Practical Proof Reconstruction for First-order Logic and Set-Theoretical Constructions

Clément Hurlin, INRIA Sophia-Antipolis
clement.hurlin@sophia.inria.fr
Amine Chaieb, Tjark Weber, Technische Universität München
{chaieb,webertj}@in.tum.de
Pascal Fontaine, Stephan Merz, INRIA Lorraine and Nancy University
{pascal.fontaine,stephan.merz}@loria.fr

July 30, 2007

Abstract

Proof reconstruction is a technique that combines an interactive theorem prover and an automatic one in a sound way, so that users benefit from the expressiveness of the first tool and the automation of the latter. We present an implementation of proof reconstruction for first-order logic and set-theoretical constructions between the interactive theorem prover Isabelle and the automatic SMT prover haRVey.

1 Introduction

Machine assistance for theorem proving can be classified into two broad categories. Automatic provers, including SMT solvers [DFR, DdM, NO05] and first-order provers [Sch04, RV02, WBH$^+$02], can solve satisfiability and validity problems in restricted languages. They emphasize efficiency and indeed can be impressively powerful, provided the verification problem falls into their domain. Interactive proof assistants emphasize expressiveness and ease of modeling, but provide relatively modest automation, usually in the way of pre-defined or user-definable proof tactics. From a software engineering point of view, the implementation of interactive proof assistants in the LCF tradition such as Coq [BC04] or Isabelle [NPW02] is based on a small trusted kernel, which gives high assurance in the soundness of the proved theorems. Automatic provers are large and employ aggressive optimizations of code and data structures, making it hard to give formal soundness guarantees.

In practice, it is desirable to combine automatic and interactive tools for deduction in order to benefit from the advantages of both. For example, in the context of system verification it is desirable to have an expressive modeling language so that specifications are readable and easy to validate. During verification, individual proof obligations often fall within decidable fragments such as fragments of arithmetic, arrays or simple set theory, that are tedious to prove in a proof assistant but can be handled by an appropriate automatic prover. It would be desirable to delegate the proof of a theorem that falls within the domain of an automatic tool. However, naive combinations where trusted external tools are called as "oracles" are error-prone. In particular, translation functions to convert formulas from, say, higher-order logic into the SMT-LIB [RT06] input language of SMT solvers present subtle problems due to typing issues or different interpretations of elementary operators. Subtle bugs may compromise soundness, as the authors have witnessed.

Because a full verification of the external tool and the necessary translations is out of the question, the solution for making automated reasoners accessible from within an interactive proof assistant without compromising soundness is to rely on proof reconstruction. In a nutshell, the automatic tool outputs not only the answer, but also a proof trace that is certified by the kernel of the proof assistant. The basic technique has been known for many years. For example, Isabelle contains a first-order reasoner based on a tableau procedure [Pau99], as well as a decision procedure for linear arithmetic [CN06] that cooperate with the kernel in this way. Coq uses certification to interact with the Elan rewrite engine [NKK02]. McLaughlin et al. [MBG06] describe a combination of the SMT solver CVC Lite and the proof assistant HOL-Light that covers the theories of arrays and of linear arithmetic. Meng et al. [MQP04] have worked on integrating resolution-based provers for first-order logic with Isabelle. We described the integration of proof-producing SAT solvers with Isabelle in [Web05] and extended it in [FMM$^+$06] to perform proof reconstruction for the theory of equality with uninterpreted function symbols.

In this paper we extend our previous work by techniques for handling quantified formulas. As a particularly useful application, these techniques allow us to automatically verify formulas of simple set theory, *e.g.*

$$\big(X \cap Y = \emptyset \wedge X \setminus Z = X\big) \Rightarrow X \cap (Y \cup Z) = \emptyset. \qquad (1)$$

Set-theoretic formulas arise frequently in the use of formal methods such as TLA$^+$ [Lam02] or B [CM03], which are useful for the verification of high-level system models. We use formula (1) as a guidance throughout the paper although we aim to provide automation for much larger formulas. Our techniques have been implemented for combining the SMT solver haRVey [DFR] with Isabelle, but they are independent of the particular proof strategy used by haRVey. To maintain the spirit of our previous work, we target a procedure which scales well, is sound, and is completely automatic. Because Isabelle as well as haRVey provide deductive capabilities, we are sometimes presented with a choice of performing certain steps within Isabelle (as part of pre-processing) or within haRVey (subject to subsequent certification by Isabelle), and we will discuss these trade-offs and motivate our choices.

