SRI International

CSL Technical Report • April, 2002

Deconstructing Shostak (Draft)

Harald Rueß
Natarajan Shankar
SRI International

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.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (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.

Contents

Chapter 1

Introduction

Ground decision procedures (GDPs) are at the core of many modern theorem proving and verification systems. Such decision procedures decide the validity of quantifier-free formulas where the free variables are implicitly universally quantified over the entire formula. The typical proof obligations that arise in extended type-checking, verification, and theorem proving, are in the ground fragment. The atomic formulas in such proof obligations often involve equalities and inequalities between terms containing a mixture of uninterpreted function symbols and function symbols with fixed interpretations from a combination of theories such as arithmetic, arrays, datatypes, and set operations. Efficient ground decision procedures for this fragment are therefore a topic of both theoretical and practical significance.

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. Variants of Shostak's procedure are employed by a number of 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. In this report, we fix these flaws in Shostak's original paper. We explain the ideas underlying Shostak's decision procedure by presenting a correct version of the algorithm along with detailed and 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*, namely, 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 $y + 1 = x$.

Nelson and Oppen [NO79] showed how decision procedures for *disjoint* equational theories, i.e., theories with no interpreted functions in common, 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*, that is, 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 the advantage of being able to combine individual decision procedures (of a specific form) for disjoint theories, but it also 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 procedure does not exploit any efficiencies that can be gained from the specific characteristics of the theories involved

Shostak's algorithm does not combine decision procedures but is instead a single decision procedure for the combination of a class of theories. It gains 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. We describe the shortcomings of Shostak's original algorithm and its many incorrect variants by examining some very simple examples where these algorithms can fail. 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 the 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.

Shostak also claimed that the individual canonizers and solvers for two disjoint theories could easily be combined to yield a canonizer and solver for their union. This claim is easily verifiable for canonizers since when canonizing a term of the form $f(a_1, \ldots, a_n)$ where f is in theory i for $i \in \{1, 2\}$, we use σ_i, the canonizer corresponding to theory i. The method for combining solvers given there is incorrect and unworkable in any variation. We have in fact solved the problem of combining solvers, but this topic lies outside the scope of the present paper. We focus here on the important case of a ground decision procedure for a single canonizable, solvable theory combined with the theory of pure equality.

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.

This report is an expanded version of a conference paper [RS01] with the same title. In this report, 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 when solved. 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.

Ford and Shankar [FS02] have formally verified the algorithm and proof presented here using the PVS verification system [ORS92]. This verification was very helpful in the development of the expanded proofs here. The formal verification was based on the earlier publication [RS01] and revealed a few minor gaps in that presentation. In particular, one lemma was found to need an antecedent condition, but this correction had no impact on the rest of the proof since only this weaker lemma was actually used.

Chapter 2.1 introduces the theory of equality, which is augmented in Chapter 2.3 with function symbols from a canonizable and solvable theory. Chapter 2.3 also introduces the basic building blocks for the decision procedure. The algorithm itself is described in Chapter 3 along with some example hand-simulations. The proofs of termination, soundness, and completeness are outlined in Chapter 4.

Chapter 2

Preliminaries

We introduce the background and basic operations needed to present the combination algorithm and its proof.

2.1 Background

This section introduces the basic terminology. 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, a, b, c, d, and e range over terms, and M and N range over sets of 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 $\mathit{vars}(a)$, $\mathit{vars}(a = b)$, and $\mathit{vars}(T)$ 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.

A finite map from a set of terms to some range R is represented as $\{a_1 \mapsto v_1, \ldots, a_n \mapsto v_n\}$. The set $\{a_1, \ldots, a_n\}$ is the domain of the map. The application of a map ρ as $\rho(a)$ returns a itself when a is not in the domain of the map ρ, and otherwise returns the value v corresponding to the a in ρ. The map $\rho\{a_1 \mapsto v_1, \ldots, a_n \mapsto v_n\}$ returns v_i when applied to a_i, but returns $\rho(a)$ when applied to some a that is distinct from each of the a_i for $1 \leq i \leq n$. When the range R is just the set of terms, then the operation of *replacement* with respect to a map ρ and a term a is written as $\rho[a]$ and is defined to be $\rho(a)$ if a is in the domain of ρ or is a variable. Otherwise, $\rho[a]$ is $f(\psi[a_1], \ldots, \psi[a_n])$ where $a \equiv f(a_1, \ldots, a_n)$. A *substitution* is a finite map from a set of variables to a set of terms.

