In CADE-97, vol. 1249 of LNCS,
pp. 101–115, Springer-Verlag, July 1997.

A Practical Integration of First-Order Reasoning and Decision Procedures*

Nikolaj S. Bjørner[1], Mark E. Stickel[2], Tomás E. Uribe[1]

[1] Computer Science Department, Stanford University
nikolaj|uribe@cs.stanford.edu
[2] Artificial Intelligence Center, SRI International
stickel@ai.sri.com

Abstract. We present a procedure for proving the validity of first-order formulas in the presence of decision procedures for an interpreted subset of the language. The procedure is designed to be practical: formulas can have large complex boolean structure, and include structure sharing in the form of `let`- expressions. The decision procedures are only required to decide the unsatisfiability of sets of literals. However, $\mathcal{T}$-refuting substitutions are used whenever they can be computed; we show how this can be done for a theory of partial orders and equality. The procedure has been implemented as part of STeP, a tool for the formal verification of reactive systems. Although the procedure is incomplete, it eliminates the need for user interaction in the proof of many verification conditions.

1 Introduction

We present a procedure for proving the validity of first-order formulas in the presence of decision procedures. Our procedure is motivated by formal verification, and shaped by the requirements of this problem domain. The Stanford Temporal Prover (STeP) [6] is a tool for the formal verification of reactive systems, including both hardware and software. Given a system description S, and a temporal specification φ to be proved for S, the deductive component of STeP generates *verification conditions*, first-order formulas that, if valid, establish the S-validity of φ. (See [20] for a description of this verification methodology.)

The validity of verification conditions is established relative to the usual theories of equality, inductive datatypes, integers, linear arithmetic, rationals, partial orders and finite domains. We want to build these theories into the deductive process as much as possible. For parameterized programs and real-time systems, verification conditions often include quantifiers, so ground reasoning is not enough; pure first-order reasoning is also insufficient. These formulas are mathematically trivial theorems, but are tedious to prove using interactive theorem-proving,

* This research was supported in part by the National Science Foundation under grants CCR-94-08630 and CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, and Army contract DABT63-96-C-0096 (DARPA).

and a verification effort can involve establishing dozens of them. Therefore, a procedure that can automatically establish the validity of most or all of these verification conditions is highly desirable, even if it is not complete.

Integrating specialized decision procedures into general first-order theorem proving systems is a much-discussed problem with a long line of research [25]. Much of this work has been carried out in the context of resolution, including theory resolution [28], constrained resolution [8], and the use of specialized unification [16, 2]. However, these methods usually make special demands on the decision procedures (computation of residues or complete sets of most general unifiers, identifying $\mathcal{T}$-unsatisfiable subsets, etc.). These requirements are not always satisfied by otherwise fast and efficient decision procedures. Furthermore, in a resolution setting they perform poorly on large formulas with a complex boolean structure.

Note that for some of the theories we consider, such as first-order logic with arithmetic, complete proof systems are impossible to obtain. However, our abstract procedure is complete for pure first-order logic (i.e. for the empty theory) and theories for which an appropriate version of Herbrand's theorem holds. This theoretical completeness claim holds for implementations that enumerate all possible substitutions. However, it does not hold for the much more effective selective generation of substitutions by unification and incomplete theory reasoning that we use in practice.

Our procedure is an extension of the Davis-Putnam-Loveland-Logemann propositional satisfiability checker [13, 12]. It operates on formulas in nonclausal form, and is extended to consider quantifiers. The procedure is intended to preserve the original structure of the formula, including structure sharing using `let`-expressions, as much as possible. Case splitting, instantiation, skolemization and simplification can all be performed incrementally, in a uniform setting. We take advantage of instantiations suggested by decision procedures whenever available, but can also use "black-box" procedures that only provide yes/no answers. We show how instantiations can be computed for a theory of partial orders with equality, and give some representative examples.

We should note that the STeP system includes other algorithms for the *simplification* of formulas (that is, producing simpler, equivalent formulas). In this paper we focus on a pure validity testing procedure.

Related work: Two other recent procedures have demonstrated practical utility for formal verification. The Stanford Validity Checker (SVC) [3] is a decision procedure for propositional logic extended with uninterpreted function symbols, equality, linear arithmetic, and other interpreted functions. The design of SVC is motivated by hardware verification tasks, but it has also been used in conjunction with the PVS system [24] as a general-purpose decision procedure. Unlike our procedure, SVC is restricted to ground formulas, that is, quantifiers are not considered. Like our procedure, it does not require clausal form.

The Extended Static Checking (ESC) project [22, 15] automatically detects simple run-time errors in annotated Modula-3 programs at compile time. Its underlying theorem-proving procedure also appears similar to ours. Like ours, it can handle quantifiers, and is designed to be efficient rather than complete. Unlike our procedure, this prover uses only clausal form, and restricts quantifier duplication to matching (see Section 3.2).

All three procedures conduct a backtracking search for a falsifying assignment, and have many decision procedures in common (see Section 5).

2 Preliminaries

Formulas and expressions: To facilitate the specification of reactive systems and their properties, STeP uses a first-order language with a rich and flexible syntax. Formulas are in nonclausal form, and boolean formulas can be nested inside arbitrary function symbols (as in *if…then…else* expressions). An essential construct is *let-binding*, which explicitly represents structure sharing within an expression.

Therefore, our expressions will include first-order quantification, the usual set of boolean connectives ($\vee, \wedge, \neg, \rightarrow, \leftrightarrow,$ `if-then-else`), and the construct `let` $x = e_1$ `in` e_2 for a variable x and arbitrary expressions e_1 and e_2. The scope of x is e_2; occurrences of x in e_1 are bound outside the `let` expression. Note that, for instance, OBDD nodes can be represented by `let`-bound variables, allowing a linear translation from OBDDs to a formula with `let`- expressions.