The paper is organized as follows: Section 2 introduces elementary concepts of first-order logic. Section 3 describes the language handled by our techniques and presents the overall combination schema. Section 4 presents proof reconstruction in detail for subsets of first-order logic of increasing strength, and Section 5 concludes the paper.

2 Notations

A first-order language $\mathcal{L} = (\mathcal{V}, \mathcal{F}, \mathcal{P})$ consists of enumerable sets of variables $\mathcal{V}$, function symbols $\mathcal{F}$, and predicate symbols $\mathcal{P}$. Function and predicate symbols are assigned an arity; nullary predicates are propositions, nullary functions are constants. The set of terms over $\mathcal{L}$ is defined in the usual way. A ground term is a term without variables. Atomic formulas are of the forms $t = t'$, for two terms t and t', or $P(t_1, \ldots, t_n)$ where P is an n-ary predicate symbol and t_1, $\ldots$, t_n are terms (for nullary predicate symbols, empty parentheses are omitted). Formulas are built from atomic formulas by applying the propositional connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\equiv$, and the quantifiers $\forall$ and $\exists$. A formula without quantifiers is said to be quantifier-free.

An interpretation $\mathcal{I}$ for a first-order language defines a non-empty universe $|\mathcal{I}|$, and assigns a value $\mathcal{I}[c] \in |\mathcal{I}|$ to every constant c. Each function (resp., predicate) symbol in $\mathcal{L}$ is interpreted by a total function (resp., Boolean-valued function) of suitable arity. A standard inductive definition extends this interpretation in order to define a value $\mathcal{I}[t] \in |\mathcal{I}|$ for every ground term t and a truth value $\mathcal{I}[\varphi] \in \{\top, \bot\}$ for every closed formula φ. A model for a formula is

$$\begin{tikzcd}
\text{Isabelle} & & \varphi\text{ (validity)} \arrow[d] & \\
& & \psi \Rightarrow \bot \arrow[dl] \arrow[dr] & \\
\text{haRVey} & \psi\text{ unsatisfiable} \arrow[d] & & \psi\text{ satisfiable} \arrow[d] \\
\text{Isabelle} & \text{proof reconstruction} & & \text{failure/ counterexample}
\end{tikzcd}$$

Figure 1: Outline of proof reconstruction.

an interpretation that makes the formula true. A formula is satisfiable if it has a model and unsatisfiable otherwise.

In refutation mode (when one is interested in the satisfiability of a formula), existential quantifiers with a positive polarity and universal quantifiers with a negative polarity are called essentially existential quantifiers. Dually, universal quantifiers with a positive polarity and existential quantifiers with a negative polarity are called essentially universal quantifiers. A Skolem formula is a formula with only essentially universal quantifiers. It is always possible to transform a given formula into an equi-satisfiable Skolem formula; this transformation is known as Skolemization [BEL01].

3 Cooperation Between Isabelle and haRVey

The cooperation between the proof assistant Isabelle and the SMT solver haRVey is illustrated in Figure 1. Suppose we wish to use haRVey to prove some first-order formula φ. Because SMT solvers decide satisfiability of formulas, we pass $\psi' = \neg\varphi$ to haRVey. If haRVey finds ψ to be satisfiable, φ cannot be determined to be valid and the proof fails. In fact, haRVey produces a satisfying assignment, which can be displayed to the user as a counter-example. (Note in particular that φ may contain some function or predicate symbols that are left uninterpreted by haRVey, so φ may indeed be valid.) If, however, haRVey determines ψ to be unsatisfiable, it produces a proof that is then passed to Isabelle for certification.

Actually, we have some more flexibility in designing the interaction between Isabelle and haRVey: ψ need not literally be $\neg\varphi$, but can be an equi-satisfiable formula. Technically, our `rv` proof method starts by the application of an initial Isabelle tactic that reduces the proof of φ to the proof of $\psi \Rightarrow \bot$. The formula ψ is then passed to haRVey, and if it is indeed unsatisfiable, the `rv` method finishes by interpreting the proof trace provided by haRVey inside the Isabelle kernel in order to prove $\psi \Rightarrow \bot$ and thus close the initial goal.

