Incorporating Decision Procedures in Implicit Induction

Alessandro Armando¹, Michaël Rusinowitch², and Sorin Stratulat¹

¹ DIST – Università di Genova, Viale Causa 13 – 16145 Genova – Italia
² LORIA-INRIA, 615, rue du Jardin Botanique – 54602 Villers les Nancy – France

Abstract. In the last decades the automation of reasoning by mathematical induction has been thoroughly investigated and several powerful techniques and heuristics have been put forward. However, when applied to proof obligations arising in practical applications, the level of automation achieved by existing induction provers is still unsatisfactory. As shown by Boyer and Moore, a higher level of automation can be achieved by the incorporation of decision procedures into induction provers. Yet in Boyer and Moore’s approach the role of the decision procedure is confined to the simplification engine and this limits the possible usage of the decision procedure by the prover. In this paper we present an extension to Boyer and Moore’s integration schema that enables the decision procedure to use suitably selected instances of the induction hypotheses. The induction proof method we consider is based on and combines Cover Set Induction and Constraint Contextual Rewriting and has been implemented in the SPIKE prover. Computer experiments on non-trivial verification problems give evidence of the effectiveness of our approach: the proof of the MJRTY algorithm does not need anymore user-defined tactics as it is the case with STel’ and Nuprl; moreover, in the proof of an ABR conformance algorithm, many of the about 80 user-defined lemmas require specific tactics with PVS whereas more than half of them are relieved automatically by our extended system.

1 Introduction

In the last decades the automation of reasoning by mathematical induction has been thoroughly investigated and several powerful techniques and heuristics have been put forward. However, when applied to proof obligations arising in practical applications, the level of automation achieved by existing induction provers is still unsatisfactory. As shown by Boyer and Moore [BMS8], a higher level of automation can be achieved by the incorporation of decision procedures into induction provers. Yet in Boyer and Moore’s approach the role of the decision procedure is confined to the simplification engine and this limits the possible usage of the decision procedure by the prover.

In this paper we present an extension to Boyer and Moore’s integration schema that enables the decision procedure to use suitably selected instances of the induction hypotheses. The approach we propose is based on and combines Cover Set Induction and Constraint Contextual Rewriting. Cover Set Induction [BR95b] is a powerful automated reasoning technique for reasoning about inductively defined objects which combines the advantages of explicit induction and proof by consistency. Constraint Contextual Rewriting [AR98,AR00], CCR or CCR(X)¹ for short, is an abstract integration schema between rewriting and decision procedures. CCR(X) generalizes contextual rewriting [ZR85,Zha95] by allowing the available decision procedure to access and manipulate the rewriting context. One of the key features of CCR is the ability to augment the state of the decision procedure with facts encoding properties of symbols which are uninterpreted for the decision procedure. (As shown in [BMS8], this feature is crucial to the effectiveness of the integration.) A key contribution of this paper is the extension of CCR so to enable the use of the induction hypotheses (other than the available definitions and lemmas) during the augmentation of the state of the decision procedure.

The extended induction proof method presented in this paper has been implemented in the SPIKE prover [BR95a]. Computer experiments on non-trivial verification problems give evidence

¹ The notation CCR(X) (by analogy with the CLP(X) notation used to denote the Constraint Logic Programming paradigm [JL87]) is used to stress the independence of CCR(X) from the theory decided by the decision procedure
of the effectiveness of our approach: the proof of the MJRTY algorithm [BM91] does not need anymore user-defined tactics or lengthy interactions as it is the case Nuprl [Jac94] and STeP [B+95]; moreover, in the proof of an ABR confluence algorithm, many of the about 80 user-defined lemmas require specific tactics with PVS [ORS92], whereas more than half of them are relieved automatically by our extended system.

Structure of the paper. In Section 2 we introduce the concept of reasoning specialist and specify the associated interface functionalities. CCR is defined in Section 3 and clause simplification is defined in Section 4. Our extended induction method is then presented in Section 5. A reasoning specialist for the union of the quantifier-free theory of equality and quantifier-free Presburger arithmetic is described in Section 6. An excerpt of the proof of the soundness of MJRTY is finally discussed in Section 7.

