Abstraction-Driven Verification
of Array Programs

David Déharbe\textsuperscript{1}, Abdessamad Imine\textsuperscript{2}, and Silvio Ranise\textsuperscript{2}

\textsuperscript{1} UFRN/DIMAp, Natal, Brazil
david@dimap.ufrn.br
\textsuperscript{2} LORIA & INRIA-Lorraine, Nancy, France
{imine,ranise}@loria.fr

Abstract. We describe a refutation-based theorem proving algorithm capable of checking the satisfiability of non-ground formulae modulo (a combination of) theories. The key idea is the use of abstraction to drive the application of (i) ground satisfiability checking modulo theories axiomatized by equational clauses, (ii) Presburger arithmetic, and (iii) quantifier instantiation. A prototype implementation is used to discharge the proof obligations necessary to show the correctness of some typical programs manipulating arrays. On these benchmarks, the prototype automatically discharge more proof obligations than \textit{Simplify} – the prover of reference for program checking – thereby confirming the viability of our approach.

1 Context and Motivation

Satisfiability procedures for equality and theories of standard data-types, such as arrays, lists, and arithmetic are at the core of most state-of-the-art verification tools (e.g., DPLL(\(T\)) \cite{6} and \textit{Simplify} \cite{3}). These are required for a wide range of verification tasks and are fundamental for efficiency. Satisfiability problems have the form \(T \land \phi\), where \(\phi\) is a Boolean combination of ground literals, \(T\) is a background theory, and the goal is to prove that \(T \land \phi\) is unsatisfiable. A satisfiability procedure for a theory \(T\) is an algorithm capable of checking whether \(T \land \phi\) is satisfiable or not, for any ground formula \(\phi\).

The task of designing, proving correct, and implementing satisfiability procedures for decidable theories of practical interest is quite difficult. First, most problems involve more than one theory, so that one needs to combine satisfiability procedures (see e.g., \cite{7}). Second, every satisfiability procedure needs to be proved correct: a key step is to show that whenever the algorithm reports “satisfiable,” its final state represents a model of \(T \land \phi\). Unfortunately, model-construction arguments can be quite complex (see e.g., \cite{10}).

Although designing and combining decision procedures are necessary and very important activities to build practically useful reasoning tools, they are not sufficient. In fact, many proof obligations encountered in routine verification problems require a degree of flexibility which is not provided by actual state-of-the-art tools. The main problem is that only a tiny portion of such proof obligations falls exactly into the domain the procedures are designed to solve. As an
example, consider a satisfiability procedure for the union of (the quantifier-free fragment of) the theory $E$ of equality, the quantifier-free fragment of Presburger arithmetic $PA$, and the formula

$$a < b \land \max(a,b) = a \land \forall X,Y.(X < Y \Rightarrow \max(X,Y) = Y).$$

(1)

The available procedure will fail to detect the unsatisfiability of (1) since it does not know how to instantiate the quantified variables. Here, we propose a mechanism to augment decision procedures to cope with quantifiers.

## 2 Abstraction-Driven Refutation Theorem Proving

For lack of space, we assume the basic notions of first-order logic [4], the superposition calculus [8], and the Nelson-Oppen combination schema [7].

Handling Ground Formulae. Recently, there has been a lot of interest around theorem proving algorithms to discharge the proof obligations arising in various verification problems, which are large ground formulae with a complex Boolean structure to be checked satisfiable modulo a background theory $T$. An integration of propositional solving (SAT, for short) and satisfiability checking modulo $T$ has been advocated to efficiently discharge these formulae. The abstract-check-refine algorithm underlying such integrations is depicted in Figure 1, where $gfol2prop(\phi_g)$ returns the propositional abstraction of the ground formula $\phi_g$ (e.g. the abstraction of $(a = b \land b = c) \Rightarrow f(a) = f(c)$ is $(p \land q) \Rightarrow r$, $p$ is the abstraction of $a = b$, $q$ of $b = c$, and $r$ of $f(a) = f(c)$), $prop2gfol$ is its inverse, and $check\_assign$ is such that $check\_assign(T, \beta) = unsat$ iff $\beta$ is unsatisfiable modulo $T$; otherwise, $check\_assign(T, \beta) = sat$. Many refinements are necessary to make this schema efficient (see e.g. [6] for details).

Handling Non-ground Formulae. Although the algorithm of Figure 1 has proved to be very effective for hardware verification problems (see again [6] for experimental evidence of this), its applicability in program verification is limited.

```
function check\_ground (\phi_g: ground formula)  
  \phi^p \leftarrow gfol2prop(\phi_g)  
  \text{while } \phi^p \neq false do  
     \text{begin}  
     \beta^p \leftarrow \text{pick a propositional assignment of } \phi^p  
     \rho \leftarrow check\_assign(T, prop2gfol(\beta^p))  
     \text{if } \rho = sat \text{ then return sat}  
     \phi^p \leftarrow \phi^p \land \neg gfol2prop(\beta^p)  
  \text{end}  
  \text{return unsat}  
end
```

Fig. 1. A refutation-based algorithm for ground formulae