Our target language of simple set theory includes the usual set operators: $\in$, $\emptyset$, Ω, $\cap$, $\cup$, $\setminus$, $\subset$, $\subseteq$, set equality, set enumeration, and set comprehension. By encoding sets as unary predicates (representing their characteristic function), we can reduce this language to standard first-order logic formulas where given sets are encoded by uninterpreted predicate symbols. It can be shown [Fon07] that quantifier-free formulas over this language (and also over a slightly richer language that allows relations and tuples) belong to a decidable fragment of first-order logic, and indeed we can be sure that haRVey will terminate when applied to such a formula.

As an example, the formula (1) in the introduction becomes

$$\big((\forall x\ .\ (X(x) \wedge Y(x)) \equiv \bot) \wedge \big(\forall x\ .\ (X(x) \wedge \neg Z(x)) \equiv X(x))\big)\big) \\ \Rightarrow \forall x\ .\ \big(X(x) \wedge (Y(x) \vee Z(x))\big) \equiv \bot \tag{2}$$

after translation to first-order logic.

Even when starting from a quantifier-free formula, the translation from set theory to first-order logic introduces quantification. It is therefore essential that our `rv` method handles quantified formulas, which was not the case of our previous work [FMM+06]. In our following exposition of proof reconstruction, we will therefore focus on the treatment of quantifiers. The version of haRVey that underlies our work handles essentially universal quantifiers by instantiation, and it implements Skolemization to eliminate essentially universal quantifiers.

As explained above, we have a choice of relying on haRVey to handle (some) quantifiers or to take care of them during the pre-processing step performed within Isabelle, and this is an important trade-off. We chose to leave the (expensive) instantiation step, which involves search, to haRVey and implement the (comparatively cheap and deterministic) Skolemization in Isabelle. Beyond Skolemization, pre-processing also takes care of generating a formula in conjunctive normal form (CNF), as haRVey expects clauses as input.

4 Proof Reconstruction

Checking the satisfiability (or the validity) of a quantifier-free formula containing set operators involves several successive reasoning or transformation steps. First, the operators are replaced by their definitions as λ-expressions (see [Fon07]). Second, those λ-terms are β-reduced to obtain a pure first-order formula, possibly with quantifiers. Third, this formula is Skolemized. Then instantiation eliminates the remaining quantifiers. Finally, the obtained quantifier-free formula is handled by a decision procedure based on congruence closure and SAT-solving techniques.

Each step of this process should either be implemented directly within Isabelle as part of the pre-processing of formulas, or be performed by haRVey. In the latter case, a proof trace should be produced that allows Isabelle to reconstruct the proof. Proof reconstruction for the final step, the satisfiability checking of quantifier-free formulas with uninterpreted symbols and equality, was presented in our previous work [FMM+06]. We now explain how we handle Skolemization, instantiation, and the transformation of formulas with set operators into first-order formulas.

4.1 Skolemization

Skolemization eliminates the essentially existential quantifiers in a formula, to obtain an equi-satisfiable formula with essentially universal quantifiers only. For instance, consider a formula $\forall y \exists x. \psi(x, y)$; it is equi-satisfiable to formula $\forall y. \psi(f(y), y)$, where f is a new function symbol. Repetitive application of this process to every essentially existential quantifier gives a Skolem form equi-satisfiable to the original formula [BEL01].

Consider again the toy formula (1), that is transformed to the first-order formula (2); Skolemization eliminates the universal (but essentially existential) quantifier in the right-hand side of the implication:

$$\big((\forall x\ .\ (X(x) \wedge Y(x)) \equiv \bot) \wedge \big(\forall x\ .\ (X(x) \wedge \neg Z(x)) \equiv X(x))\big)\big) \\ \Rightarrow \big(X(s) \wedge (Y(s) \vee Z(s))\big) \equiv \bot. \tag{3}$$

There exist different Skolemization techniques that differ in the arity of the introduced function symbol. The haRVey solver implements inner Skolemization [NW01]: inner-Skolemizing a formula φ consists in replacing every occurrence of subformulas of the form $Qx.\psi(x)$ (where Q

is a essentially existential quantifier) in φ by $\psi(f(y_1,\ldots,y_n))$, where f is a new function symbol and $\{y_1,\ldots,y_n\}$ is the set of free variables of the subformula $Qx.\psi(x)$. For outer Skolemization [NW01], the arguments of the Skolem term are rather the variables of the quantifiers having $Qx.\psi(x)$ in their scope.