For a given formula $\mathcal{F}$, the *universal closure* of $\mathcal{F}$, written $\forall *.\mathcal{F}$, is the formula $\forall x_1. \ldots \forall x_n.\mathcal{F}$ where $\{x_1, \ldots, x_n\}$ are the free variables of $\mathcal{F}$. The *existential closure* of $\mathcal{F}$, written $\exists *.\mathcal{F}$, is defined similarly.

A *substitution* θ is a mapping $\theta : \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are distinct variables and $t_1, \ldots, t_n$ are terms. $e\theta$ is the result of simultaneously replacing all free occurrences of x_i by t_i in the expression e. We also write $e[t/x]$ for $e\,\{x \mapsto t\}$. Replacement is always *safe*, in that quantifiers are renamed to prevent capture, and bound variables are not replaced (see [21]). For substitutions θ and ρ, $\theta \cdot \rho$ is the substitution such that $x(\theta \cdot \rho) = (x\theta)\rho$. θ is *more general* than ρ if $\theta \cdot \gamma = \rho$ for some γ. The empty substitution is written as $\{\}$.

An *atom* is a formula with no boolean connectives; a *literal* is an atom or its negation. A *top-level conjunct* of a formula $\mathcal{F}$ is one of $\mathcal{F}_i$ if $\mathcal{F}$ is of the form $\mathcal{F}_1 \wedge \cdots \wedge \mathcal{F}_n$, and $\mathcal{F}$ otherwise. A *top-level literal* is a top-level conjunct that is a literal. We write $\mathcal{F}[e]$ for a formula with one or more occurrences of subexpression e, where e does not occur within the scope of a quantifier.

Polarity: We define the *polarity* of a subexpression in $\mathcal{F}$ in the usual way [21]: an occurrence of a subexpression e is *positive* (resp. *negative*) in $\mathcal{F}$ if it occurs within an even (resp. odd) number of negations, written as $\mathcal{F}[e]^+$ (resp. $\mathcal{F}[e]^-$). An occurrence has *both polarities*, written as $\mathcal{F}[e]^\pm$, if it appears under the $\leftrightarrow$ boolean connective or in the if-clause of an `if-then-else` expression. If e has two occurrences in $\mathcal{F}$, one positive, and one negative, we can refer to both occurrences by $\mathcal{F}[e]^\pm$.

$\mathcal{F}[e]^+$, $\mathcal{F}[e]^-$ and $\mathcal{F}[e]^\pm$ respectively denote strictly positive, strictly negative and bipolar occurrences of e in $\mathcal{F}$.

Theories and decision procedures: Our goal is to decide general validity with respect to a background theory or combination of theories $\mathcal{T}$ (not necessarily complete or first-order axiomatizable). Following [4], we define the following semantic properties of formulas:

Definition 1. A closed sentence $\mathcal{F}$ is *$\mathcal{T}$-valid* if every model of $\mathcal{T}$ satisfies $\mathcal{F}$, and *$\mathcal{T}$-unsatisfiable* if no model of $\mathcal{T}$ satisfies $\mathcal{F}$.

Definition 2 ($\mathcal{T}$-complementary). A sentence $\mathcal{F}$ is *$\mathcal{T}$-complementary* if $\exists *.\mathcal{F}$ is $\mathcal{T}$-unsatisfiable.

Definition 3 ($\mathcal{T}$-refuter). θ is a *$\mathcal{T}$-refuter*, or *$\mathcal{T}$-refuting substitution*, for a sentence $\mathcal{F}$ if $\mathcal{F}\theta$ is $\mathcal{T}$-complementary.

The last two notions are extended to sets of formulas by identifying a set with the conjunction of its elements. $\mathcal{T}$-complementary sets of literals in theory reasoning correspond to syntactically complementary pairs of literals in resolution—no instance is satisfiable in the theory.

A *decision procedure* for a theory $\mathcal{T}$ should always be able to identify the $\mathcal{T}$-complementarity of a set of quantifier-free literals.[2] However, if $\mathcal{T}$ is a combination of theories, each with its own decision procedure, we do not expect to obtain a combined decision procedure that is complete for the combined theory (i.e., not all $\mathcal{T}$-unsatisfiable sets will be identified). On some occasions, decision procedures will also be able to provide $\mathcal{T}$-refuting substitutions for a given set of literals. In Section 5 we describe how this can be done in a particular case.

In the rest of this paper, validity and satisfiability will always be understood relative to a theory $\mathcal{T}$, unless it is explicitly stated otherwise.

3 Refutation Search: A General Procedure

For an arbitrary closed formula $\mathcal{G}$, *satisfiability-preserving skolemization* constructs a quantifier-free formula $\mathrm{Sk}(\mathcal{G})$ such that $\forall *.\mathrm{Sk}(\mathcal{G})$ is satisfiable iff $\mathcal{G}$ is satisfiable. $\mathcal{G}$ is valid iff $\neg\mathcal{G}$ is unsatisfiable, which is the case iff $\forall *.\mathrm{Sk}(\neg\mathcal{G})$ is unsatisfiable. This is the case if (but not only if) there is a ground-unsatisfiable instance $\mathrm{Sk}(\neg\mathcal{G})\theta$. Thus, the validity of a first-order formula can be established by finding a substitution for which a given quantifier-free formula is ground-unsatisfiable.

We now present a procedure in which skolemization, instantiation, quantifier duplication and the refutation search are all carried out within a unified framework. The procedure operates on a set $\mathcal{S}$ of formulas $\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, where $\mathcal{S}$ is said to be satisfiable iff $\forall *.(\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$ is satisfiable. To finish a proof we need to show that all of the elements of $\mathcal{S}$ are, in fact, unsatisfiable, under a common instantiation. The abstract procedure proceeds by transforming the set $\mathcal{S}$, at each step applying one of the rules in Figure 1.