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. $\mathit{solvables}(a)$ denotes the set of outermost

Axiom: (for $a = b \in T$)

$$\frac{}{T \vdash a = b}$$

Reflexivity:

$$\frac{}{T \vdash a = a}$$

Symmetry:

$$\frac{T \vdash a = b}{T \vdash b = a}$$

Transitivity:

$$\frac{T \vdash a = b \qquad T \vdash b = c}{T \vdash a = c}$$

Congruence:

$$\frac{T \vdash a_1 = b_1 \quad \ldots \; T \vdash a_n = b_n}{T \vdash f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)}$$

Figure 2.1: Inference rules for equality.

non-interpreted subterms of a, i.e., those that do not occur uninterpreted in a.

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

2.2 The Theory of Equality

Proof Theory. The theory of equality deals with sequents of the form $T \vdash a = b$. We will insist that these sequents be such that $\mathit{vars}(a = b) \subseteq \mathit{vars}(T)$. Figure 2.1 gives the proof theory for equality.

Semantics. The semantics for terms is given by a model M over a domain D and an assignment ρ for the variables so that $M[\![a]\!]\rho \in D$ for all a, and the following equations hold.

$$\begin{array}{rcl} M[\![x]\!]\rho & = & \rho(x) \\ M[\![f(a_1, \ldots, a_n)]\!]\rho & = & M(f)(M[\![a_1]\!]\rho, \ldots, M[\![a_n]\!]\rho) \end{array}$$

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 $\mathit{vars}(a = b)$. 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 $(M, \rho \models T) \supset (M, \rho \models a = b)$.

Decidability. The congruence closure decision procedure for equality decides judgements of the form $T \vdash a = b$ by partitioning the term universe (the set of subterms of T, a, and b) into equivalence classes generated from T [Koz77, NO80, Sho78, DST80]. The partitioning is *congruence closed* if $f(a_1, \ldots, a_n)$ and $f(b_1, \ldots, b_n)$ are in the same equivalence class whenever for each i, $1 \leq i \leq n$, a_i and b_i are in the same equivalence class. Congruence closure is subsumed by the combination decision procedure given in Chapter 3.

2.3 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 theory τ is thus specified in terms of an oracle for deciding the validity of equalities in this theory.

The canonizer and solver are first described for pure τ-terms, i.e., without any uninterpreted function symbols, in this section. The next section shows how these operations can be extended to impure terms by regarding the uninterpreted subterms as variables. Note that in this section, all terms are pure.

Definition 2.1 *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 2.2 *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. If $M \models a = b$ for σ-model M, then $a = b$ is σ-*valid.

Lemma 2.3 (σ-distributivity) $\sigma(f(\sigma(a_1), \ldots, \sigma(a_n))) \equiv \sigma(f(a_1, \ldots, a_n))$.

Proof. By Definition 2.1(4), for any i, $1 \leq i \leq n$, $\sigma(\sigma(a_i)) \equiv \sigma(a_i)$. By Definition 2.1(1), $\models_\tau \sigma(a_i) = a_i$, and hence $\models_\tau f(\sigma(a_i), \ldots, \sigma(a_n)) = f(a_1, \ldots, a_n)$, and therefore by Definition 2.1(1), $\sigma(f(\sigma(a_1), \ldots, \sigma(a_n))) \equiv \sigma(f(a_1, \ldots, a_n))$. ∎

Lemma 2.4 (substitutivity) *For any substitution ψ mapping variables to terms, if $\models a = b$, then $\models \psi[a] = \psi[b]$.*

Proof. Let ψ be of the form $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. For a given ρ, let ρ' be $\rho\{x_1 \mapsto M[\![t_1]\!]\rho, \ldots, x_n \mapsto M[\![t_n]\!]\rho\}$. Then $M[\![\psi(a)]\!]\rho = M[\![a]\!]\rho'$, and similarly, $M[\![\psi[b]]\!]\rho = M[\![b]\!]\rho'$, and hence $M \models \psi[a] = \psi[b]$. ∎

Lemma 2.5 (σ-substitutivity) *For any a, $\sigma(\psi[\sigma(a)]) \equiv \sigma(\psi[a])$.*

Proof. Follows from Lemma 2.4 and Definition 2.1(1). ∎

Lemma 2.6 (σ-validity) *If σ is a canonizer for τ, then $\models_\tau a = b$ iff $a = b$ is σ-valid.*

Proof. If $\models_\tau a = b$, then by Definition 2.1(1), $\sigma(a) \equiv \sigma(b)$. Since in all σ-models M, $M \models \sigma(a) = a$ and $M \models \sigma(b) = b$, we have $M \models a = b$.