For the combination of haRVey and Isabelle, we have to decide if Skolemization should be part of the pre-processing performed by Isabelle or part of the proof search performed by haRVey. In the latter case, haRVey's inner Skolemization procedure would have to be replayed by Isabelle during proof certification, which is delicate for two reasons. First, performing a step of inner Skolemization requires access to subformulas that can be nested deeply within the current goal, and Isabelle only provides access to the top-most symbol of a formula. Second, the notion of free variables of a formula is not directly accessible in Isabelle.

The essential Isabelle rule to implement Skolemization is

$$\frac{\Gamma, \forall x.\exists y.P(x,y) \vdash \Delta}{\Gamma, \exists f.\forall x.P(x,f(x)) \vdash \Delta}\ \text{choice}$$

and, combined with $\exists$-elimination, this rule removes one essentially existential quantifier. As an example, to Skolemize $\exists y$ in the formula $\forall x.P \wedge \exists y.Q(x,y)$ appearing on the left-hand side of a sequent, one first needs to move the quantifier outside the Boolean structure of the formula, use the above rule, and finally eliminate the resulting quantifier over the Skolem function:

$$\begin{array}{cl}
\Gamma, \forall x.P \wedge \exists y.Q(x,y) \vdash \Delta & \\
\Downarrow & \text{extra-scope } \exists \\
\Gamma, \forall x.\exists y.P \wedge Q(x,y) \vdash \Delta & \\
\Downarrow & \text{choice} \\
\Gamma, \exists f.\forall x.P \wedge Q(x,f(x)) \vdash \Delta & \\
\Downarrow & \exists\text{-elimination} \\
\Gamma, \forall x.P \wedge Q(x,f(x)) \vdash \Delta &
\end{array}$$

In the following, slightly more complicated example, the combination of rules used above introduces a binary function symbol whereas inner Skolemization as implemented in haRVey would generate a unary one because z is not a free variable of $\exists y.Q(x,y)$.

$$\begin{array}{cl}
\Gamma, \forall x.\forall z.P(x,z) \wedge \exists y.Q(x,y) \vdash \Delta & \\
\Downarrow & \text{extra-scope } \exists \\
\Gamma, \forall x.\forall z.\exists y.P(x,z) \wedge Q(x,y) \vdash \Delta & \\
\Downarrow & \text{choice} \\
\Gamma, \forall x.\exists f.\forall z.P(x,z) \wedge Q(x,f(z)) \vdash \Delta & \\
\Downarrow & \text{choice} \\
\Gamma, \exists g.\forall x.\forall z.P(x,z) \wedge Q(x,g(x,z)) \vdash \Delta & \\
\Downarrow & \exists\text{-elimination} \\
\Gamma, \forall x.\forall z.P(x,z) \wedge Q(x,g(x,z)) \vdash \Delta &
\end{array}$$

A faithful implementation of inner Skolemization in Isabelle would require more expensive formula transformations, making proof reconstruction more costly.

Moreover, Skolemization is a deterministic computation that does not involve any search. We have therefore decided to perform Skolemization as part of the initial pre-processing step inside Isabelle, and use an existing Skolemization procedure implemented in Isabelle that performs outer Skolemization, similar to the strategy outlined above. We have experimented with both inner and outer Skolemization and have observed that there is practically no measurable influence on the running times for our examples.

4.2 Instantiation

It is simple to build an automatic procedure for Skolem formulas that only contain equalities, and uninterpreted symbols on top of a decision procedure for quantifier-free formulas with equalities and uninterpreted symbols. An increasing number of SMT-solvers use *instantiation* to deal with the remaining essentially universal quantifiers. This efficiently handles large Boolean formulas with a few quantifiers, whereas the traditional resolution provers are more dedicated to problems where the quantifiers play much more crucial role. In the present case, where instantiating by a small number of terms is always sufficient, a procedure built on instantiation gives very good results in practice. Below, we detail how we integrated this technique in our framework. As in our previous work, our technique relies on Boolean abstraction to keep an approach that scales.

The primary goal of SMT-solvers is to check the satisfiability of quantifier-free formulas on a language containing equalities, uninterpreted symbols, and symbols for a combination of decidable theories. Handling quantifiers is not deeply embedded into the deduction engine. In haRVey, quantified formulas are simply abstracted by Boolean propositions, and refined on-demand. For instance, assume that a formula $\varphi(\forall x.x = a)$ is given to the solver, where $\forall x.x = a$ is the only quantified subformula. First, $\varphi(p)$ will be checked for satisfiability, where p is a new proposition. If the result is satisfiable, and if p is true in the model, the formula is refined by adding a lemma $p \Rightarrow t = a$ where t is some term found in $\varphi(p)$. The formula $\varphi(p) \wedge (p \Rightarrow t = a)$ is then checked for satisfiability. Continuing in a similar fashion, new instances are added until the resulting formula is found to be unsatisfiable or no new instance can be generated, in which case the original formula is satisfiable. (In practice, generation of new instances is bounded by imposing a time limit.)