[2] Note that decision procedures are not expected to reason about boolean formulas.

succeed	$\emptyset$	$\rightarrow$ refuted
reduce	$\{\mathit{false}\} \cup \mathcal{S}$	$\rightarrow \mathcal{S}$
simplify	$\{\mathcal{F}\} \cup \mathcal{S}$	$\rightarrow \{\mathit{SIMPLIFY}(\mathcal{F})\} \cup \mathcal{S}$
split	$\{\mathcal{F}[e]\} \cup \mathcal{S}$	$\rightarrow \{e = d_i \ \wedge \ \mathcal{F} \mid d_i \in \mathbf{dom}(e)\} \cup \mathcal{S}$
instantiate	$\mathcal{S}$	$\rightarrow \mathcal{S}\theta : \{\mathcal{F}\theta \mid \mathcal{F} \in \mathcal{S}\}$ for some substitution θ
skolemize^{+}	$\left\{\mathcal{F}[\forall x.\varphi]^{+}\right\} \cup \mathcal{S}$	$\rightarrow \left\{\mathcal{F}[\varphi[y/x] \ \wedge \ \forall x.\varphi]^{+}\right\} \cup \mathcal{S}$
skolemize^{-}	$\left\{\mathcal{F}[\forall x.\varphi]^{-}\right\} \cup \mathcal{S}$	$\rightarrow \left\{\mathcal{F}[\varphi[f_x(\overline{y})/x]]^{-}\right\} \cup \mathcal{S}$
skolemize$^{\pm}$	$\left\{\mathcal{F}[\forall x.\varphi]^{\pm}\right\} \cup \mathcal{S}$	$\rightarrow \left\{ \left(\begin{array}{c} a_x(\overline{y}) \wedge \forall x.\varphi \\ \vee \\ \neg a_x(\overline{y}) \wedge \neg\forall x.\varphi \end{array} \right) \wedge \ \mathcal{F}[a_x(\overline{y})]^{\pm} \right\} \cup \mathcal{S}$
let-eliminate	$\{\mathcal{F} \left[\begin{array}{l} \texttt{let}\ x = e_1 \\ \texttt{in}\ e_2 \end{array} \right] \} \cup \mathcal{S}$	$\rightarrow \{f_x(\overline{y}) = e_1 \ \wedge \ \mathcal{F}[e_2[f_x(\overline{y})/x]]\} \cup \mathcal{S}$

Fig. 1. Rules for general $\mathcal{T}$-refuting procedure

- **succeed**: This rule concludes the refutation search.
- **reduce**: *false* can be disregarded in the search for a satisfiable disjunct.
- **simplify**: $\mathcal{F}$ is simplified, possibly to *false*, by the available decision procedures and simplification mechanisms (see Section 5). $\mathit{SIMPLIFY}(\mathcal{F})$ simplifies $\mathcal{F}$ with respect to its top-level literals, producing a formula $\mathcal{T}$-equivalent to $\mathcal{F}$. Minimal requirements for *SIMPLIFY* are:
  - If $e = d_i$ is a top-level literal of the formula, then e occurs nowhere else in the simplified formula.
  - If the top-level literals of the formula are recognized as $\mathcal{T}$-complementary, then the simplified formula is *false*.
- **split**: Subexpressions e taking values from a finite domain $\mathbf{dom}(e)$ can be analyzed according to the domain values. This includes boolean subformulas e, which are split with $e = \mathit{false}$ and $e = \mathit{true}$. In this case, the conjuncts added are $\neg e$ and e, respectively. Special cases of this rule are discussed in Section 4.
- **instantiate**: The substitution θ can instantiate free variables in $\mathcal{S}$ by arbitrary (quantifier-free) terms.
- **skolemize**$^{+}$: y is a fresh variable.[3]
- **skolemize**$^{-}$: $\overline{y}$ is a tuple of all the free variables in $\forall x.\varphi$ and f_x is a fresh function symbol.
- **skolemize**$^{\pm}$: a_x is a fresh predicate symbol, and $\overline{y}$ is a tuple of all the free variables in $\forall x.\varphi$.
- **let-eliminate**: $\overline{y}$ is a tuple of all the free variables in e_1 and f_x is a fresh function symbol.

[3] Similar skolemization rules apply to existential quantifiers, when $\exists x.\varphi$ has the opposite polarity.

3.1 Main Properties

We write $\mathcal{S} \to^* \mathcal{S}'$ if one or more rules transform the set $\mathcal{S}$ into the set $\mathcal{S}'$. We say that a rule *preserves satisfiability* when it transforms $\mathcal{S}$ to $\mathcal{S}'$, if:

$$\forall *.\ \bigvee_{\mathcal{F} \in \mathcal{S}} \mathcal{F} \text{ is } \mathcal{T}\text{-satisfiable} \qquad \text{iff} \qquad \forall *.\ \bigvee_{\mathcal{F} \in \mathcal{S}'} \mathcal{F} \text{ is } \mathcal{T}\text{-satisfiable.}$$

Lemma 4. *Except for* **instantiate***, each rule in Section 3 preserves satisfiability when applied to any set $\mathcal{S}$. If the original set contains only closed formulas, and only these rules are applied, then* **instantiate** *preserves satisfiability as well.*

In practice, we are only concerned with the "only if" direction of satisfiability preservation. This direction is always maintained by the **instantiate** rule, as well as the rule refinements we consider later on. If the old set is satisfiable only if the new one is, then we have:

Theorem 5 (Soundness). *For any closed formula $\mathcal{F}$, if $\{\neg\mathcal{F}\} \to^*$ refuted then $\mathcal{F}$ is $\mathcal{T}$-valid.*

Rules that preserve satisfiability are *invertible*: if $\mathcal{S} \to \mathcal{S}'$ using an invertible rule, then $\mathcal{S} \to^*$ refuted iff $\mathcal{S}' \to^*$ refuted. Lemma 4 tells us that all the rules in Section 3 are invertible. In particular, rules **reduce**, **skolemize**$^-$, and **let-eliminate** should be applied whenever possible, since they reduce the complexity of $\mathcal{S}$ and preserve satisfiability. Finally, we have:

Theorem 6 (First-order completeness). *Let $\mathcal{F}$ be a closed first-order formula. If $\mathcal{F}$ is (generally, or $\emptyset$-) valid then $\{\neg\mathcal{F}\} \to^*$ refuted.*

This follows, for example, from the completeness of the general matings procedure, given a suitable amplification of the formula [1]. As in the case of resolution [26], the completeness of most such procedures relies on Herbrand's theorem to guarantee that an appropriate finite ground instantiation always exists. Herbrand's theorem can be extended to account for certain classes of background theories [16, 4, 17]. Since practical implementations will sacrifice completeness by considering only instantiations with a limited amount of quantifier duplication, (see Section 3.2), we will not be concerned with ensuring that such an extended Herbrand theorem holds.