Preliminaries. By $\Sigma$ (possibly subscripted) we denote finite sets of function symbols (with their arity). $V$ (possibly subscripted) denotes a finite set of variables. $\tau(\Sigma, V)$ is the set of terms built out of $\Sigma$ and $V$ in the usual way. $\tau(\Sigma)$ abbreviates $\tau(\Sigma, \emptyset)$, i.e. the set of ground terms. We assume the usual conceptual machinery (e.g. the notion of substitution, the definition of position of a sub-expression) as given, e.g., in [DJ90]. A $\Sigma$-equation is an expression of the form $t_1 = t_2$ where $t_1, t_2 \in \tau(\Sigma, V)$. $(\Sigma, V)$-formulae are built in the usual way using the standard logical connectives (i.e., $\land, \lor, \rightarrow, \iff$). A $(\Sigma, V)$-literal is either a $(\Sigma, V)$-equation or a negated $(\Sigma, V)$-equation. We write $\Sigma$-equation (literal) instead of $(\Sigma, \emptyset)$-atom (literal, resp.). A $(\Sigma, V)$-clause is a disjunction of literals which we indicate as finite set of $(\Sigma, V)$-literals. We denote by true any tautology of the form $t = t$, for any $t \in \tau(\Sigma, V)$. If $a$ is an atom, then $\overline{a}$ abbreviates $\neg a$ and $\overline{\overline{a}}$ stands for $a$; similarly, if $E$ is a set of literals, then $\overline{E}$ abbreviates $\{ p : p \in E \}$. $(p_1, \ldots, p_n \Rightarrow p)$ abbreviates the clause $\{ \overline{p_1}, \ldots, \overline{p_n}, p \}$. $\land S$ stands for any conjunction of the literals in $S$.

$\kappa$ is a reduction ordering over the $(\Sigma_j, V)$-expressions (i.e. a well-founded relation over the $(\Sigma_j, V)$-expressions closed under substitution and replacement) containing the sub-expression relation. We also assume that true $\not\prec e$ for all equations $e \not\equiv$ true. Given a congruence relation $\approx$ on terms that is stable (i.e., $s \approx t$ or $s \approx t$) and compatible with $\prec$ (i.e., $s' \prec t'$ if $s \prec t$, $s \approx s'$, and $t \approx t'$) we define $\leq$ as $s \cup \approx \lessdot \prec$ is the multiset extension of $\prec$. A conditional equation $\Lambda^m_{i=1} a_i = b_i \Rightarrow l = r$ can be oriented into the conditional rewrite rule $\Lambda^m_{i=1} a_i = b_i \Rightarrow l \rightarrow r$ if, for each substitution $\sigma$, $\{ \sigma \} \cup (\cup_{i=1}^m \{ a_i \sigma, b_i \sigma \} \lessdot \{ l \sigma \}$. A conditional rewrite system is a set of conditional rewrite rules. Given a conditional rewrite system $R$ obtained from the orientation of a set of axioms $Ax$, we define by $\rightarrow_R$ the conditional rewriting relation defined by $t[\sigma]_u \rightarrow_R t[\sigma]_u$ iff $\sigma$ is a substitution and $\Lambda^m_{i=1} a_i = b_i \Rightarrow l \rightarrow r$ a conditional rewrite rule in $R$ such that for any $i = 1, \ldots, m$, there exists a term $c_i$ with $a_i \sigma \rightarrow_R c_i$ and $b_i \sigma \rightarrow_R c$. A term $t$ is $R$-irreducible if there is no term $s$ such that $s \rightarrow_R t \Sigma$ and $t$ is $R$-irreducible, we say that $t$ is the $R$-normal form of $s$.