The procedure is sound: if

$$\varphi(p) \wedge (p \Rightarrow t_1 = a) \wedge \ldots \wedge (p \Rightarrow t_k = a)$$

is unsatisfiable for an unspecified proposition p and certain terms $t_1, \ldots, t_k$, then so is

$$\varphi(\forall x.x = a) \wedge ((\forall x.x = a) \Rightarrow t_1 = a) \wedge \ldots \wedge ((\forall x.x = a) \Rightarrow t_k = a),$$

and since $(\forall x.x = a) \Rightarrow t_i = a$ is a tautology for each term t_i, the original formula $\varphi(\forall x.x = a)$ is unsatisfiable. It can also be shown [Fon07] that the procedure is complete under certain conditions, notably when quantified variables are used only as an argument of predicates, but not of functions. This is the case when the formulas are generated from quantifier-free formulas containing uninterpreted predicates and functions, equalities, and set operators.

The deduction method for quantified formulas within the SMT-solvers is built on top of the quantifier-free decision procedure. Similarly, our proof reconstruction for Skolem formulas is based on our previous work [FMM$^+$06] on proof reconstruction for quantifier-free formulas. When checking the satisfiability of a formula $\varphi(\forall x.\psi(x))$, the solver makes use of lemmas such as $(\forall x.\psi(x)) \Rightarrow \psi(t)$. Those lemmas are valid in first-order logic. For proof reconstruction, it is sufficient to record those lemmas that were used by the underlying decision procedures. As a consequence, proof reconstruction for Skolem functions consists of two steps:

- First, collect all instantiation lemmas that are required by the proof, and prove each one of them by applying the specialization rule of first-order logic.

- Second, add these lemmas to the set of facts available for proof reconstruction for the quantifier-free formula obtained by abstracting quantified formulas to Boolean propositions. This abstraction must be unsatisfiable by the proof found by the SMT solver.

For our example, the Skolemized formula (3) is given to the haRVey solver. The proof trace consists of the two instantiation lemmas

$$\big(\forall x\ .\ (X(x) \wedge Y(x)) \equiv \bot\big) \Rightarrow \big((X(s) \wedge Y(s)) \equiv \bot\big)$$
$$\big(\forall x\ .\ (X(x) \wedge \neg Z(x)) \equiv X(x)\big) \Rightarrow \big((X(s) \wedge \neg Z(s)) \equiv X(s)\big)$$

which are proved valid automatically within the `rv` tactic. The conjunction of the negation of formula (3) and those two lemmas is propositionally unsatisfiable. To prove in the proof-assistant that formula (3) is indeed valid, it is thus now sufficient to rely on the techniques for proof reconstruction of Boolean formulas. Note that, although the two instantiation lemmas appear as subformulas of the original goal (3), the instantiation procedure does not deal with the logical structure of the goal: this step is delegated to the procedure for Boolean formulas. Thus, because the procedure for Boolean formulas scales well, so does the instantiation procedure.

4.3 Procedure for Set Formulas

On top of our procedure for first-order logic, it is easy to increase expressiveness by providing extensions for languages that are reducible to first-order logic. The set-theoretical constructions defined in Section 3 are such an example.

The process of translating a set formula to a first-order formula is done by induction, both on the structure of sets (symbols $\cap, \cup$, and $\setminus$) and of set formulas (symbols $\subseteq$, $\subset$, and $\approx$). This translation introduces (first-order) quantifiers and thus would not have been realizable if the underlying procedure did not handle first-order logic. To compare performances, we implemented this translation in three pragmatically different manners: *divide and conquer*, *rewriting*, and *reflection*. In paragraphs below, we detail the last two approaches since the first one is standard.