Theorem 7 (Ground decidability). *Let $\mathcal{F}$ be a closed formula where all occurrences of $\forall$ are strictly positive. If $\{\neg\mathcal{F}\} \to^* \{\mathcal{F}'\} \cup \mathcal{S}$ and $\mathcal{F}'$ is $\mathcal{T}$-consistent, then any $\mathcal{T}$-model for $\mathcal{F}'$ is also a model for $\neg\mathcal{F}$.*

Thus, if we can decide the $\mathcal{T}$-consistency of a formula $\mathcal{F}'$ obtained from the analysis of $\neg\mathcal{F}$, then we can conclude that $\mathcal{F}$ is not valid; a model for $\mathcal{F}'$ is a counterexample.

3.2 Equations, Rewrites and Limited Quantifier Duplication

To narrow the search, one can limit the number of quantifier duplications in rule **skolemize**$^+$. For most practical applications the quantifier need not be duplicated at all, using the following rule:

- **skolemize**$_0^+$**:** $\{\mathcal{F}[\forall x.\varphi]^+\} \cup \mathcal{S} \;\;\to\;\; \{\mathcal{F}[\varphi[y/x]\} \cup \mathcal{S}.$

In this case, rules **skolemize**$_0^+$ and **skolemize**$^\pm$ should take precedence over **split**, and the entire formula is fully skolemized before the search begins.

As a special case of quantifier duplication, conjuncts can be added whenever they are an immediate consequence of a universally quantified top-level literal.[4] A common case is that of equalities: if the formula $\forall *.(s = t)$ is known (variables renamed apart), one can add the rules:

- **rewrite:** $\{\mathcal{F}[e]\} \cup \mathcal{S} \;\;\to\;\; \{e = t\theta \;\wedge\; \mathcal{F}[e]\} \cup \mathcal{S}$ where $e = s\theta$.
- **narrow:** $\{\mathcal{F}[e]\} \cup \mathcal{S} \;\;\to\;\; (\{e = t \;\wedge\; \mathcal{F}[e]\} \cup \mathcal{S})\,\theta$ where $e\theta = s\theta$.

In this way, equations that are not terminating or confluent can be applied step by step. A conditional rewrite rules, which rewrites e to e' under condition c, can be applied yielding $\{(c\theta \to e = e') \;\wedge\; \mathcal{F}[e]\} \cup \mathcal{S}$ or adding the equality $e = e'$ after ensuring that $c\theta$ holds under the assumption $\mathcal{F}[e]$.

3.3 Sequent Calculus

The above presentation is analogous to proof in a Gentzen-style sequent calculus, where each transformation corresponds to a rule, and each element of the set $\mathcal{S}$ is a branch in the proof. To illustrate this, we show how well-founded (transfinite) induction and a *cut rule* can be added in very much the same way they are added to sequent-style calculi. (These rules are not part of our implementation, described in Section 4.)

induction	$\{\mathcal{F}[\forall x.\varphi]^-\} \cup \mathcal{S}$	$\to \left\{\mathcal{F}\left[\forall x.\begin{pmatrix}\forall y.\,(y \prec x \to \varphi[y/x]) \\ \to \varphi\end{pmatrix}\right]\right\} \cup \mathcal{S}$
cut	$\{\mathcal{F}\} \cup \mathcal{S}$	$\to \{\mathcal{G} \wedge \mathcal{F},\, \neg\mathcal{G} \wedge \mathcal{F}\} \cup \mathcal{S}$ for an arbitrary formula $\mathcal{G}$.

In the **induction** rule, $\prec$ should be a well-founded order, and y a fresh variable.

A standard proof-theoretic analysis can demonstrate how to transform an arbitrary Gentzen-style derivation into a derivation of the calculus presented here, and viceversa. Furthermore, a *cut-elimination* theorem holds for the calculus presented here: derivations involving splits on non-atomic formulas can be converted into derivations using only splits on atomic formulas. Uses of rule **cut** can also be eliminated from the first-order (uninterpreted) calculus using a standard cut-elimination procedure.

[4] Quantifier duplication in the ESC system [15] is in the form of such matching, limited by a heuristic bound.

4 Refutation Search: Backtracking Implementation

Following is a description of the nondeterministic refutation search procedure rewritten to suggest a practical implementation that uses depth-first search with backtracking. It assumes the formula has already been skolemized. When successful, $\mathit{REFUTE}\,(\mathcal{F}, \{\})$ returns a $\mathcal{T}$-refuting substitution for $\mathcal{F}$. Our inspiration for this approach is the Davis-Putnam-Loveland-Logemann (DPLL) propositional satisfiability procedure, which is effective and requires little memory.