The induction principle used by SPIKE is based on an order over clauses, closed under substitution and well-founded, built from a multiset extension of $\prec$ [BR95b,Na96]. Let $\leq_{C^\Sigma}$ be such an order and let $\prec \in \{ <, \leq_{C^\Sigma} \}$. If $C$ is a clause, then $C_{<\Sigma}$ is the set $\{ C \sigma | C \sigma <_{\Sigma} \}$. If $E$ is a set of clauses then $E_{<\Sigma}$ is the set of annotated clauses $\{ C_{<\Sigma} | C \in E \}$.

Let $Ax$ be a set of axioms, then we say that $\phi$ is an initial consequence of $Ax$, in symbols $Ax \models_{p_1} \phi$, iff $\phi$ is valid in the initial model of $Ax$. In the sequel $T_j$ is the $(\Sigma_j, V)$-theory comprising all the initial consequences of $Ax$, and $T_c$ is a decidable fragment of $T_j$ (i.e. $T_c \subseteq T_j$). We say that $\phi$ is $T_c$-unsatisfiable if $\phi$ is not valid in the initial model of $Ax$ and that $\phi$ is $T_c$-unsatisfiable if no model of $T_c$ is also a model of $\phi$. Similarly, we say that $\phi$ is $T_j$-valid if $\phi$ is an initial consequence of $Ax$ and that $\phi$ is $T_c$-valid iff all models of $T_c$ are also models of $\phi$.

2 The Reasoning Specialist

A reasoning specialist is a state-based procedure whose states (called constraint stores) are finite sets of $\Sigma_c$-literals represented in some internal form and whose functionalities are abstractly characterized in the following way.
**Initialization of the Constraint Store.** The first functionality we consider is the relation \( cs-init(S) \) which characterizes the "empty" constraint stores. \( cs-init(S) \) is required to be a decidable relation such that \( cs-init(S) \) holds only if \( S \) is \( T_c \)-valid.

**Detection of Unsatisfiability.** \( cs-unsat(S) \) characterizes a set of \( T_c \)-unsatisfiable constraint stores \( S \) whose \( T_c \)-unsatisfiability can be checked by means of a computationally inexpensive syntactic check. We require that \( cs-unsat(S) \) is decidable and that \( cs-unsat(S) \) implies the \( T_c \)-unsatisfiability of \( S \).

**Constraint Store Simplification.** The main functionality of the reasoning specialist is a transition relation over constraint stores, \( S \xrightarrow{cs\text{-simp} \ P} S' \), which models the activity of adding a finite set of \( \Sigma_c \)-literals \( P \) to \( S \) yielding a new constraint store \( S' \). For soundness we require that \( T_c \models \bigwedge (P \cup S) \Rightarrow \bigwedge S' \) whenever \( S \xrightarrow{cs\text{-simp} \ P} S' \).

**Augmentation.** Let \( P \) be a finite set of literals. It is a trivial consequence of the above definitions the fact that if \( S_0 \) and \( S \) are constraint stores such that \( cs-init(S_0) \), \( S_0 \xrightarrow{cs\text{-extend} \ P} S \), and \( cs-unsat(S) \) then \( P \) is \( T_c \)-unsatisfiable. This observation shows that the functionalities we have presented so far allow us to check the \( T_c \)-unsatisfiability of any given set of literals \( P \). Unfortunately in most cases \( P \) (and hence \( S \)) is \( T_j \)-unsatisfiable but not \( T_c \)-unsatisfiable. When this is the case, the \( T_j \)-unsatisfiability of \( S \) cannot possibly be detected by the reasoning specialist. The occurrence in \( S \) of (function) symbols interpreted in \( T_j \) but not in \( T_c \) is the main cause of the problem. The key idea of augmentation is to extend \( S \) with \( T_j \)-valid facts, thereby informing the reasoning specialist about properties of function symbols it is otherwise not aware of. By adding \( T_j \)-valid facts to the rewriting context, the heuristics aims at generating a \( T_j \)-equivalent but \( T_c \)-unsatisfiable context whose \( T_j \)-unsatisfiability can therefore be detected by the reasoning specialist. The selection of suitable \( T_j \)-valid facts is done by looking up \( R \) or \( \mathcal{H} \) which contains the available induction hypotheses. To model augmentation we define a new relation, \( S \xrightarrow{cs\text{-extend} \ R, H, P} S' \) as the smallest transitional relation (i.e. such that \( S \xrightarrow{cs\text{-extend} \ R, H, P} S' \) and \( S' \xrightarrow{cs\text{-extend} \ R, H, P} S'' \) imply \( S \xrightarrow{cs\text{-extend} \ R, H, P} S' \) for all \( R, H, \) and \( P \)) such that:

