Deciding Propositional Combinations of Equalities and Inequalities

Jean-Christophe Filliâtre[1], Sam Owre[2], Harald Rueß[2], and N. Shankar[2]

[1] LRI, URA 410 CNRS Bat 490,
Université Paris Sud 91405 Orsay Cedex, France
filliatr@lri.fr

[2] Computer Science Laboratory SRI International,
333 Ravenswood Ave. Menlo Park, CA 94025, USA
{owre,ruess,shankar}@csl.sri.com

Abstract

We address the problem of combining individual decision procedures into a single decision procedure. Our combination approach is based on using the canonizer obtained from Shostak's combination algorithm for equality. We illustrate our approach with a combination algorithm for equality, disequality, arithmetic inequality, and propositional logic. Unlike the Nelson–Oppen combination where the processing of equalities is distributed across different closed decision procedures, our combination involves the centralized processing of equalities in a single procedure. The termination argument for the combination is based on that for Shostak's algorithm. We also give soundness and completeness arguments.

1 Introduction

Deductive support for the analysis of a large range of systems requires cooperation among different verification techniques. Theories such as those for equality, propositional logic, arithmetic, arrays, and bit-vectors occur naturally in many system specifications, both for hardware and software. In addition, the use of uninterpreted function symbols has proven to be essential for significantly scaling up verification methods [Cyr93, BD94]. Decision procedures for the combination of theories are usually based on Nelson and Oppen's [NO79, TH96] or on Shostak's [Sho84] algorithm.

There are effective point solutions for each of the theories above. Binary decision diagrams, for example, are often used for deciding propositional formulas describing hardware circuits. Congruence closure is an efficient method for deciding the satisfiability of conjunctions of equalities over terms with uninterpreted function symbols. There are many algorithms for finding solutions of systems of inequalities, and there are specialized procedures for dealing with equational theories such as bit-vectors [MR98]. Most problems contain symbols from these theories. The challenge is to effectively decide the combination of these theories.

We use Shostak's [Sho84] combined decision procedure for equality and the canonizer for terms given by this procedure as the core of our combination framework. Shostak's method yields a single combined online decision procedure for a class of solvable and canonizable theories, whereas Nelson and Oppen [NO79] methods can be used to combine decision procedures for disjoint equality theories. Shostak's method additionally yields a canonizer, a function *canon* such that $canon(E)(a)$ and $canon(E)(b)$ are syntactically equal whenever the equality $a = b$ is derivable from the equalities in E. Since Shostak's algorithm and subsequent variations of it [CLS96, BDL96, Bjø99] turned out to be incomplete and non-terminating, Rueß and Shankar [RS01] developed an algorithm which is indeed complete for the combination of Shostak theories. This algorithm is able to effectively decide proof obligations such as

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

However, verification conditions such as

$$f(f(x) - f(y)) \neq f(z),\ y \leq x,\ y \geq x + z,\ z \geq 0 \vdash \mathit{false}$$

are outside the scope of the algorithm [RS01], since inequalities are not in the domain of this procedure.

We describe a natural extension of Shostak-like decision procedures for also dealing with inequalities by combining specialized procedures for deciding equalities, disequalities, inequalities, and propositional formula. The canonizer given by Shostak's procedure for equality can be used by other decision procedures such as those for disequality, inequality, and propositional logic. Conversely, these decision procedures can generate equality information that can be employed by the equality procedure. We address the specific case of combining individual decision procedures for disequality, inequality, and propositional logic with Shostak's algorithm for equality.

Our combination method is different from that of Nelson and Oppen [NO79, BDS00] for disjoint equality theories. As with Nelson and Oppen, we require the individual decision procedures to only generate equalities but their inputs can be inequalities or propositional formulas. Also, we do not require disjointness between the signatures of the individual decision procedures. Since our combination method is based on sharing certain operations like canonization across individual theories, we obtain a simplification procedure for the combined theory.

A fragment of the theory considered here, namely propositional logic with uninterpreted functions over arbitrary domains and equality over these domains, can also be decided using Ackermann's trick [Ack54] for eliminating function symbols at the expense of introducing new variables and congruence constraints. This equivalence-preserving transformation may result in exponential blow-up, but recent refinements [PRSS99, BGV99, GSZA98, GvdP00] have made it possible to use Ackermann's trick for some interesting case studies. Congruence-closure based algorithms, however, have the potential to scale better, since the core algorithm has an $n * \mathit{log}(n)$ worst-case complexity, but there are no systematic comparisons on a representative range of examples. Also, procedures based on Ackermann's encoding do not integrate well with other verification methods such as rewriting.