$\mathit{REFUTE}\,(\mathcal{F}, \sigma_1) =$
    $\mathcal{F}' \leftarrow \mathit{SIMPLIFY}(\mathcal{F}\sigma_1)$
    if $\mathcal{F}' = \mathit{false}$ **then return** σ_1
    else do one of
        instantiate: $\theta \leftarrow$ a substitution
            return $\mathit{REFUTE}\,(\mathcal{F}', \sigma_1 \cdot \theta)$
        split: $e, \{d_1, \ldots, d_n\} \leftarrow$ an expression and possible values
            $\sigma_2 \leftarrow \mathit{REFUTE}\,(e = d_1 \wedge \mathcal{F}', \sigma_1)$
            $\sigma_3 \leftarrow \mathit{REFUTE}\,(e = d_2 \wedge \mathcal{F}', \sigma_2)$
              $\vdots$
            return $\mathit{REFUTE}\,(e = d_n \wedge \mathcal{F}', \sigma_n)$
        amplify: $\mathcal{G} \leftarrow$ an amplification formula
            return $\mathit{REFUTE}\,(\mathcal{G} \wedge \mathcal{F}', \sigma_1)$

The DPLL procedure is an instance of *REFUTE* when $\mathcal{F}$ is in clause form, *SIMPLIFY* implements unit resolution and subsumption, only the *split* operation is used, and e is an atomic formula that occurs in a nonunit clause. The added *instantiate* and *amplify* operations extend the substitution and formula respectively. The approach is reminiscent of the search for general matings [1, 5, 19] except here paths are refuted by $\mathcal{T}$-complementary sets of literals instead of syntactically complementary pairs (cf. theory matings [28]).

The instantiate operation: *instantiate* extends the current substitution σ_1 by a substitution θ chosen "don't know" nondeterministically with backtracking. Ideally, if $\mathcal{F}'$ is unsatisfiable, then θ should be a $\mathcal{T}$-refuting substitution for $\mathcal{F}'$. $\mathcal{T}$-refuters for the top-level literals of $\mathcal{F}'$ can sometimes be found and used as θ.[5] $\mathcal{T}$-refuters include substitutions that make literals complementary by ordinary unification; others may be proposed by the decision procedures (see Section 5). Saving substitutions σ_1 for which *REFUTE* fails enables elimination of redundant work due to duplicate substitutions.

Trying to find a refutation using only the $\mathcal{T}$-refuters one knows about may seem overly optimistic, but it appears to work often enough to be a reasonable approach.

[5] When a substitution known to be a $\mathcal{T}$-refuter of the top-level literals is chosen as θ, the succeeding call on *REFUTE* is guaranteed to succeed immediately and can be optimized away.

A second, less optimistic approach entails enumerating in advance possible values for the variables in the formula. The *instantiate* operation would then be used to generate the space of alternative substitutions. A good, but still incomplete way of finding possible values is to look at the positions of variables in the formula and then find terms that occur in complementary positions. For example, t is a possible value for x if x occurs as argument i of P and t as argument i of $\neg P$. The notion of complementary position can be extended in theory-specific ways, e.g., t and x are in complementary positions in $s \prec t$ and $x \prec y$ (see Section 5.1). Our current focus and examples use the first approach.

The split operation: *split* can select an atom e to split on with possible values *true* and *false* as in the DPLL procedure, provided e or $\neg e$ is not already a top-level literal (to avoid repetition). As an extension of the DPLL procedure, *split* can also select a nonconstant term e to split on with the elements of its finite domain $\{d_1, \ldots, d_n\}$ as values.

Good heuristic selection of what expression to split on can have a dramatic effect on the size of the search space. Unlike the DPLL procedure, we are using nonclausal, nonground formulas, but criteria similar to those used in the DPLL procedure [18] are useful, such as number of occurrences and the length of the shortest clause a literal would occur in if the formula were converted to clause form. Constraint satisfaction heuristics, such as preferring expressions with smaller domains to split on first, can also be used.

In the DPLL procedure, the selection of which atom to split and the order of values to try are "don't care" nondeterministic choices that affect the search space but not completeness. However, this selection can affect whether *REFUTE* succeeds or not. For *REFUTE*, we assume that $\mathcal{T}$-complementarity can be recognized, but not that $\mathcal{T}$-refuters can always be found. For example, the $\mathcal{T}$-complementarity of $P(2)\wedge\neg P(1+1)$ may be recognized without assuming that the decision procedures are also able to propose $\{x \mapsto 1\}$ as a $\mathcal{T}$-refuter of $P(2)\wedge\neg P(x+1)$. When *REFUTE* is applied to $P(2)\wedge Q(1)\wedge(\neg P(x+1)\vee\neg Q(x))$, some search orders would succeed, because $\{x \mapsto 1\}$ is discovered as a unifier for $Q(1)\wedge\neg Q(x)$ before attempting to refute $P(2)\wedge\neg P(x+1)$, while others would fail when the latter subproblem is encountered first. However, backtracking through alternative orders of splitting is combinatorially expensive, so we do not do it and accept this additional source of incompleteness.

The amplify operation: Davis [11] defines *obvious inferences* as those that only require substitution for single instances of the formulas (i.e., no quantifier duplication is needed). The combination of *split* and *instantiate* is complete for obvious first-order inferences. It will also make some obvious $\mathcal{T}$-inferences, though not all. Even if $\mathcal{T}$ consists only of the theory of equality, the undecidability of simultaneous rigid E-unification [14] limits completeness of obvious $\mathcal{T}$-inference procedures.

Using only *split* and *instantiate* is our preferred approach. They are sufficient for the examples in this paper, which we believe are typical problems for STeP. The search space is finite and often small. If quantifier duplication is allowed, the search space would be much larger (with limited duplication) or infinite (with unlimited duplication). The single-instance restriction is a natural one that is readily understood by the user. The restriction is easily circumvented by the user's explicit inclusion of additional copies of the formulas (e.g, by manual application of **skolemize**$^{+}$).

Nevertheless, the *amplify* operation is allowed to do limited quantifier duplication, principally for the purpose of applying rewrites. Rather than duplicating a quantifier "in place", *amplify* is defined to add an amplification formula as a conjunct to the formula being refuted. The amplification formula may be any formula that can be soundly used in the refutation; it will typically be a fresh instance of a rewrite or premise (see Section 3.2).