\[
\text{CS-SIMP: } S \xrightarrow{cs\text{-simp} \ P} S' \text{ if } S \xrightarrow{cs\text{-simp} \ P} S'
\]

\[
\text{AUGMENT: } S \xrightarrow{cs\text{-extend} \ R, H, P} S' \text{ if } (Q \Rightarrow c) \in R \text{ or } (Q \Rightarrow c) \in H, c \text{ is a } \Sigma_c \text{-literal, } q \sigma \xrightarrow{cor \ R, H, S} \text{ true for all } q \sigma \in Q, \text{ and } S \xrightarrow{cs\text{-extend} \ R, H, \{c\sigma\}} S'
\]

where \( cor \) is the constraint contextual rewriting relation we define next.

### 3 Constraint Contextual Rewriting

Constraint Contextual Rewriting is modeled by the relation \( E \xrightarrow{cor \ R, H, S} E' \) which is defined to be the smallest transitional relation such that

**Entailment Check:** \( e \xrightarrow{cor \ R, H, S} \text{ true} \) if \( e \) is a \( \Sigma_c \)-literal, \( e \neq \text{ true} \), \( S \xrightarrow{cs\text{-extend} \ R, H, \{e\}} S' \) and \( cs-unsat(S') \)

**Conditional Rewriting:** \( e[\sigma] \xrightarrow{cor \ R, H, S} e[r\sigma] \)

\[
\text{if } (Q \Rightarrow l = r) \in R \text{ or } (Q \Rightarrow l \Rightarrow r) \sigma \in H, \text{ and } q \sigma \xrightarrow{cor \ R, H, S} \text{ true for all } q \in Q.
\]

**Theorem 1 (Soundness of CCR).**

1. If \( S \xrightarrow{cs\text{-extend} \ R, H, P} S' \) then \( R, H, P, S \models_{init} \bigwedge S' \);
2. If \( e \xrightarrow{cor \ R, H, S} e' \) then \( R, H, S \models_{init} (e \sim e') \) and \( e' < e \);
Proof. Since the definitions of \( \text{cs-extend} \) and \( \text{corr} \) are mutually dependent, we prove facts 1 and 2 by mutual induction. Let us consider a calculus comprising the rules cs-simp, augment, entailment check and CCR, and let us consider any sensible definition of derivation in such a combined/hybrid calculus. We reason by induction on the depth of the derivations. The base case (i.e. the number of occurrences of \( \text{cs-extend} \) and \( \text{corr} \) in the derivation is 1) amounts to proving the following case:

- \( S \xrightarrow{\text{cs-extend}} R \vdash \{ S' \} \) results from the application of cs-simp and therefore \( S' \) is such that \( S' \xrightarrow{\text{cs-simp}} R \vdash \{ S' \} \) and hence \( T_c \models \bigwedge (P \cup S) \Rightarrow \bigwedge S' \). From this 1 readily follows.