This paper is organized as follows. Section 2 presents the logical background. Section 3 describes the combination of decision procedures for equality, disequality, propositional reasoning, and inequality reasoning. Section 4 states the results of termination, soundness and completeness of this combination together with proof outlines. In Section 5 we give an overview of an implementation of our combination procedure. Section 6 contains some concluding remarks.

2 Background

We assume that terms, atoms, and propositions are built from a given signature. Atomic propositions are either equalities, disequalities, inequalities or one of the constants true and false. Propositions are either atomic propositions or conditionals. Terms, atoms, and propositions obey the following grammar:

$$\begin{array}{lcll}
\text{terms} & \mathcal{T} & ::= & x \mid f(\mathcal{T}, \ldots, \mathcal{T}) \\
\text{equalities} & \mathcal{E} & ::= & \mathcal{T} = \mathcal{T} \\
\text{disequalities} & \mathcal{D} & ::= & \mathcal{T} \neq \mathcal{T} \\
\text{inequalities} & \mathcal{I} & ::= & \mathcal{T} < \mathcal{T} \mid \mathcal{T} \leq \mathcal{T} \\
\text{atoms} & \mathcal{A} & ::= & \mathcal{E} \mid \mathcal{D} \mid \mathcal{I} \mid \mathsf{true} \mid \mathsf{false} \\
\text{propositions} & \mathcal{P} & ::= & \mathcal{A} \mid \mathsf{if}\ \mathcal{A}\ \mathsf{then}\ \mathcal{P}\ \mathsf{else}\ \mathcal{P}
\end{array}$$

where x is a variable name and f a function symbol. Constants are 0-ary function symbols, and the usual infix notations are used for binary function symbols. The symbol $\equiv$ denotes syntactic equality. We use the convention that $\lhd$ stands for either $<$ or $\leq$. The notion of a model for a set of propositions T is defined in the usual way, and $T \models p$ expresses the fact that every model of T is also a model of p. Two sets of propositions T and T' are said to be equivalent, written $T \approx T'$, if for any proposition p, $T \models p$ iff $T' \models p$.

The logical connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$,$\Leftrightarrow$) are defined from conditionals, and conditionals in propositions are implicitly assumed to be simplified by the following:

$$\begin{array}{ll}
 & \mathsf{if}\ (\mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q)\ \mathsf{then}\ p'\ \mathsf{else}\ q' \\
\rightsquigarrow & \mathsf{if}\ a\ \mathsf{then}\ (\mathsf{if}\ p\ \mathsf{then}\ p'\ \mathsf{else}\ q')\ \mathsf{else}\ (\mathsf{if}\ q\ \mathsf{then}\ p'\ \mathsf{else}\ q')
\end{array}$$

The negation of an atom a, written $neg(a)$, is defined as follows:

$$\begin{array}{lcllcl} neg(t_1 = t_2) & \stackrel{\text{def}}{=} & t_1 \neq t_2 & neg(t_1 < t_2) & \stackrel{\text{def}}{=} & t_2 \leq t_1 \\ neg(t_1 \neq t_2) & \stackrel{\text{def}}{=} & t_1 = t_2 & neg(t_1 \leq t_2) & \stackrel{\text{def}}{=} & t_2 < t_1 \\ neg(\mathsf{true}) & \stackrel{\text{def}}{=} & \mathsf{false} & neg(\mathsf{false}) & \stackrel{\text{def}}{=} & \mathsf{true} \end{array}$$

Some of the function symbols are *interpreted*, i.e., they have a specific interpretation, while the remaining function symbols are uninterpreted, and can be assigned arbitrary interpretations. If t is a term (or a proposition) $U(t)$ denotes the set of its uninterpreted subterms.

3 The Combination Procedure

We describe a procedure for deciding the combination of certain equality theories with disequality, arithmetic inequalities, and propositional reasoning. Our combination procedure relies on two oracles: one for equality reasoning, and the other for reasoning about linear arithmetic inequalities. We first describe the interfaces of these two oracles together with semantic requirements. Then we include the combination procedure itself.

The decision procedure for equality is the one described in [RS01]. Given a canonizer and solver for equalities over interpreted terms, the procedure in [RS01] yields a decision procedure for the equality theory over terms with both interpreted and uninterpreted function symbols.[1]

The central notion is that of an *estate*, which is a representation of a congruence-closed partition of terms given by the currently processed equations E. Such a state maintains the solved form for equations in the form of a solution for *solvables*, that is, variables or terms with a top-level uninterpreted function symbol. A solved form for solvable x, for example, may contain x in the scope of uninterpreted terms. Furthermore, solved forms may contain new variables, which are not subterms of any term in E. These new variables are introduced by theory-specific solvers such as the solver for the theory of tuples and projections. An *estate* can be thought of as a function with solvables in the domain and solved forms in the codomain. We also require the domain to include all uninterpreted subterms of $U(E)$. The extension $extend(S)(U)$ of an estate E denotes an estate which extends E with $x \mapsto x$ for all solvable subterms x of U not already in the domain of E.