Conversely, if in all σ-models M and assignments ρ, $M, \rho \models a = b$, then we can show that $\sigma(a) \equiv \sigma(b)$. Let the domain D consist of the canonical terms c such that $\sigma(c) \equiv c$. The metavariable $\mathbf{c}$ ranges over such canonical terms. Let M_σ be defined so that $M_\sigma(f)(\mathbf{c_1}, \ldots, \mathbf{c_n}) = \sigma(f(\mathbf{c_1}, \ldots, \mathbf{c_n}))$. To show that M_σ is a σ-model, we first observe that $M_\sigma[\![a]\!]\rho \equiv \sigma(\rho(a))$ by induction on a using Lemma 2.6. By Lemma 2.5, $\sigma(\rho(\sigma(a))) \equiv \sigma(\rho(a))$, hence $M_\sigma[\![\sigma(a)]\!]\rho \equiv M_\sigma[\![a]\!]\rho$ and M_σ is a σ-model.

If we let ρ_σ be defined so that $\rho_\sigma(x) = x$, it is easy to see that for any d, $M_\sigma[\![d]\!]\rho_\sigma = \sigma(d)$, by induction on d and Lemma 2.3. Since $M_\sigma, \rho_\sigma \models a = b$, we have $\sigma(a) \equiv \sigma(b)$. ∎

When σ is a canonizer for theory τ, Lemma 2.6 allows us to replace $\models_\tau a = b$ with the σ-validity of $a = b$. We can, without loss of generality, replace Definition 2.1(1) by the conditions σ-distributivity and σ-substitutivity verified in Lemmas 2.3 and 2.5.

Definition 2.7 *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 2.8 *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

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

In general, $solve(a = b)$ may contain variables that do not occur in $a = b$, and vice-versa. A solved form of the equation $\pi_1(x) = y$ in the theory of pairs, for example, is given by the equation $x = (y, d)$, where d is a fresh constant.

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.

2.4 Canonization and Solving with Uninterpreted Terms

The solvers and canonizers characterized above are intended to work in the absence of uninterpreted function symbols. They can be 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 map between a set of variables X that do not appear in a and the uninterpreted subterms of 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

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

For example,

$$\begin{array}{ll} & solve(f(v-1) - 1 = v + 1) \\ \equiv & \gamma[solve(u - 1 = v + 1)] \\ \equiv & \gamma[u = v + 2] \\ \equiv & (f(v-1) = v + 2) \end{array}$$

where f is uninterpreted and $\gamma = \{u \mapsto f(v-1)\}$. When uninterpreted terms are handled as above, the conditions in Definitions 2.1 and 2.8 must be suitably adapted by using $solvables(a)$ instead of $vars(a)$. With this replacement, Definitions 2.2 and 2.7 remain otherwise unchanged. Lemmas 2.3 and 2.6 remain valid for the expanded definition of σ.

2.5 Proof Theory for Equality in Canonizable and Solvable Theories

Recall that a sequent $T \vdash a = b$ is well-formed only when $vars(a = b) \subseteq vars(T)$. The proof theory for equality is augmented for canonizable, solvable theories by the proof rules:

Canonization: for any term a,

$$\frac{}{T \vdash a = \sigma(a)}$$

Solve: If $S = solve(a = b) \neq \bot$ and $vars(c = d) \subseteq vars(T)$, then:

$$\frac{T \vdash a = b \quad T \cup S \vdash c = d}{T \vdash c = d}$$

Solve-$\bot$: If $solve(a = b) = \bot$, then:

$$\frac{T \vdash a = b}{T \vdash false}$$

Definition 2.9 (derivability) *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-$\bot$.*

Sometimes, we simply say $T \vdash c = d$ instead of $T \vdash c = d$ is derivable. 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.

Lemma 2.10 (weakening) *If $T \subseteq T'$ and $T \vdash a = b$ is derivable, then $T' \vdash a = b$ is derivable.*

Proof. By induction on the derivation of $T \vdash a = b$. ■

Lemma 2.11 (cut) *If $T' \vdash T$ and $T \vdash a = b$ is derivable, then $T' \vdash a = b$ is derivable.*

Proof. By induction on the derivation of $T \vdash a = b$. ■

Theorem 2.12 (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 case of the *solve* and solve-$\bot$ rules are interesting. If $T \vdash a = b$ is the conclusion of an application of the *solve* rule, then for some c and d, we have a subderivation of $T \vdash c = d$ and $T, S \vdash a = b$, where $S = solve(c = d)$. Given that $M, \rho \models T$, we have by the induction hypothesis that $M, \rho \models c = d$. By the condition on *solve*, there is some extension ρ' of ρ such that $M, \rho' \models T, S$. Then, once again by the induction hypothesis, $M, \rho \models a = b$, but since $vars(a = b) \subseteq vars(T)$, we have $M, \rho \models a = b$, and hence the conclusion.

The soundness of the *solve*-$\bot$ rule can be similarly proved. ■

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., $dom(S)$.

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

Note that it may be the case that $S(a) \equiv a$ for a in the domain $dom(S)$.