Rewriting. In this approach we use the rewriting facility of Isabelle's simplifier to normalize set formulae using the easily proved theorems of Figure 2. Note that in Isabelle sets defined by explicitly listing their elements is expressed by successive insertion (operator :) into the empty set. Note also that set membership $x \in S$ is a shorthand for *member S x*.

$$\begin{array}{rcl}
A \subseteq B & \equiv & \forall x.x \in A \Rightarrow x \in B \\
A \subset B & \equiv & (\forall x.x \in A \Rightarrow x \in B) \wedge (\exists x.x \notin A \wedge x \in B) \\
A \approx B & \equiv & \forall x.x \in A \equiv x \in B \\
x \in A \cap B & \equiv & x \in A \wedge x \in B \\
x \in A \cup B & \equiv & x \in A \vee x \in B \\
x \in A \setminus B & \equiv & x \in A \wedge x \notin B \\
x \in (a : B) & \equiv & x = a \vee x \in B \\
x \in \Omega & \equiv & \top \\
x \in \{\} & \equiv & \bot \\
x \in \{y.Py\} & \equiv & P(x)
\end{array}$$

Figure 2: Normalizing rewrite-rules for set formulae

We can easily see that this rewriting system normalizes any set formula into a set formula where set membership is the *only* symbol not belonging to the first-order language. Moreover, *member* is only applied to set variables and set variables can only occur under the application of *member*. Therefore we just abstract over *member* S, for every set variable S occurring in the normalized problem. The result is clearly purely first-order.

Reflection. This approach is based on the following key observations:

- The syntactical structure of, *e.g.* set, expressions is not accessible inside the logic since two syntactically different expressions can be equal. Consider $S \cup \emptyset$ and S for instance.
- A logic as expressive as higher-order logic (available in our implementation in Isabelle/HOL) contains a programming language allowing the definition of data types and recursive functions using pattern matching, which are typical properties of the meta-language.

The main idea of this approach is to use the programming language inside the logic to reflect the intended transformation. Note that consequently the transformation is a function in the logic and hence we can state and prove theorems about it. This idea goes back at least to the *meta-functions* [BM81] and has been successfully used for decision procedures in Isabelle [Cha06].

We define the structure of set expressions using a data type $\mathcal{E}$, parametrized by the type α of the set elements.

$$\textsf{datatype}\ \alpha\ \mathcal{E} = \ \dot{Atom}\ (\alpha\ set) \mid \alpha\ \mathcal{E}\ \dot{\cap}\ \alpha\ \mathcal{E} \mid \alpha\ \mathcal{E}\ \dot{\cup}\ \alpha\ \mathcal{E} \mid \alpha\ \mathcal{E}\ \dot{\setminus}\ \alpha\ \mathcal{E}$$

This data type definition must be understood in connection with the semantics $(\!|.|\!)$, a recursive function in the logic which maps $\mathcal{E}$ back to set expressions:

$$\begin{array}{lcl} (\!|\ \dot{Atom}\ \ S|\!) & \equiv & S \\ (\!|e\ \dot{\cap}\ e'|\!) & \equiv & (\!|e|\!)\ \cap\ (\!|e'|\!) \\ (\!|e\ \dot{\cup}\ e'|\!) & \equiv & (\!|e|\!)\ \cup\ (\!|e'|\!) \\ (\!|e\ \dot{\setminus}\ e'|\!) & \equiv & (\!|e|\!)\ \setminus\ (\!|e'|\!) \end{array}$$

Note that the dotted symbols $\dot{\cap}$, $\dot{\cup}$ and $\dot{\setminus}$ are just constructors and reflect their counterparts in the logic $\cap, \cup$ and $\setminus$.

The problem of transforming set expressions to an appropriate element of $\mathcal{E}$, generally referred to by *reification*, can only be solved by a function in the meta language, due to the first key observation above. Our specific reflection is very simple and reification roughly consists in rewriting with the inverted defining rules of $(\!|.|\!)$. Lately, Isabelle/HOL has been enhanced with a generic mechanism for performing reification of simple functions parametrized with their equations. This mechanism also handles simple bindings. Our examples are thus dealt with automatically. Note that reification not only comes up with the solution, but also provides a *proof* that it indeed reflects the input problem.