Given an estate E and a term t, the operation $ecanon(E)(t)$ returns a canonical representative of t with respect to this partitioning. Given an estate E and a new equality $t_1 = t_2$, the operation $eprocess(E)(t_1 = t_2)$ returns either $\perp$ if an inconsistency is found, or a new estate (the congruence-closure state in that the partitioning is refined). For the purpose of this paper it is sufficient to state the following properties of *ecanon* and *eprocess*.

(E_1) $ecanon(E)(t_1) \equiv ecanon(E)(t_2) \iff E \models t_1 = t_2$

(E_2) $eprocess(E)(t_1 = t_2) = \perp \iff E,\ t_1 = t_2 \models \mathsf{false}$

(E_3) $eprocess(E)(t_1 = t_2) \approx \{t_1 = t_2\} \cup E$

(E_4) $eprocess(E)$ can bring in new variables, but only when the number of equivalence classes of uninterpreted terms in $U(E)$ decreases.

For definitions of the *ecanon* and *eprocess* operators together with proofs of the requirements above see [RS01]. Two terms t_1 and t_2 are said to be in the same equivalence class if $ecanon(E)(t_1) \equiv ecanon(E)(t_2)$. Furthermore, a solvable term is said to be *eliminated* if $canon(E)(a) \not\equiv a$; otherwise it is *uneliminated*.

The inequality reasoner consists of a single operation *iprocess*. Given a set of inequalities I, the operation $iprocess(I)$ returns either $\perp$ if an inconsistency is found, or a complete set of derived equalities. This operation is required to be computable and to satisfy the properties (let I be a set of inequalities):

(I_1) $iprocess(I) = \perp \iff I \models \mathsf{false}$

[1] The exposition in [RS01] is limited to one interpreted theory only, but canonizers and solvers for combinations of theories can be obtained in certain cases from those of the constituent theories. Such modularity results have been obtained, for example, for canonizers based on term rewriting systems [Toy87, Rus87] and equational unifiers [BS96].

(I_2) $iprocess(I) = E \implies \forall e \in E.\ I \models e$

(I_3) $iprocess(I) = E \implies \forall e.\ I \models e \Rightarrow E \models e$

(I_4) $\forall e \in iprocess(I).\ U(e) \neq \emptyset$ and $U(e) \subseteq U(I)$

It is easily verified that, for example, a Fourier-Motzkin elimination procedure satisfies these requirements. Notice that requirement (I_2) implies that $I \approx I \cup E$ for $E = iprocess(I)$.

There are two main operations in the combination algorithm. The *process* operation adds a proposition to a combined state, and *inconsistent* checks for inconsistencies in a complete way. This separation simplifies the presentation and it avoids unnecessary computations by avoiding eager case splits by the *process* operation.

Definition 1 (decision procedure state) *A decision procedure state is a quadruple (E, D, I, P) where E is a set of equalities (an estate), D a set of disequalities, I a set of inequalities and P a set of non-atomic propositions, such that*

- *the atomic part is consistent, i.e., $E, D, I \not\models \mathsf{false}$*
- *any subterm t occurring in $D \cup I \cup P$ is canonized with respect to E, i.e. $ecanon(E)(t) \equiv t$*

The empty state is $empty \stackrel{def}{=} (\emptyset, \emptyset, \emptyset, \emptyset)$.

The E component of a state is an *estate*. Therefore, any given term may be canonized by $ecanon(E)$. This canonization is extended to propositions in order to simplify them.

$$
\begin{array}{rcl}
canon(E)(a) & = & canonatom(E)(a) \\
canon(E)(\mathsf{if}\ a\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2) & = & canon(E)(p_1) \quad when\ a' \equiv \mathsf{true} \\
& = & canon(E)(p_2) \quad when\ a' \equiv \mathsf{false} \\
& = & \mathsf{if}\ a'\ \mathsf{then}\ canon(E)(p_1)\ \mathsf{else}\ canon(E)(p_2) \\
& & \quad where\ a' = canonatom(E)(a) \\
\\
canonatom(E)(t_1 = t_2) & = & \mathsf{true} \quad when\ t_1' \equiv t_2' \\
& = & t_1' = t_2' \quad otherwise \\
& & \quad where\ t_1' = ecanon(E)(t_1) \\
& & \quad and\ t_2' = ecanon(E)(t_2) \\
canonatom(E)(t_1 \neq t_2) & = & \mathsf{false} \quad when\ t_1' \equiv t_2' \\
& = & t_1' \neq t_2' \quad otherwise \\
& & \quad where\ t_1' = ecanon(E)(t_1) \\
& & \quad and\ t_2' = ecanon(E)(t_2) \\
canonatom(E)(t_1 \lhd t_2) & = & ecanon(E)(t_1) \lhd ecanon(E)(t_2) \\
canonatom(E)(\mathsf{true}) & = & \mathsf{true} \\
canonatom(E)(\mathsf{false}) & = & \mathsf{false}
\end{array}
$$

Figure 1: The *canon* operation.