In the step case we must prove that 1 and 2 hold for all derivations of depth \( k + 1 \) provided that they hold for all derivations of depth \( k \). In the step case we have the following cases to consider:

- \( S \xrightarrow{\text{cs-extend}} R \vdash \{ S' \} \) results from the application of augment and therefore \( S' \) is such that \( S \xrightarrow{\text{cs-extend}} R \vdash \{ s \} \) where either \( (Q \Rightarrow c) \in R \) or \( (Q \Rightarrow c) \in \mathcal{H} \) and \( q \sigma \xrightarrow{\text{corr}} \text{true for all } q \sigma \in Q \). From the induction hypothesis we know that \( R, S \models_{\mu_i} q \sigma \) for all \( q \sigma \in Q \). From this and the fact that \( (Q \Rightarrow c) \in R \) or \( (Q \Rightarrow c) \in \mathcal{H} \) it readily follows that \( R, S \models_{\mu_i} q \sigma \). By induction hypothesis we also know that \( R, S \models_{\mu_i} \bigwedge S' \) and therefore we can conclude that \( R, S \models_{\mu_i} \bigwedge S' \) and hence 1.

- \( e \xrightarrow{\text{corr}} S \) results from the application of entailment check and hence \( e \) is a \( \Sigma_i \)-literal and \( e' \) is true. In this case we know that \( S \xrightarrow{\text{cs-extend}} R \vdash \{ S' \} \) and \( \text{cs-unsat}(S') \). By induction hypothesis we have \( R, S \models_{\mu_i} \bigwedge S' \) and from the \( T_c \)-unsatisfiability of \( S' \) it readily follows that \( R, S \models_{\mu_i} e \). It is immediate to see that true \( \equiv \) holds.

- \( e \xrightarrow{\text{corr}} S \) results from the application of conditional rewriting and there exists a substitution \( \sigma \) and a clause \( Q \Rightarrow l = r \) such that \( l \sigma \) occurs in \( e \), i.e. \( e = e[l \sigma] \), \( e' = e[r \sigma] \), \( Q \leftarrow \{ l \sigma = r \sigma \} \), \( r \sigma \ll l \sigma \), and either \( (Q \Rightarrow l = r) \in R \) or \( (Q \Rightarrow l \Rightarrow r) \sigma \in \mathcal{H} \); moreover \( q \sigma \xrightarrow{\text{cs-extend}} \text{true for all } q \sigma \in Q \). By induction hypothesis we know that \( R, S \models_{\mu_i} q \sigma \) holds for all \( q \sigma \in Q \). If \( (Q \Rightarrow l = r) \in R \) we can conclude that \( R, S \models_{\mu_i} l \sigma = r \sigma \), and therefore \( R, S \models_{\mu_i} e[l \sigma] \ll e[r \sigma] \). Also, \( e[r \sigma] \ll e[l \sigma] \) holds since \( r \sigma \ll l \sigma \) and \( \ll \) is monotonic and stable. The case in which \( (Q \Rightarrow l \Rightarrow r) \sigma \in \mathcal{H} \) is proven along the same lines.

4 Clause Simplification

Clause simplification is modeled by the relation \( E \xrightarrow{\text{cs-simp}} E' \) which is defined to be the smallest transitive relation such that:

\[
\begin{align*}
\text{DELETE: } & \{ C \} \cup E \xrightarrow{\text{cs-simp}} E \text{ if true } \in C \\
\text{CCR: } & \{ \{ p \} \cup C \} \cup E \xrightarrow{\text{cs-simp}} \{ \{ p' \} \cup C \} \cup E \text{ if } \text{cs-init}(S_0), \quad \{ C \} \cup E \xrightarrow{\text{cs-extend}} S \text{ and } p \xrightarrow{\text{corr}} S' \Rightarrow p'
\end{align*}
\]

Theorem 2 (Soundness of Clause Simplification). If \( E \xrightarrow{\text{cs-simp}} E' \) and \( \ll_e \) is monotonic, then for all clauses \( C \in E \) either \( \text{true } \in C \) or there exists a clause \( C' \in E' \) such that \( R, S \models_{\mu_i} (C \Rightarrow C') \) and \( C' \ll_e C \).

The proof of this result is straightforward and therefore it is omitted.