Now that we can go back and forth from set expressions to $\mathcal{E}$, we define a recursive function $(\cdot\ \dot{\in}\ \cdot)$ for membership in $\alpha\ \mathcal{E}$ by:

$$x\ \dot{\in}\ \ \dot{Atom}\ \ S \equiv x \in S \qquad x\ \dot{\in}\ e\ \dot{\cap}\ e' \equiv x\ \dot{\in}\ e \wedge x\ \dot{\in}\ e'$$
$$x\ \dot{\in}\ e\ \dot{\cup}\ e' \equiv x\ \dot{\in}\ e \vee x\ \dot{\in}\ e' \qquad x\ \dot{\in}\ e\ \dot{\setminus}\ e' \equiv x\ \dot{\in}\ e \wedge \neg x\ \dot{\in}\ e'$$

Now it is easy to prove by induction that $\dot{\in}$ "reflects" $\in$:

$$x \in (\!|S|\!) \equiv x\ \dot{\in}\ S$$

As easy consequences we prove the following equivalences to reduce the remaining set relations to membership:

$$\begin{aligned} (\!|A|\!) \subseteq (\!|B|\!) &\equiv \forall x \,.\, x \mathrel{\dot\in} A \Rightarrow x \mathrel{\dot\in} B \\ (\!|A|\!) \subset (\!|B|\!) &\equiv (\forall x \,.\, x \mathrel{\dot\in} A \Rightarrow x \mathrel{\dot\in} B) \wedge (\exists x \,.\, x \mathrel{\dot\in} B \wedge \neg x \mathrel{\dot\in} A) \\ (\!|A|\!) = (\!|B|\!) &\equiv \forall x \,.\, x \mathrel{\dot\in} A \equiv x \mathrel{\dot\in} B \end{aligned}$$

Note that these proved equivalences together with the defining rules of $\dot\in$ form a strongly convergent rewriting system and reduce a statement involving set relations to a statement in pure first-order logic, provided all involved set expressions have the form $(\!|e_A|\!)$ for some e_A.

Given a set relation, e.g. $A \subseteq B$, this method transforms it into FOL by first replacing the set expressions it involves by their reification, i.e. A and B by $(\!|e_A|\!)$ and $(\!|e_B|\!)$ for appropriate e_A and e_B, and then just unfolds the rules above. Note that our use of reflection does *not* involve any code-generation facility (as done in other work [Cha06]) or any untrusted component. The proof is by *full* inference.

Benchmarks comparing performances of each method are visible in Figure 3. The divide and conquer approach which uses high-level Isabelle's tactics is the slowest method. The reflection method and the rewriting one are comparable in terms of performance because they both rely on Isabelle's simplifier. They outperform the divide and conquer technique since Isabelle's simplifier is implemented using low-level and effective tactics.

Finally, all stages of our procedure are summarized in Figure 4.

Formula	Size		Time (sec)		
	# set operators	average size of sets	Divide and conquer	Rewriting	Reflection
1	63	14	0.69	0.10	0.02
2	75	5	0.40	0.04	0.02
3	53	25	0.92	0.12	0.03
4	7	127	0.67	0.12	0.02
5	334	3	0.77	0.10	0.18
6	3	995	37.10	2.62	1.71
7	285	4	1.02	0.11	0.18
8	116	11	2.15	0.22	0.73

Figure 3: Running time for transformation of set formulas to first-order logic.

5 Conclusions and Further Work

We have extended a technique for the sound integration of an automatic prover and an interactive proof assistant from a quantifier-free fragment to the full language of first-order logic. This added expressiveness allows us to encode simple set-theoretic problems by a straightforward pre-processing step. Combinations of automatic and interactive provers are important because they make recent advances in automatic proof techniques and efficient implementations available within an interactive environment that contains an expressive modeling language. Soundness of the combination is ensured by proof reconstruction: theorems proved with the help of the external prover come with the same soundness guarantees as those proved with standard proof tactics of the interactive prover. A prototype implementation of our technique has been realized by integrating the automatic prover haRVey with the Isabelle/HOL proof assistant. With

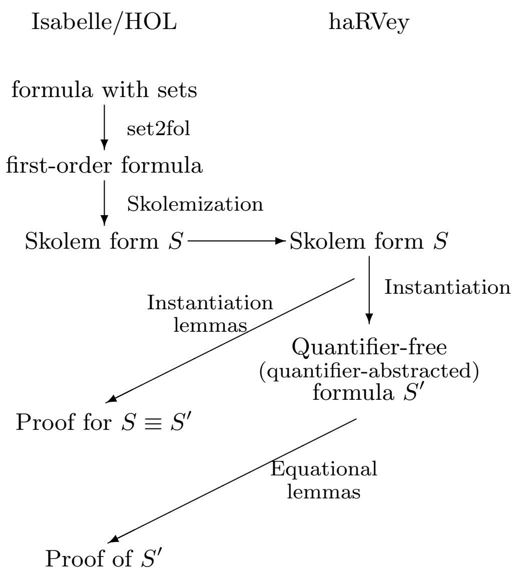

Figure 4: Entire procedure

this implementation we can automatically prove theorems that cannot be handled by Isabelle's built-in proof methods. We are not aware of other work on the integration of SMT solvers and interactive proof assistants that provide a similar level of expressiveness.