Definition 2 (canon) *Given an estate E and a proposition p, $canon(E)(p)$ is a proposition defined following equations in Figure 1.*

For example, if x is canonized to 0 in E, then $canon(E)(\mathsf{if}\ x = 0\ \mathsf{then}\ p\ \mathsf{else}\ q)$ simplifies to $canon(E)(p)$. If S is a state, we write $canon(S)$ for $canon(E)$ where E is the equality part of S. The following lemma expresses that the result of the *canon* operation is equivalent with its argument.

Lemma 1 *For an estate E and a proposition p, we have*

$$\{p\} \cup E \approx \{canon(E)(p)\} \cup E$$

The operation *process* is used to build modified states. Given a state and a proposition, it returns either a new state or an inconsistency.

Definition 3 (process) *Given a state S and a proposition p, the operation $process(S)(p)$ returns either $\perp$ or a new state, following equations in Figure 2.*

The extension of the domain in the definition of *process* is needed to maintain the invariant that every uninterpreted term in D, I, or P is in the domain of the estate E.

The *process* operation does not perform any case split and is therefore incomplete. The operation *inconsistent* is used to check for the inconsistency of a state in a complete way.

$$
\begin{array}{rcl}
process(E,D,I,P)(p) & = & processone(E_0,D,I,P)(p_0) \\
 & & \quad where\ p_0 = canon(E)(p) \\
 & & \quad and\ E_0 = extend(E)(U(p_0)) \\
 & & \\
processone(E,D,I,P)(t_1 = t_2) & = & \perp \quad when\ E_0 = \perp \\
 & = & processmany(E_0,\emptyset,\emptyset,\emptyset)(D \cup I \cup P) \quad otherwise \\
 & & \quad where\ E_0 = eprocess(E)(t_1 = t_2) \\
processone(E,D,I,P)(t_1 \neq t_2) & = & \perp \quad when\ t_1 \equiv t_2 \\
 & = & (E, \{t_1 \neq t_2\} \cup D, I, P) \quad otherwise \\
processone(E,D,I,P)(t_1 \lhd t_2) & = & \perp \quad when\ E_0 = \perp \\
 & = & processmany(E,D,\{t_1 \lhd t_2\} \cup I, P)(E_0) \quad otherwise \\
 & & \quad where\ E_0 = iprocess(\{t_1 \lhd t_2\} \cup I) \\
processone(S)(\mathsf{true}) & = & S \\
processone(S)(\mathsf{false}) & = & \perp \\
processone(E,D,I,P)(p) & = & (E,D,I,\{p\} \cup P) \quad otherwise \\
 & & \\
processmany(\perp)(T) & = & \perp \\
processmany(S)(\emptyset) & = & S \\
processmany(S)(\{p\} \uplus T) & = & processmany(process(S)(p))(T)
\end{array}
$$

Figure 2: The *process* operation.

Definition 4 (inconsistent) *Given a state S, the operation $inconsistent(S)$ returns* true *or* false *following the equations in Figure 3.*

$$
\begin{array}{rcl}
inconsistent(E,D,I,\emptyset) & = & \mathsf{false} \\
inconsistent(E,D,I,\{\mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q\} \uplus P) & = & split(S', S'')
\end{array}
$$

$$
\begin{array}{rrcl}
where & S' & = & processmany(E,D,I,P)(\{a,p\}) \\
and & S'' & = & processmany(E,D,I,P)(\{neg(a),q\})
\end{array}
$$

$$
\begin{array}{rcl}
split(\perp,\perp) & = & \mathsf{true} \\
split(\perp,S) & = & inconsistent(S) \\
split(S,\perp) & = & inconsistent(S) \\
split(S,S') & = & inconsistent(S) \ \wedge \ inconsistent(S')
\end{array}
$$

Figure 3: The *inconsistent* operation.

Processing the example from the introduction may help in understanding how the combination procedure works.