The terms a and b are said to be *congruent in* S, symbolically, $a \overset{S}{\sim} b$, if

$$\begin{array}{rcl} a & \equiv & f(a_1, \ldots, a_n) \\ b & \equiv & f(b_1, \ldots, b_n) \\ S(a_i) & \equiv & S(b_i) \text{ for } 1 \leq i \leq n \ . \end{array}$$

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 $b \in solvables(a)$.

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

The term computed by $norm(S)(a)$ 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. For a fixed S, we drop the reference to S and use $\hat{a}$ as a syntactic abbreviation for $norm(S)(a)$.

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

Lemma 2.13 *If solve$(a = b) = S \neq \bot$, then norm$(S)(a) \equiv$ norm$(S)(b)$.*

Proof. By Definitions 2.7 and 2.8(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 2.2, 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.

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

$can(S)(a)$ 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. For a fixed S, we use $\overline{a}$ as a syntactic abbreviation for $can(S)(a)$.

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

Lemma 2.14 (σ-norm) *If S is functional, then $norm(S)(\sigma(a)) \equiv norm(S)(a)$ and $can(S)(\sigma(a)) \equiv can(S)(a)$.*

Proof. The first part is equivalent to $\sigma(S[\sigma(a)]) \equiv \sigma(S[a])$ which is a trivial consequence of Lemma 2.5.

For the second part, 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. Whenever $R \circ S$ 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.

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

The following lemmas about composition are given without proof.

Lemma 2.15 (norm decomposition) *If $R \circ S$ is functional, then*

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

Lemma 2.16 (associativity of composition) *If $Q \circ R \circ S$ is functional, then*

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

Lemma 2.17 (monotonicity) *If $R \circ S$ is functional, then for any a and b in $dom(R)$:*

$$R(a) \equiv R(b) \ \textit{implies}\ (R \circ S)(a) \equiv (R \circ S)(b)$$

Chapter 3

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 3.1, 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 $process(T)$ defined in Figure 3.1 returns a final solution set by starting with an empty solution set and successively processing each equality $a = b$ in T. The steps in the algorithm are:

1. Let T' be $T - \{a = b\}$. First, S is computed as $process(T')$.

2. Then $assert(a = b, S)$ is computed. If $S = \bot$, then $assert(a = b, S)$ returns $\bot$. Otherwise, the canonical forms $\overline{a}$ and $\overline{b}$ of a and b with respect to S are computed. This step identifies the subterms of $a = b$ that are known to be equal with respect to S.

3. S^+ is computed by adding to S the reflexivity equalities $e = e$ for any subterms e of $\overline{a} = \overline{b}$ that are not already in $dom(S)$.[1] 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.

4. The *merge* operation then solves the equality $\overline{a} = \overline{b}$ to get a solution S', and

[1]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. Going even further, the interpreted subterms can completely eliminated from $dom(S)$ if $f(a_1, \ldots, a_n) \overset{S}{\sim} f(b_1, \ldots, b_n)$ is redefined to hold exactly when $\hat{a_i} \equiv \hat{b_i}$ for $0 < i \leq n$. This variant of the definition is employed in the version of the algorithm that is formally verified in PVS [FS02].

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

Figure 3.1: Main Procedure: *process*

returns $S \circ S'$ as the new value for the state S.[2] If S' is $\bot$, then so is $S \circ S'$. Otherwise, the new solution set $S \circ S'$ affirms $a = b$ (i.e., $\mathit{norm}(S \circ S')(a) \equiv \mathit{norm}(S \circ S')(b)$) but it might not be congruence-closed.

5. Finally, $\mathit{cc}(S \circ S')$ is invoked to compute the congruence closure of the resulting solution set. The operation $\mathit{cc}(S)$ repeatedly picks a pair of congruent terms c and d from $\mathit{dom}(S)$ (i.e., $c \overset{S}{\sim} d$) such that $S(c) \not\equiv S(d)$. The selection of congruent pairs of left-hand side terms uses the Hilbert choice operator. The corresponding right-hand sides $S(c)$ and $S(d)$ are then merged using $\mathit{merge}(S(c), S(d), S)$. 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.

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

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

Example 3.1 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*, $\mathit{can}(S)(f^5(x))$ 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

$$\{x = x, f(x) = x, f^2(x) = x, f^3(x) = x\}.$$

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

Example 3.2 Consider the goal

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

which is a permutation of our earlier example. Starting with $S = \emptyset$ 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

$$\begin{array}{l}\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x,\\ \quad f(-1 + x) = f(-1 + x), 1 + x = 1 + x\}.\end{array}$$

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

$$\begin{array}{l}\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x,\\ \quad f(-1 + x) = 2 + x, 1 + x = 1 + x\}.\end{array}$$

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

$$\begin{array}{l}\{\ 1 = 1, -1 = -1, x = x, -1 + x = -1 + x,\\ \quad f(-1 + x) = 2 + x, 1 + x = 1 + x, y = y,\\ \quad f(y) = -2 + y, -1 + y = -1 + y,\\ \quad 1 + f(y) = -1 + y\}.\end{array}$$

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

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

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.