5 Integration with Decision Procedures

Like the PVS [24], SVC [3] and ESC [15] systems, our decision procedures rely heavily on congruence closure [23] for efficient treatment of ground equality. We also use Shostak's combination of decision procedures [27, 10], where congruence closure is the basis for combining *solvable* theories, such as linear arithmetic.

On this base, we add specialized procedures for partial orders and linear inequalities, built-in simplification, special treatment of datatype constructors and destructors, and conditional and unconditional rewrite rules. These are used in the *SIMPLIFY* component of our implementation.

The decision procedures maintain a specialized data-structure, called the *context*, that is updated incrementally as new constraints are asserted. The context maintains a union-find structure for congruence closure, a transitivity graph for partial orders and a loop-residue structure for linear arithmetic.

Most of the above procedures do not, by themselves, suggest $\mathcal{T}$-refuting substitutions. We now briefly describe a combination of decision procedures for congruence closure and partial orders, and its use in obtaining $\mathcal{T}$-refuting substitutions.

5.1 The Theory of Partial Orders

For a partial order $\prec$, we represent a conjunction of equality and inequality constraints ($t_1 \prec t_2$, $t_1 \preceq t_2$, $t_1 = t_2$, and their negation) using a directed graph $\mathcal{G}$, called the *transitivity graph*. Edges in $\mathcal{G}$ are labeled by either $\preceq$, $\neq$, or $\not\prec$, where the $\neq$-edges are undirected. Vertices are labeled by disjoint sets of terms.

With an arbitrary term t we associate a vertex v_t in the transitivity graph as follows: if there is a vertex v in $\mathcal{G}$ such that $t \in v$, then v_t is v; otherwise, v_t is a new vertex labeled with $\{t\}$.

Constraints are added incrementally to the transitivity graph as follows:

$t_1 = t_2$: Merge the vertices v_{t_1} and v_{t_2}, redirecting all edges from (to) v_{t_1} and v_{t_2} from (resp. to) the new vertex. Label the new vertex with $v_{t_1} \cup v_{t_2}$.

$t_1 \preceq t_2$ ($t_1 \neq t_2$, $t_1 \not\prec t_2$): Update $\mathcal{G}$ adding an edge from v_{t_1} to v_{t_2}, labeled $\preceq$ (resp. $\neq$, $\not\prec$).

$t_1 \prec t_2$**:** Update $\mathcal{G}$ by adding the constraints $t_1 \preceq t_2$ and $t_1 \neq t_2$.
$t_1 \not\preceq t_2$**:** Update $\mathcal{G}$ by adding the constraints $t_1 \not\prec t_2$ and $t_1 \neq t_2$.

Total orders are a special case, where constraints of the form $(t_1 \not\prec t_2)$ are treated as $t_2 \preceq t_1$, so $\not\prec$ edge labels are not used.

A transitivity graph $\mathcal{G}$ can be collapsed to $\mathcal{G}'$ by creating a vertex V in $\mathcal{G}'$ for each maximally strongly connected subgraph (MSCS) $\{v_1, \ldots, v_n\}$ in $\mathcal{G}$, considering only $\preceq$-edges. The new vertex V is labeled with $v_1 \cup \ldots \cup v_n$. An edge (V, r, V') is added to $\mathcal{G}'$ iff there is an edge (v, r, v') in $\mathcal{G}$ such that $v \subseteq V$ and $v' \subseteq V'$. If a reversed $\not\prec$-edge completes a loop formed by $\preceq$ edges, the corresponding nodes are joined in $\mathcal{G}'$ as well. Tarjan's algorithm for finding MSCS's can be used to collapse a transitivity graph in linear time.

A decision procedure results from the following observation: a conjunction of inequality constraints is unsatisfiable in the theory of partial orders iff its associated transitivity graph is collapsed into a *contradictory graph*, one that contains a $(v, \neq, v)$ edge.

The above procedure is tightly integrated with congruence closure by replacing each vertex label t by an equivalence class (a congruence closure node), representing all the terms equal to t under the known set of equalities. Collapsing cycles among $\preceq$ edges can generate new equalities, updating the congruence closure structure, which can then collapse the transitivity graph further. We also extend the notion of a contradictory graph to include inconsistencies detected by other decision procedures integrated into the congruence closure, e.g. in the presence of datatypes and arithmetic.

5.2 Obtaining $\mathcal{T}$-refuting Substitutions

The transitivity graph is not only able to detect ground unsatisfiability, but can also serve as a guide for finding $\mathcal{T}$-refuters. We say there is a *$\prec$-edge* from u to v if $(u, \preceq, v)$ and $(u, \neq, v)$ are edges in $\mathcal{G}$. To find $\mathcal{T}$-refuting substitutions for a set of equalities and inequalities, one can find pairs of vertices $\langle v, w\rangle$ that are connected by a $\prec$-edge in the transitivity graph. If E is the set of known equalities at this point, a substitution θ such that $v\theta = w\theta$ under the equalities $E\theta$ is a $\mathcal{T}$-refuter; that is, θ should be a *rigid E-unifier* [17] of v and w.

This approach to finding $\mathcal{T}$-refuters is clearly not complete. A more thorough but still incomplete approach is to consider a pair $\langle v_1, w_1\rangle$ connected by a $\preceq$-path containing a $\prec$-edge, and another pair $\langle w_2, v_2\rangle$ connected by a $\preceq$-path. A substitution θ that is a rigid E-unifier for $\{v_1 = v_2, w_1 = w_2\}$ will also be a $\mathcal{T}$-refuter.[6]

In the general case, the transitivity graph can be searched to find a sequence of paths and a unifier θ that concatenates the paths into a loop containing a $\prec$-edge. Since θ will be obtained incrementally in a congruence closure context, we define the following:

[6] Edges labeled $\not\prec$ and single $\neq$ edges can be similarly used to obtain $\mathcal{T}$-refuting substitutions. To simplify the exposition, we omit these cases.