Example. Suppose we successively process the propositions $p_1 \equiv f(f(x) - f(y)) \neq f(z)$, $p_2 \equiv y \leq x$, $p_3 \equiv y \geq x + z$ and $p_4 \equiv z \geq 0$. Successively processing p_1, p_2, p_3 propositions yields a state $S = (\emptyset, \{p_1\}, \{p_2; p_3\}, \emptyset)$. Then the processing of p_4 triggers a call to *iprocess*$(\{p_2; p_3; p_4\})$, which returns the set $E = \{z = 0; x = y\}$. Then *processmany*$(S)(E)$ is called recursively. The processing of $z = 0$ triggers the re-processing of the disequality to become $f(f(x) - f(y)) \neq f(0)$. Then, the inequalities p_2, p_3 and p_4 are re-processed. p_2 is already in canonical form, and p_3 canonizes to $y \geq x$. Thus *iprocess* is called on $\{y \leq x; y \geq x\}$ and returns $\{x = y\}$. This equality is processed and the disequality is canonized to $f(0) \neq f(0)$, and consequently $\perp$ is returned.

4 Termination, soundness and completeness

This section establishes termination, soundness and completeness of our combination procedure.

4.1 Termination

The hard part is showing the termination of the operation *process*: indeed, the introduction of a new equality triggers the re-processing of all the elements of the state which are not equalities. This processing involves first canonization and thus may lead to other equalities, as in the processing of $\mathsf{if}\ x = 1\ \mathsf{then}\ y = 2\ \mathsf{else}\ z = 3$ when one knows that $x = 1$.

Lemma 2 (Termination) *The operations process and inconsistent always terminate.*

PROOF. Assuming the termination of *process* and *processmany*, it is easy to establish the termination of *inconsistent*(E, D, I, P). The termination measure is given by the total number of if-then-else operations in P. Note that the total number of if-then-else operations in *processmany*$(E, D, I, P)(T)$ is the sum of those in P and T.

The termination of *process* and *processmany* is not as obvious. Each invocation of *canon*$(E)(t)$ in the procedure introduces no new uninterpreted terms, i.e., $U(\mathit{canon}(E)(t)) \subseteq U(E)$. E maintains the solved form for equations processed by *eprocess* in the form of a solution for a solvable term, i.e., a variable or uninterpreted term. Also, recall that a solvable term is said to be eliminated if $\mathit{canon}(E)(a) \not\equiv a$. Each equation generated by *iprocess*$(\{t_1 \lhd t_2\} \cup I)$ or *canon*$(E)(p)$ must be in canonical form. Since such an equation must not be vacuously true or false, it must contain at least one uneliminated solvable term. When this equation is processed by *eprocess*, at least one uneliminated solvable term in the equation is eliminated. The measure used to establish termination of *processmany*$(E, D, I, P)(T)$ is a lexicographic combination of

- the number of equivalence classes of uninterpreted terms in E,
- the number of uneliminated solvable terms in E, and
- the length of T.

In the recursive definition of *processmany*$(S)(T)$, if the number of equivalence classes of uninterpreted terms in S does not decrease, then no new variables are introduced by *eprocess*, and therefore each invocation of *eprocess* decreases the number of uneliminated solvable terms in E. When *eprocess* is not invoked in a recursive call of *processmany*, the length of T decreases. Note that for the inequality case for *processmany*, E_0 is either empty or nonempty. Termination is obvious when E_0 is empty, and if it is not, then a single definition expansion is enough to decrease the measure by processing the first equation in E_0. $\square$

4.2 Soundness

The main observation that yields soundness and completeness is that every transition from state S to state S' preserves equivalence. This is expressed by the following central lemma.

Lemma 3 (Main lemma) *Let S be a state and T a set of propositions. Then*

$$S' = \mathit{processmany}(S)(T) \neq \bot \;\Rightarrow\; S' \approx S \cup T$$

PROOF. The proof is by induction over T. The case $T = \emptyset$ is immediate. The induction case requires to show that

$$S' = \mathit{processmany}(\mathit{process}(S)(p))(T) \neq \bot \;\Rightarrow\; S' \approx S \cup \{p\} \cup T$$

Using induction hypothesis over *processmany*, it is enough to show that

$$\forall S. \quad S' = \mathit{process}(S)(p) \neq \bot \;\Rightarrow\; S' \approx \{p\} \cup S$$

This is established by induction over the code for *process*. There are six cases depending on the value of $\mathit{canon}(E)(p)$. The proofs of all cases are straightforward. □

As an immediate corollary of Lemma 3, we have

$$S = \mathit{processmany}(\mathit{empty})(T) \neq \bot \;\Rightarrow\; S \approx T$$

Theorem 1 (Soundness of process)
Let T be a set of propositions and $S = \mathit{processmany}(\mathit{empty})(T)$. Then