Chapter 4

Analysis

We describe the proofs of termination, soundness, and completeness, and also present a complexity analysis. The termination of *process* hinges on that of *cc*. The number of uninterpreted terms in $dom(S)$ stays fixed during the execution of *cc*. With each recursive call, a pair of distinct equivalence classes each containing an uninterpreted term from $dom(S)$ are merged. Since there is a bound on the number of pairs of uninterpreted terms, this bounds the number of times the *merge* operation can be invoked from *cc*. The proof of correctness is given by observing that the following are equivalent:

1. If $process(T) = S$, then $S = \bot$ or $can(S)(a) \equiv can(S)(b)$.
2. $T \vdash a = b$ is derivable.
3. $T \vdash a = b$ is σ-valid, i.e., valid in all σ-models.

The implication between (1) and (2) captures soundness (Theorem 4.9). It is proved by showing that for every equality $c = d \in S$, the sequent $T \vdash c = d$ is derivable, and the the sequent $S \vdash e = \overline{e}$ is derivable for any term e. The implication between (2) and (3) captures proof soundness (Theorem 2.12) and is verified by induction on proofs. The implication between (3) and (1) captures completeness (Theorem 4.17). It is verified by constructing for $S = process(T) \neq \bot$, a σ-model M_S and an assignment ρ_S such that $M_S[\![a]\!]\rho_S \equiv can(S)(a)$ and showing that for any equality $c = d \in T$, $can(S)(c) \equiv can(S)(d)$.

Key Invariants. The *merge* operation is clearly the workhorse of the procedure since it is invoked from within both *assert* and *cc*. Let

$$U(X) = \{a \in X \mid a \text{ uninterpreted}\}$$

of uninterpreted terms in the set X. For a, b, and S be such that $U(A \cup B) \subseteq dom(S)$ and for all $c \in A \cup B$, $S(c) \equiv c$, where A is $solvables(a)$ and B is $solvables(b)$, let $S' = merge(a, b, S)$. Then the following properties hold of S' if they hold of S:

1. Functionality.
2. Subterm closure: S is *subterm-closed* if for any $d \in dom(S)$,

$$[\![d]\!] \subseteq dom(S).$$

3. Range closure: S is *range-closed* if for any $d \in \mathit{dom}(S)$,

$$U(\mathit{solvables}(S(d))) \subseteq \mathit{dom}(S),$$

and for any $c \in \mathit{solvables}(S(a))$,

$$S(c) \equiv c.$$

4. Norm closure: S is *norm-closed* if for any $d \in \mathit{dom}(S)$,

$$S(d) \equiv \mathit{norm}(S)(d).$$

This of course holds trivially for uninterpreted terms d.

5. Idempotence: S is *idempotent* if

$$\begin{array}{rcl} S[S(d)] & \equiv & S(d), \\ \mathit{norm}(S)(S(d)) & \equiv & S(d), \text{ and} \\ \mathit{norm}(S)(\mathit{norm}(S)(d)) & \equiv & \mathit{norm}(S)(d)\ . \end{array}$$

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 invariants 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 4.1 (congruence closure) *If $S = cc(S') \neq \bot$, then S is congruence-closed. Similarly, if $S = \mathit{process}(T) \neq \bot$, then S is congruence-closed.*

Proof. If $cc(S')$ terminates returning a solution set S, then by the definition of *cc*, we know that S does not contain any unmerged pairs of congruent left-hand sides and is therefore congruence-closed. If $\mathit{process}(T) = S$ for some solution set S, then $S = cc(S')$ for some S', and hence S must be congruence-closed. ∎

Lemma 4.2 (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 $S' = \mathit{merge}(a, b, S) \neq \bot$, 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 2.13,

$$\mathit{norm}(R)(a) \equiv \mathit{norm}(R)(b).$$

Since $S(c) \equiv c$ for $c \in A \cup B$,

$$\begin{array}{rcl} norm(S)(a) & \equiv & a, \text{ and} \\ norm(S)(b) & \equiv & b. \end{array}$$

Hence, by *norm decomposition*, we have

$$norm(S')(a) \equiv norm(S')(b).$$

By Definition 2.8, $dom(R) \subseteq A \cup B$, hence $U(dom(S')) = U(dom(S))$. ■

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

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

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 $a, b \in dom(S)$ 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*,

$$\begin{array}{rcl} norm(S')(S(a)) & \equiv & norm(S')(S(b)) \text{ and} \\ U(dom(S')) & = & U(dom(S)). \end{array}$$

