used in the base case can then be repeated to derive the conclusion.

Otherwise, $\sigma \equiv f(\overline{a_1}, \ldots, \overline{a_n})$ and by the induction hypothesis and the definition of $\text{can}$, $\text{can}(S')(\sigma) \equiv \text{lookup}(S')(f(\text{can}(S')(a_1), \ldots, \text{can}(S')(a_n))) \equiv \text{can}(S')(a)$.

When $a$ is interpreted, by the induction hypothesis and the $\sigma$-norm lemma,

\[
\text{can}(S')(\sigma) \\
\equiv \text{can}(S')(\sigma(f(\overline{a_1}, \ldots, \overline{a_n}))) \\
\equiv \sigma(f(\text{can}(S')(\overline{a_1}), \ldots, \text{can}(S')(\overline{a_n}))) \\
\equiv \text{can}(S')(\sigma).
\]

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

Lemma 5.11 (can $\sigma$-model) If $S = \text{process}(T) \neq \bot$ and $X = \text{vars}(T)$, then $M_S, \rho_X^T \models a = b$ for any $a = b \in T$.

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

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

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

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

Proof. Since $M_S, \rho_X^T \models T$ by can $\sigma$-model for $X = \text{vars}(T)$, we have by proof soundness that $\text{can}(S)(a) \equiv \text{can}(S)(b)$.

Complexity. We have already seen in the termination argument that the number of iterations of $\omega$ in process is bounded by the number of distinct equivalence classes of terms in $\text{dom}(S)$ 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 $\sigma$ 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 non-terminating, 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