1. $S = \bot \;\Rightarrow\; T \models \mathsf{false}$
2. $S \neq \bot \;\Rightarrow\; \forall t_1, t_2.\ \mathit{canon}(S)(t_1) \equiv \mathit{canon}(S)(t_2) \;\Rightarrow\; T \models t_1 = t_2$

PROOF. Part 1 is proved by first generalizing to the property

$$\mathit{processmany}(S)(T) = \bot \;\Rightarrow\; S, T \models \mathsf{false}$$

and then by induction over *processmany* and *process*. The case $T = \emptyset$ is immediate. The induction case for *processmany* requires to show that

$$\mathit{processmany}(\mathit{process}(S)(p))(T) = \bot \;\Rightarrow\; S, T, p \models \mathsf{false}$$

If $\mathit{process}(S)(p) = \bot$ then $S, p \models \mathsf{false}$ by induction hypothesis and thus $S, p, T \models \mathsf{false}$ by weakening. Regarding *process*, we need to show that

$$\forall S. \quad \mathit{process}(S)(p) = \bot \;\Rightarrow\; S, p \models \mathsf{false}$$

and we proceed by induction over *process*. Let $S = (E, D, I, P)$. The non-trivial cases for $\mathit{canon}(S)(p)$ are the following:

$t_1 = t_2$: Let $E_0 = \mathit{eprocess}(E)(t_1 = t_2)$. If $E_0 = \bot$ then $E, t_1 = t_2 \models \mathsf{false}$ by (E_2) and then $S, p \models \mathsf{false}$ by Lemma 1 and Lemma 3. Otherwise, $\mathit{processmany}(E_0, \emptyset, \emptyset, \emptyset)(D \cup I \cup P) = \bot$. By induction hypothesis over *processmany* we have $E_0, D, I, P \models \mathsf{false}$. By (E_3) and Lemma 1 we have $E_0 \approx \{t_1 = t_2\} \cup E \approx \{p\} \cup E$. Thus $S, p \models \mathsf{false}$.

$t_1 \neq t_2$ when $t_1 \equiv t_2$: By (E_1) we have $E \models t_1 = t_2$ and thus $S, p \models t_1 = t_2$ by weakening. But we have also $S, p \models t_1 \neq t_2$ by Lemma 1. Thus $S, p \models \mathsf{false}$ by contradiction.

$t_1 \lhd t_2$: Let $E_0 = \mathit{iprocess}(\{t_1 \lhd t_2\} \cup I)$. If $E_0 = \bot$ then $t_1 \lhd t_2, I \models \mathsf{false}$ by (I_1) and thus $S, p \models \mathsf{false}$ by Lemma 1 and weakening. Otherwise, the result holds by induction hypothesis over *processmany*.

false : The result holds by Lemma 1.

Part 2 is almost immediate. By definition, $canon(S)(t) = ecanon(E)(t)$. Therefore, $canon(S)(t_1) \equiv canon(S)(t_2)$ implies $E \models t_1 = t_2$ by property (E_1). Then $S \models t_1 = t_2$ by weakening and finally $T \models t_1 = t_2$ by Lemma 3.

□

Theorem 2 (Soundness of inconsistent)
Let T be a set of propositions and $S = processmany(empty)(T) \neq \bot$. Then

$$inconsistent(S) = \mathsf{true} \;\Rightarrow\; T \models \mathsf{false}$$

PROOF. The proof is by induction over *inconsistent*. Let us consider the case where $S' = \bot$ and $S'' \neq \bot$. Since $S \approx T$ the goal reduces to $S \models \mathsf{false}$. By excluded-middle over a, it suffices to show $S, a \models \mathsf{false}$ and $S, neg(a) \models \mathsf{false}$. Since $\mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q \in S$ the first goal is equivalent to $S, a, p \models \mathsf{false}$. By weakening it reduces to $E, D, I, P, a, p \models \mathsf{false}$ which holds by Theorem 1 and Lemma 3 since $S' = processmany(E, D, I, P)(\{a, p\}) = \bot$. Similarly, the second goal reduces to $E, D, I, P', neg(a), q \models \mathsf{false}$, which holds by induction hypothesis over S''. The other cases are similar. □

4.3 Completeness

The completeness of *inconsistent* relies on the fact that *process* itself is already complete on atomic propositions. This is expressed by the following lemma.