Definition 8. Given a substitution θ and a congruence closure structure CC, a substitution ϕ is a *θ-compatible rigid CC-unifier* of congruence classes v_1 and v_2 iff ϕ is less general than θ and ϕ is a rigid E-unifier of t_1 and t_2 for some $t_1 \in v_1$ and $t_2 \in v_2$, where E is the set of equations implicit in CC. We write $E\_mgus(CC, \theta, v_1, v_2)$ for a set of θ-compatible rigid CC-unifiers of v_1 and v_2.

Rigid E-unification is NP-complete [17]. In practice, we are content with quickly identifying a subset of the rigid E-unifiers.[7] We collect E-unifiers using a fast, but again incomplete, test to eliminate redundant substitutions.

The procedure *EXPAND* defined below updates a set S of $\mathcal{T}-$refuting substitutions for the theory of partial orders. It searches the set of paths in the transitivity graph examining one sequence of paths at most once. $TC(v)$ denotes the *$\preceq$-transitive closure* from vertex v, i.e., the set of vertices reachable from v by $\preceq$-edges. $TC^+(v)$ is $TC(v) \setminus \{v\}$. $\mathcal{V}(\mathcal{G})$ is the set of vertices of $\mathcal{G}$. $\mathcal{CC}(\mathcal{G})$ is the congruence closure structure associated with $\mathcal{G}$. After the invocation $EXPAND(\mathcal{G}_0, v_0, v_0, \{\})$, each recursive call $EXPAND(\mathcal{G}, v_1, v_2, \theta)$ maintains the invariants (a) $v_2 \in TC(v_1)$ in $\mathcal{G}$, and (b) $\mathcal{G}$ is obtained from $\mathcal{G}_0$ by asserting the equalities given by θ. This is ensured by the function *add_substitution*, which merges nodes and collapses the graph as described above. The vertices v_1' and v_4' are the counterparts of v_1 and v_4 in $\mathcal{G}'$. *EXPAND* must terminate, since the size of V_1 decreases with each recursive call.

$S \leftarrow \emptyset$; $EXPAND(\mathcal{G}_0, v, v, \{\})$; **return** S; where:
$EXPAND(\mathcal{G}, v_1, v_2, \theta) =$
 $V_1 \leftarrow \mathcal{V}(\mathcal{G}) \setminus TC^+(v_1)$
 $V_2 \leftarrow TC^+(v_2)$
 for each $(v_3, v_4) \in V_1 \times V_2$ **do**
  $S' \leftarrow E\_mgus(\mathcal{CC}(\mathcal{G}), \theta, v_3, v_4)$
  for each $\theta' \in S'$ **do**
   $\mathcal{G}' \leftarrow add\_substitution(\theta', \mathcal{G})$
   if $\mathcal{G}'$ is a contradictory graph
    then $S \leftarrow S \cup \{\theta'\}$
    else $EXPAND(\mathcal{G}', v_1', v_4', \theta')$

In the worst case, *EXPAND* will search exponentially many paths. However, the moderate size of transitivity graphs arising from typical verification conditions, and the incremental unification restriction, make the procedure practical.

More general linear arithmetic inequalities are represented in a structure similar to a transitivity graph. The search for $\mathcal{T}$-refuters proceeds in a way reminiscent to that of partial orders.

6 Examples

The STeP system is implemented in Standard ML of New Jersey. The times below, measured on a 200Mhz ULTRA-Sparc SUN workstation, are only intended

[7] Note that at this point we do not attempt, or need to, solve a (undecidable) simultaneous rigid E-unification problem.

to convey a rough idea of the speed of the procedure. We first show the performance of our backtracking implementation on some small but representative formulas:

1. Theory of partial orders: the validity of

$$(\forall x.(x \preceq y \rightarrow P(x))) \wedge (\forall u.\exists z.z \prec u) \rightarrow \exists v.(v \prec y \wedge P(v))$$

is established in 0.07 seconds.
2. Theory of integers and rationals:

$$\forall t, s. \, [(\forall x.x \leq t \rightarrow P(x)) \wedge s \leq t/2 \ \rightarrow \forall y.(y \leq s \rightarrow P(t/2 + y))]$$

is proved valid in 0.04 seconds. The variables t, s, x, y range over the rationals. The formula

$$\forall t, s. \left[(\forall x.x \leq t \rightarrow P(x)) \wedge s \leq t/2 \ \rightarrow \forall y.(y \leq s \rightarrow P(\boxed{t/3} + y))\right]$$

is *not* valid, and our procedure fails after 0.03 seconds, correctly suggesting the counter-example $t = -2, s = y = -1$ for the universal-force variables.
3. Theory of total orders and arrays, making essential use of the procedure *EXPAND* (Section 5.2):

$$\begin{bmatrix} \forall i.y[i] \preceq \max(y) \\ \wedge \\ \forall a.\mathrm{inc}(a) \succ a \end{bmatrix} \rightarrow \forall k. \begin{bmatrix} \forall j.(k \preceq y[j]) \\ \rightarrow \\ \forall i_1, i_2.k \preceq \mathrm{update}(y, \mathrm{inc}(\max(y)), i_1)[i_2] \end{bmatrix}$$

This formula is proved valid in 0.37 seconds.

Verification examples: If a formula contains only universal force quantifiers, the generation of instantiations is not invoked, and the decision procedures only need to decide ground validity. The properties of a look-up table used in the hardware SRT division algorithm investigated in [9] fall under this class. Our procedure can prove the validity of such a property, which requires decision procedures for linear arithmetic and tuples in essential ways, by checking over 4,700 branches in 30 seconds.

Finally, we consider the verification conditions generated in the deductive verification of a parameterized real-time railroad-crossing example [7]. We tested our procedure on 75 verification conditions from this benchmark. Of these, 60 originally required interactive verification, where the user provided trivial quantifier instantiations and selected the right axioms from a set of background properties. Using the new procedure, all but 5 are proved automatically; the remaining still require a few obvious manual quantifier instantiations before the first-order procedure can automatically establish them.