By *monotonicity*, for any c and d in $dom(S)$ 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 4.3 (substitution soundness)

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

Lemma 4.4 (norm soundness)

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

Lemma 4.5 (can soundness)

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

Lemma 4.6 (merge soundness)
Let $S' = merge(a, b, S) \neq \perp$, then:

1. *If $T, S \vdash a = b$, and $T, S' \vdash c = d$ with $vars(c = d) \subseteq vars(T \cup S)$, then*
$$T, S \vdash c = d.$$
2. *Otherwise, $merge(a, b, S) = \perp$, and*
$$T, S \vdash \perp.$$

Proof. If $S' = merge(a, b, S) \neq \perp$, 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 $merge(a, b, S) = \perp$, then by similar reasoning using the *solve*-$\perp$ rule, $T, S \vdash$ *false* is derivable. ■

Lemma 4.7 (cc soundness)

1. *If $S' = cc(S) \neq \perp$, $T, S' \vdash a = b$ for $vars(a = b) \subseteq vars(T, S)$, then*
$$T, S \vdash a = b.$$
2. *Otherwise, $cc(S) = \perp$, and*
$$S \vdash false.$$

Proof. By computation-induction on the definition of *cc*, and applying *merge soundness*. ■

Lemma 4.8 (process soundness)

1. *If $S = process(T_1) \neq \perp$, $T_1 \subseteq T_2$, and $T_2, S \vdash c = d$ for $vars(c = d) \subseteq vars(T_2)$, then*
$$T_2 \vdash c = d.$$
2. *Otherwise, $process(T_1) = \perp$, and*
$$T_1 \vdash false.$$

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 $S' = \mathit{process}(T_1')$, we have the induction hypothesis that for any c, d such that $\mathit{vars}(c = d) \subseteq \mathit{vars}(T_2)$

$$T_2, S' \vdash c = d \text{ implies } T_2 \vdash c = d.$$

We know by *can soundness* that

$$\begin{array}{rcl} T_2, S' & \vdash & \overline{a} = a, \text{ and} \\ T_2, S' & \vdash & \overline{b} = b \end{array}$$

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 $\mathit{process}(T_1) = \bot$, then $T_2 \vdash \mathit{false}$. ■

Theorem 4.9 (soundness)

1. *If $S = \mathit{process}(T) \neq \bot$, $\mathit{vars}(a = b) \subseteq \mathit{vars}(T)$, and $\overline{a} \equiv \overline{b}$, then*

$$T \vdash a = b.$$

2. *Otherwise, $\mathit{process}(T) = \bot$, and*

$$T \vdash \mathit{false}.$$

Proof. If $S = \mathit{process}(T) \neq \bot$, then by *can soundness*,

$$\begin{array}{rcl} T, S & \vdash & a = \overline{a}, \text{ and} \\ T, S & \vdash & b = \overline{b} \end{array}$$

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 $\mathit{process}(T) = \bot$, then already by *process soundness*, $T \vdash \mathit{false}$. ■

Completeness. We show that when $S = \mathit{process}(T)$ then $\mathit{can}(S)$ 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 4.10 (confluence)
If S is congruence-closed and $U(\llbracket a \rrbracket) \subseteq \mathit{dom}(S)$, then $\mathit{can}(S)(a) \equiv \mathit{norm}(S)(a)$.

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

If a is uninterpreted and of the form $f(a_1, \ldots, a_n)$, then

$$\mathit{can}(S)(a) \equiv \mathit{lookup}(S)(f(\overline{a_1}, \ldots, \overline{a_n})).$$

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

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

$$\begin{array}{ll} & \mathit{can}(S')(S(a)) \\ \equiv & \mathit{norm}(S')(S(a)) \\ \equiv & \mathit{norm}(R)(\mathit{norm}(S)(S(a))) \\ \equiv & \mathit{norm}(R)(\mathit{norm}(S)(a)) \\ \equiv & \mathit{norm}(S')(a) \\ \equiv & \mathit{can}(S')(a). \end{array}$$

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 $f(b_1, \ldots, b_n) \in \mathit{dom}(S)$, 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*,

$$\mathit{can}(S')(\overline{a}) \equiv \mathit{lookup}(S')(f(\mathit{can}(S')(a_1), \ldots, \mathit{can}(S')(a_n))) \equiv \mathit{can}(S')(a).$$

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

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

■

Lemma *can composition* with $\emptyset$ for R yields the idempotence of $\mathit{can}(S)$ for congruence-closed S so that we can define a σ-model in terms of $\mathit{can}(S)$.