Lemma 4 *Let T be a set of atomic propositions and $S = processmany(empty)(T)$. Then*

1. $T \models \mathsf{false} \;\Rightarrow\; S = \bot$
2. $S \neq \bot \;\Rightarrow\; \forall t_1, t_2.\, T \models t_1 = t_2 \;\Rightarrow\; canon(S)(t_1) \equiv canon(S)(t_2)$

PROOF. The proof is by induction over *process* and *processmany*, generalizing the property to

$$\begin{array}{l} S \not\models \mathsf{false} \;\wedge\; (\forall t, t'.\ S \models t = t' \Rightarrow canon(S)(t) \equiv canon(S)(t')) \;\Rightarrow \\ S' = processmany(S)(T) \neq \bot \;\Rightarrow \\ (S, T \models \mathsf{false} \Rightarrow S' = \bot) \;\wedge\; (\forall t, t'.\ S, T \models t = t' \Rightarrow canon(S)(t) \equiv canon(S)(t')) \end{array}$$

and

$$\begin{array}{l} S \not\models \mathsf{false} \;\wedge\; (\forall t, t'.\ S \models t = t' \Rightarrow canon(S)(t) \equiv canon(S)(t')) \;\Rightarrow \\ S' = process(S)(p) \neq \bot \;\Rightarrow \\ (S, p \models \mathsf{false} \Rightarrow S' = \bot) \;\wedge\; (\forall t, t'.\ S, p \models t = t' \Rightarrow canon(S)(t) \equiv canon(S)(t')) \end{array}$$

The case of *processmany* is easily handled. In the case of *process* we have five cases, depending on the canonization of p (the sixth one is unreachable since propositions are atomic):

$t_1 = t_2$: The result holds by induction hypothesis over *processmany*, since $E_0 \not\models \mathsf{false}$.

$t_1 \neq t_2$: For the first property, we have $S, t_1 \neq t_2 \models \mathsf{false}$ and $S \not\models \mathsf{false}$ thus $S \models t_1 = t_2$. But then $t_1 \equiv t_2$ by induction hypothesis over S and thus $\bot$ is returned.

For the second property, let us assume that $S, t_1 \neq t_2 \models t = t'$. Since $S, t_1 \neq t_2 \not\models \mathsf{false}$ we have $S \models t = t'$ (this equality holds in the initial model of E). Thus $canon(S)(t) \equiv canon(S)(t')$ by hypothesis over S.

$t_1 \lhd t_2$: We cannot apply the induction hypothesis directly, since $(E, D, \{t_1 \lhd t_2\} \cup I, \emptyset)$ may be inconsistent. But since we are processing a set of equalities (E_0), we can unfold *processmany* once, and after *eprocess* is called for the first time the induction hypothesis applies again since $E_0 \not\models \mathsf{false}$ by (E_2).

true : We have $S \cup \{p\} \approx S$. Thus for the first property, we contradict $S \not\models \mathsf{false}$ and for the second one we can use induction hypothesis.

false : The first property is immediate. For the second one, we contradict $S' \neq \bot$.

□

We are now in position to establish completeness of *inconsistent*.

Theorem 3 (Completeness of inconsistent)
Let T be a set of propositions and $S = processmany(empty)(T)$. Then

$$T \models \mathsf{false} \;\Rightarrow\; S = \bot \;\vee\; inconsistent(S) = \mathsf{true}$$

PROOF. The proof is by induction over the code of *inconsistent*. We assume that $T \models \mathsf{false}$ and we have to prove $S \neq \bot \Rightarrow inconsistent(S) = \mathsf{true}$. The case of an empty set of non-atomic propositions would contradict Lemma 4. Thus we consider a set of non-atomic propositions $\{\mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q\} \cup P$. Let $S' = processmany(E, D, I, P)(\{a, p\})$ and $S'' = processmany(E, D, I, P)(\{neg(a), q\})$. Let us consider the case where $S' = \bot$ and $S'' \neq \bot$. By hypothesis, we have $T \models \mathsf{false}$ and thus $E, D, I, P, \mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q \models \mathsf{false}$ by Lemma 3. Since $neg(a), q \models \mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q$ we have $E, D, I, P, neg(a), q \models \mathsf{false}$ and thus $S'' \models \mathsf{false}$ by Lemma 3. Then $inconsistent(S'') = \mathsf{true}$ by induction hypothesis over S'' and the result holds. The other cases are similar. □

5 Implementation

We have developed a prototype implementation of our combination algorithm in the framework of the integrated canonization and solver system ICS [FORS01]. For inequality reasoning *iprocess* we are using a Fourier-Motzkin procedure, and the algorithm underlying the implementation of *ecanon* and *eprocess* is our decision procedure for the theory of equality and disequality with both uninterpreted and interpreted function symbols [RS01]. Our equality reasoner includes canonizers and solvers for the following theories.