We should note that the running times increase rapidly as the number of quantified background properties grow. The largest number of background axioms used in a successful run was 15, containing 9 existential force quantifiers.

	background axioms	# atoms	variables instantiated	time (seconds)
1	8	18	4	0.68
2	8	21	4	0.95
3	9	22	4	14.6
4	10	21	5	10.97
5	10	28	4	—

Fig. 2. Results on verification conditions for a real-time safety property.

The theories essential in establishing the validity of these verification conditions include arrays and datatypes (including tuples and records), linear arithmetic, and equality, which, combined, interact with quantification.

Figure 2 describes the performance of our procedure in discharging verification conditions needed to verify a safety property of this system. The set of background axioms contained ground formulas and 3-4 quantifiers, for a total of 15-20 additional atoms (not counted in the figure). Formula 5 corresponds to a verification condition that was *not* established automatically (in a reasonable amount of time). After instantiating two of the 4 quantifiers in the interactive prover, the resulting formula was automatically established in 1.6 seconds.

More information about the implemented procedure is available at
`http://theory.stanford.edu/~nikolaj/cade97.html`.

Acknowledgements: We thank Mark Pichora, Bernd Finkbeiner and Uri Lerner for their comments and suggestions.

References

1. Andrews, P. B. Theorem proving via general matings. *J. ACM 28*, 2 (Apr. 1981), 193–214.
2. Baader, F., and Siekmann, J. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Oxford University Press, Oxford, UK, 1993.
3. Barrett, C., Dill, D. L., and Levitt, J. Validity checking for combinations of theories with equality. In *1st Intl. Conf. on Formal Methods in Computer-Aided Design* (Nov. 1996), vol. 1166 of *LNCS*, pp. 187–201.
4. Baumgartner, P., Furbach, U., and Petermann, U. A unified approach to theory reasoning. Research Report 15-92, Fachbereich Informatik, Universität Koblenz, 1992.
5. Bibel, W. *Automated Theorem Proving.* Friedr. Vieweg & Sohn, Braunschweig, Germany, 1982.
6. Bjørner, N. S., Browne, A., Chang, E. S., Colón, M., Kapur, A., Manna, Z., Sipma, H. B., and Uribe, T. E. STeP: Deductive-algorithmic verification of reactive and real-time systems. In *Proc.* 8^{th} *Intl. Conference on Computer Aided Verification* (July 1996), vol. 1102 of *LNCS*, Springer-Verlag, pp. 415–418.
7. Bjørner, N. S., Manna, Z., Sipma, H. B., and Uribe, T. E. Deductive verification of real-time systems using STeP. In *4th Intl. AMAST Workshop on Real-Time Systems* (May 1997), vol. 1231 of *LNCS*, Springer-Verlag, pp. 22–43.

8. Bürckert, H.-J. *A Resolution Principle for a Logic with Restricted Quantifiers*, vol. 568 of *LNAI*. Springer-Verlag, 1991.
9. Clarke, E.M., German, S., and Zhao, X. Verifying the SRT division algorithm using theorem-proving techniques. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification* (July 1996), vol. 1102 of *LNCS*, Springer-Verlag, pp. 111–122.
10. Cyrluk, D., Lincoln, P., and Shankar, N. On Shostak's decision procedure for combinations of theories. In *Proc. 13th Int. Conf. on Automated Deduction* (1996), vol. 1104 of *LNCS*, Springer-Verlag.
11. Davis, M. Obvious logical inferences. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (August 1981), pp. 530–531.
12. Davis, M., Logemann, G., and Loveland, D. W. A machine program for theorem-proving. *Communications of the ACM 5*, 7 (July 1962), 394–397.
13. Davis, M., and Putnam, H. A computing procedure for quantification theory. *J. ACM 7* (1960), 201–215.
14. Degtyarev, A., and Voronkov, A. Simultaneous rigid E-unification is undecidable. UPMAIL Technical Report No. 105, Computing Science Department, Uppsala University, 1995.
15. Detlefs, D. An overview of the extended static checking system. In *Proc. First Workshop on Formal Methods in Software Practice* (Jan. 1996), ACM (SIGSOFT), pp. 1–9.
16. Frisch, A. M. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence 49* (1991), 161–198.
17. Gallier, J., Narendran, P., Raatz, S., and Snyder, W. Theorem proving using equational matings and rigid E-unification. *J. ACM 39*, 2 (Apr. 1992), 377–429.
18. Hooker, J. N., and Vinay, V. Branching rules for satisfiability. *J. Automated Reasoning 15* (1995), 359–383.
19. Issar, S. Path-focused duplication: A search procedure for general matings. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (July–August 1990), pp. 221–226.
20. Manna, Z., and Pnueli, A. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.
21. Manna, Z., and Waldinger, R. *The Deductive Foundations of Computer Programming.* Addison-Wesley, Reading, MA, 1993.
22. Nelson, G., and Detlefs, D. ESC pages and *Simplify* man page. On-line documentation, DEC Systems Research Center, 1996. `http://www.research.digital.com/SRC/esc/Esc.html`.
23. Nelson, G., and Oppen, D. C. Fast decision procedures based on congruence closure. *J. ACM 27*, 2 (Apr. 1980), 356–364.
24. Owre, S., Rajan, S., Rushby, J., Shankar, N., and Srivas, M. K. PVS: Combining specification, proof checking, and model checking. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification* (July 1996), vol. 1102 of *LNCS*, Springer-Verlag, pp. 411–414.
25. Plotkin, G. Building in equational theories. *Machine Intelligence 7* (1972), 73–90.
26. Robinson, J. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (1965), 23–41.
27. Shostak, R. Deciding combinations of theories. *J. ACM 31*, 1 (Jan. 1984), 1–12.
28. Stickel, M. E. Automated deduction by theory resolution. *J. Automated Reasoning 1*, 4 (1985), 333–355.