Definition 4.12 (Model M_S) *M_S is defined as a model with domain D_S defined as $\{a | \mathit{can}(S)(a) = a\}$ such that*

$$\begin{array}{rcl} M_S(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) & = & \mathit{lookup}(S)(f(\mathbf{a_1}, \ldots, \mathbf{a_n})),\ \textit{if } f \textit{ is uninterpreted} \\ M_S(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) & = & \sigma(f(\mathbf{a_1}, \ldots, \mathbf{a_n})),\ \textit{if } f \textit{ is interpreted.} \end{array}$$

Lemma 4.13 (M_S σ-model) *M_S is a σ-model.*

Proof. We need to show that $M_S, \rho \vdash \sigma(a) = a$ for any assignment ρ from $\mathit{vars}(a)$ to values in D_S. Let us define $\rho[a]$ as

$$\begin{array}{rcl} \rho[x] & = & \rho(x) \\ \rho[f(a_1, \ldots, a_n)] & = & f(\rho[a_1], \ldots, \rho[a_n]) \end{array}$$

Note that $M_S[\![a]\!]\rho = \mathit{can}(S)(\rho[a])$ since

- If $a \equiv x$
$$l.h.s. = M_S[\![x]\!]\rho = \rho(x) = \rho[x] = r.h.s..$$
- If $a \equiv f(a_1, \ldots, a_n)$, then
$$\begin{array}{rcl} l.h.s. & = & \mathit{can}(S)(f(M_S[\![a_1]\!]\rho, \ldots, M_S[\![a_n]\!]\rho)) \\ \{I.H\} & = & \mathit{can}(S)(f(\mathit{can}(S)(\rho[a_1]), \ldots, \mathit{can}(S)(\rho[a_n]))) \\ \{\mathit{idempotence\ of\ can}\} & = & \mathit{can}(S)(f(\rho[a_1], \ldots, \rho[a_n])) \\ & = & r.h.s. \end{array}$$

Since $\models_\tau \sigma(a) = a$, it must also be the case that $\models_\tau \rho[\sigma(a)] = \rho[a]$ by Lemma 2.4. Then, by Definition 2.1, $\sigma(\rho[\sigma(a)]) \equiv \sigma(\rho[a])$. Then we can show that $M_S[\![\sigma(a)]\!]\rho = M_S[\![a]\!]\rho$ as follows

$$\begin{array}{rcl} l.h.s. & = & \mathit{can}(S)(\rho[\sigma(a)]) \\ \{\sigma{-}\mathit{norm}\} & = & \mathit{can}(S)(\sigma(\rho[\sigma(a)])) \\ & = & \mathit{can}(S)(\sigma(\rho[a])) \\ \{\sigma{-}\mathit{norm}\} & = & \mathit{can}(S)(\rho[a]) \\ & = & r.h.s. \end{array}$$

■

Lemma 4.14 (expand congruence closed) *If S is congruence-closed and $S^+ = expand(S, \overline{a}, \overline{b})$, then S^+ is also congruence-closed.*

Proof. Note that $S^+(c) \equiv S(c)$ for all c and $can(S)(c) \equiv c$ for all $c \in dom(S^+) - dom(S)$. If there are uninterpreted terms d and e such that $d \overset{S^+}{\sim} e$, then by the construction of S^+, both d and e must be in $dom(S^+) - dom(S)$. If $d \in dom(S)$ and $e \in dom(S^+) - dom(S)$, then by congruence-closure of S, $can(S)(e) \equiv S(d) \not\equiv e$ which contradicts the idempotence of can. On the other hand, if d and e are both in $dom(S^+) - dom(S)$, then $d \equiv f(d_1, \ldots, d_n)$ and $e \equiv f(e_1 \ldots, e_n)$, and for some i, $d_i \not\equiv e_i$ but $S(d_i) \equiv S(e_i)$. Without loss of generality, suppose $S(d_i) \not\equiv d_i$, then $d_i \in dom(S)$. Then, by confluence, $can(S)(d_i) \equiv norm(S)(d_i) \equiv S(d_i) \not\equiv d_i$, but this violates the idempotence of can. ∎

Lemma 4.15 (can expand) *If S is congruence-closed and $S^+ = expand(S, \overline{a}, \overline{b})$, then $can(S^+)(c) = can(S)(c)$.*

Proof. The proof is by induction on c. If $c \equiv x$ for some variable x, either $x \in dom(S)$ and $can(S^+)(x) \equiv can(S)(x)$ or $x \notin dom(S)$ and $can(S^+)(x) \equiv x \equiv can(S)(x)$.