- Rational constants and the arithmetic operators $+$, $-$, $*$, where multiplication is restricted to multiplication by constants.
- Tuples $(t_1, \ldots, t_n)$ and projections $\pi_k(t)$. Solved forms in this theory may contain variables that do not occur in the equation to be solved. A solved form of the equation $\pi_1(x) = ()$ in the theory of tuples, for example, is given by the equation $x = ((), d)$, where d is a fresh constant.
- Array lookup $a[x]$ and update $a[x := t]$ for a functional array a. The theory of functional arrays is not solvable in the sense of Section 2. Therefore, update operators are expanded to their corresponding conditionals. These conditionals are then lifted up-front to the propositional level.

For the propositional part of the canonizer we use binary decision diagrams [Bry86] for canonizing propositions.

The combined decision procedure is capable of deciding formulas such as $x+2 = y \Rightarrow f(a[x := 3][y-2]) = f(y-x+1)$, which contains a combination of propositional connectives, uninterpreted function symbols (f), arithmetic operators, and function update.

When decision procedures are used in theorem proving systems like PVS [ORS92], verification conditions are usually proved within the context of a large number of assertions derived from the antecedents of implications, conditional tests, and predicate subtype constraints. These contexts must be changed in an incremental manner when assertions are either added or removed. Through the use of functional data structures, our implementation allows contexts to be incrementally enriched in an efficient yet side-effect-free manner. This feature supports efficient, incremental proof search.

We use many standard optimization techniques in the implementation of these algorithms such as hash-consing of terms for constant comparisons, Patricia trees for representing data structures like sets of (hash-consed) terms, and memoization for avoiding re-computations.

6 Conclusions

We have described a procedure for combining decision procedures for equalities and inequalities to obtain a decision procedure for the union of the theories of equality, disequality, inequality, and propositional logic.

Our combination algorithm is a natural extension of the equality and disequality decision procedure in [RS01], in that decision procedures for individual theories have been combined using canonization with respect to a congruence-closed partition as a shared operation. Another characteristics of our decision procedure is its laziness with respect to the processing of nonatomic propositions.

We used the Fourier-Motzkin procedure for linear inequalities as an instance of a convex decision procedure, and a BDD-based propositional simplifier as an example of a non-convex decision procedure. By examining the arguments for this specific combination, it seems to be possible to obtain a general combination method. In the convex case, the individual procedure *process* should be able to process a list of assertions A such that if E is the state of the equality procedure, then $process(E)(A)$ must yield the set of equalities S such that no equality S is already true in E and for any equality e built from the uninterpreted terms in E, $S \models e$ iff $A \models e$. In the non-convex case, the decision procedure must yield a split operation that can convert a subset of assertions into an equivalent assertion in disjunctive normal form. We are currently working on proving such a general result.

References

[Ack54] W. Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundation of Mathematics*, 1954.

[BD94] J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In David Dill, editor, *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80, Stanford, CA, June 1994. Springer-Verlag.

[BDL96] C. Barrett, D. Dill, and J. 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.

[BDS00] C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In David McAllester, editor, *17th International Conference on Computer Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000.

[BGV99] R. E. Bryant, S. German, and M. N. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In N. Halbwachs and D. Peled, editors, *Proceedings of CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 470–482. Springer-Verlag, 1999.

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

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

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

[CLS96] D. Cyrluk, P. 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.

[Cyr93] D. Cyrluk. Microprocessor verification in PVS: A methodology and simple example. Technical Report SRI-CSL-93-12, Computer Science Laboratory, Menlo Park, CA, 94025, December 1993.

[FORS01] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving (Tool presentation). In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV'2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer-Verlag, 2001.

[GSZA98] A. Goel, K. Sajid, H. Zhou, and A. Aziz. BDD based procedures for a theory of equality with uninterpreted functions. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 244–255. Springer-Verlag, 1998.

[GvdP00] J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. Technical Report SEN-R0006, Centrum voor Wiskunde en Informatica, February 2000.

[MR98] O. Möller and H. Rueß. Solving bit-vector equations. In G. Gopalakrishnan and Ph. Windley, editors, *Formal Methods in Computer-Aided Design (FMCAD '98)*, volume 1522 of *Lecture Notes in Computer Science*, pages 36–48, Palo Alto, CA, November 1998. Springer-Verlag.

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

[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 Computer Science*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469, Trento, Italy, July 1999. Springer-Verlag.

[RS01] H. Rueß and N. Shankar. Deconstructing Shostak. In *16th Symposium on Logic in Computer Science (LICS 2001)*. IEEE Press, June 2001.

[Rus87] M. Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.

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

[TH96] C. Tinelli and M. Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In Franz Baader and Klaus U. Schulz, editors, *Frontiers of Combining Systems: First International Workshop*, volume 3 of *Applied Logic Series*, pages 103–119, Munich, Germany, March 1996. Kluwer.

[Toy87] Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.