Future Work. The work presented here could be adapted to similar interactive and automatic deduction tools, and extended to other theories, such as fragments of arithmetic. Of particular interest are problems having short certificates which are computationally hard to find (*e.g.* construction of Gröbner bases for quantifier-free ring equalities [CW07]).

Before addressing such problems, it would be useful to design a generic interface dedicated to proof reconstruction that could be reused across a wide range of automatic and interactive tools. A standard proof format for SMT solvers would facilitate this task.

References

[BC04] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions.* European Association of Theoretical Computer Science. Springer-Verlag, 2004.

[BEL01] M. Baaz, U. Egly, and A. Leitsch. Normal form transformations. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 273–333. Elsevier Science B.V., 2001.

[BM81] R. S. Boyer and J. S. Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In *The Correctness Problem in Computer Science*, pages 103–84. Academic Press, New York, 1981.

[Cha06] A. Chaieb. Verifying mixed real-integer quantifier elimination. In U. Furbach and N. Shankar, editors, *International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 528–540. Springer-Verlag, 2006.

[CM03] D. Cansell and D. Méry. Foundations of the B method. In *Computing and Informatics*, volume 22, pages 1–31, 2003.

[CN06] A. Chaieb and T. Nipkow. Proof synthesis and reflection for linear arithmetic. Submitted, November 2006.

[CW07] A. Chaieb and M. Wenzel. Context aware calculation and deduction: Ring equalities via Gröbner bases in Isabelle. Submitted, March 2007.

[DdM] B. Dutertre and L. de Moura. The Yices SMT solver. Available at `http://yices.csl.sri.com/`.

[DFR] D. Déharbe, P. Fontaine, and S. Ranise. The haRVey theorem prover. Available at `http://harvey.loria.fr/`.

[FMM$^+$06] P. Fontaine, J-Y. Marion, S. Merz, L. Prensa Nieto, and A. Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181, Vienna, Austria, 2006. Springer-Verlag.

[Fon07] P. Fontaine. Combinations of theories and the Bernays-Schönfinkel-Ramsey class, 2007. Submitted.

[Lam02] L. Lamport. *The book Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Pearson Education, 2002.

[MBG06] S. McLaughlin, C. Barrett, and Y. Ge. Cooperating theorem provers: A case study combining HOL-light and CVC lite. *Electronic Notes in Theoretical Computer Science*, 144(2):43–51, 2006.

[MQP04] J. Meng, C. Quigley, and L. C. Paulson. Automation for interactive proof: First prototype. *Journal of Infomation and Computation*, 2004.

[NKK02] Q. H. Nguyen, C. Kirchner, and H. Kirchner. External rewriting for skeptical proof assistants. *Journal of Automated Reasoning*, 29(3-4):309–336, 2002.

[NO05] R. Nieuwenhuis and A. Oliveras. Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. (Invited Paper). In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming Artificial Intelligence and Reasoning*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer-Verlag, 2005.

[NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science.* Springer-Verlag, 2002.

[NW01] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.

[Pau99] L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–83, 1999.

[RT06] S. Ranise and C. Tinelli. *The SMT-LIB Standard: Version 1.2*, August 2006. Available at `http://combination.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`.

[RV02] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *Journal of AI Communications*, 15(2):91–110, 2002.

[Sch04] S. Schulz. System Description: E 0.81. In D. Basin and M. Rusinowitch, editors, *International Joint Conference on Automated Reasoning*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 223–228, Cork, Ireland, 2004. Springer-Verlag.

[WBH$^+$02] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topić. SPASS version 2.0. In Andrei Voronkov, editor, *Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*, pages 275–279, Kopenhagen, Denmark, 2002. Springer-Verlag.

[Web05] T. Weber. Integrating a SAT solver with an LCF-style theorem prover. In A. Armando and A. Cimatti, editors, *Pragmatics of Decision Procedures in Automated Reasoning*, Edinburgh, UK, July 2005.