When c is an uninterpreted term of the form $f(c_1, \ldots, c_n)$, we know by the induction hypothesis that $can(S^+)(c_i) \equiv can(S)(c_i)$ for $1 \leq i \leq n$. If there is a term $d \in dom(S^+)$ such that $d \equiv f(d_1, \ldots, d_n)$ and $S(d_i) \equiv can(S^+)(c_i)$ for $1 \leq i \leq n$, then by Lemma 4.14, either $d \in dom(S)$ and $can(S^+)(c) \equiv can(S)(c)$, or $d \in dom(S^+) - dom(S)$ and $can(S^+)(c) \equiv S^+(d) \equiv d \equiv lookup(S)(f(\overline{c_1}, \ldots, \overline{c_n})) \equiv can(S)(c)$.

If c is interpreted and of the form $f(c_1, \ldots, c_n)$, then by the induction hypothesis and the definition of can, $can(S^+)(c) \equiv \sigma(f(\overline{c_1}, \ldots, \overline{c_n})) \equiv can(S)(c)$. ∎

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 4.16 (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,

$$\mathit{norm}(S)(a') \equiv \mathit{norm}(S)(b').$$

Since S is congruence-closed, by *confluence*,

$$\begin{array}{rcl} \mathit{can}(S)(a') & \equiv & \mathit{norm}(S)(a') \text{ and} \\ \mathit{can}(S)(b') & \equiv & \mathit{norm}(S)(b'). \end{array}$$

Hence, $\mathit{can}(S)(a') \equiv \mathit{can}(S)(b')$.

By *can composition* and *can expand*,

$$\begin{array}{rcccl} \mathit{can}(S)(a') & = & \mathit{can}(S)(\mathit{can}(S^{+})(a)) & = & \mathit{can}(S)(a) \\ \mathit{can}(S)(b') & = & \mathit{can}(S)(\mathit{can}(S^{+})(b)) & = & \mathit{can}(S)(b) \end{array}$$

Hence, $\mathit{can}(S)(a) = \mathit{can}(S)(b)$ and

$$M_S, \rho_S^X \models a = b.$$

A similar argument shows that for $c = d \in T'$, $\mathit{can}(S)(c) = \mathit{can}(S)(d)$. Since, by the induction hypothesis, $\mathit{can}(S')(c) \equiv \mathit{can}(S')(d)$, we also have

$$\mathit{can}(S)(c) \equiv \mathit{can}(S)(d).$$

■

Theorem 4.17 (completeness) *If $S = \mathit{process}(T) \neq \perp$ and $T \vdash a = b$, then $\mathit{can}(S)(a) \equiv \mathit{can}(S)(b)$.*

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

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 $\mathit{dom}(S)$ 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*.

Chapter 5

Conclusions

Shostak's original decision procedure [Sho84] 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. Subsequent variants of Shostak's algorithm have been similarly flawed. We have presented the first correct algorithm that captures Shostak's key insights, and described proofs of termination, soundness, and completeness.

Our algorithm has been formally verified using PVS [FS02]. The verification took about four man-months. Some of the detailed proofs given here were developed as input for the formal verification. The PVS verification identified some minor gaps in the earlier version of the proof. In particular, the conditions of σ-distributivity and σ-substitutivity (Lemmas 2.3 and 2.5) were identified during the formal verification and the proofs of these lemmas were developed interactively with PVS. The short version of this paper [RS01] was missing the condition in Lemma 2.17 that a and b must be restricted to elements of $dom(R)$. The need for this condition was noticed when preparing the PVS proof. The lemma was in fact applied only when the condition already held. Also, Lemma 4.15 was stated and used in the proof of Lemma 4.16 with the implicit claim that the justification was trivial. However, the proofs of Lemmas 4.14 and 4.15 are not straightforward.

The algorithm described here is the basis of the ICS decision procedures [FORS01]. The theory developed in this paper has been instrumental in building a reliable, efficient, and compact implementation.

Decision procedures are critical deductive components that merit careful theoretical scrutiny. We have presented a rigorous development of a correct version of Shostak's widely used combination decision procedure. The theory outlined in this paper is the basis for both a formal verification of the algorithm as well as a practical implementation.

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.

Bibliography

[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. 1, 3

[Bjø99] Nikolaj Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Stanford University, 1999. 1, 3

[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. 1

[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. 1, 3, 7

[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. 1, 7

[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. 1

[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. 4, 27

[FS02] Jonathan Ford and Natarajan Shankar. Formal verification of a combination decision procedure. Submitted to CADE'02. Available at `ftp://ftp.csl.sri.com/pub/users/shankar/ford-shostak.pdf`., 2002. 4, 13, 27

[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. 1

[Koz77] Dexter Kozen. Complexity of finitely presented algebras. In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pages 164–177, Boulder, Colorado, 2–4 May 1977. 7

[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. 1

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

[NO80] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980. 1, 7

[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. 1, 4

[RS01] Harald Rueß and Natarajan Shankar. Deconstructing Shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, Boston, MA, 2001. IEEE Computer Society. 3, 4, 27

[Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, July 1978. 1, 7

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984. 1, 7, 27

[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. 1