5 Cover Set Induction

Let \( R \) be a rewrite system derived from the orientation of a set of axioms \( Ax \). A term \( t \) is said to be inductively \( R \)-reducible (resp. \( R \)-irreducible) if, for each substitution \( \gamma \) mapping variables to \( R \)-irreducible terms, \( t \gamma \) is \( R \)-reducible (resp. \( R \)-irreducible). A cover set for a conditional rewrite system \( R, CS(R) \), is a finite set of \( R \)-irreducible terms such that for all ground \( R \)-irreducible term...
s, there is a term t in CS(R) and a ground substitution σ such that Ax ⊨ tσ = s. From a cover set for a conditional rewrite system, we can build cover sets for clauses. A cover substitution for a clause C instantiates a particular subset of Var(C) (called induction variables) by terms obtained from CS(R) whose variables are replaced by fresh ones. We will denote by CSΣ(C) the set of all possible cover substitutions for the clause C. Then, the set \{Cσ | σ ∈ CSΣ(C)\} is a cover set for the clause C.

The induction method we consider incrementally modifies two sets of clauses, \( (E, H) \), where \( E \) contains the conjectures to be checked and \( H \) contains clauses, previously in \( E \), that have been reduced. The method is modeled by means of the relation \( (E, H) \xrightarrow{\text{spike}_s \text{Ax}} (E', H') \) which is defined to be the smallest transitive relation such that:

**Generate:** \( (E \cup \{C\}, H) \xrightarrow{\text{spike}_s \text{Ax}} (E \cup \bigcup_{\sigma \in \text{CSΣ}(C)} E_{\sigma}, H \cup \{C\}) \)

if \( \{C_{\sigma} \} \xrightarrow{R \subseteq C \cup E_{\sigma} \subseteq C_{\sigma}} E_{\sigma} \) for \( \sigma \in \text{CSΣ}(C) \)

**Simplify:** \( (E \cup \{C\}, H) \xrightarrow{\text{spike}_s \text{Ax}} (E \cup E', H) \)

if \( \{C \} \xrightarrow{R \subseteq C \cup E_{\sigma} \subseteq C_{\sigma}} E' \)

The above two rules synthesize a simplified version of the current inference system of SPIKE. The Generate inference rule computes the covering substitutions which are then applied to conjectures thereby generating special instances which are then simplified by rules, lemmas and the available induction hypotheses. The Simplify inference rule simplifies conjectures. The set of induction hypotheses are ad-hoc instances of the current set of \( E \), \( \{C\} \) and \( H \) depending on the treated clause \( C \).

\( E_0 \) is an inductive theorem w.r.t. \( Ax \) if there exists a finite derivation of the from \( (E_0, \emptyset) \xrightarrow{\text{spike}_s \text{Ax}} \cdots \xrightarrow{\text{spike}_s \text{Ax}} (\emptyset, H) \). More in general, we say that \( E_0 \) is an inductive theorem w.r.t. \( Ax \), in symbols \( Ax \vdash_{\text{ind}} E_0 \), if there exists a fair derivation \( (E_0, \emptyset) \xrightarrow{\text{spike}_s \text{Ax}} (E_1, H_1) \xrightarrow{\text{spike}_s \text{Ax}} \cdots \), i.e. iff the set of persisting clauses \( \bigcup_{i > 0} \bigcap_{j > i} E_j \) is empty.

**Theorem 3 (Soundness of Cover Set Induction).** If \( Ax \vdash_{\text{ind}} E_0 \) then \( Ax \vdash_{\text{ind}} E_0 \).

**Proof (Adapted from [Bou97]).** Let us assume that \( Ax \vdash_{\text{ind}} E_0 \) but \( Ax \not\vdash_{\text{ind}} E_0 \). Since \( Ax \vdash_{\text{ind}} E_0 \), then there exists a fair derivation \( (E_0, \emptyset) \xrightarrow{\text{spike}_s \text{Ax}} (E_1, H_1) \xrightarrow{\text{spike}_s \text{Ax}} \cdots \). Let \( C \in \bigcup_i E_i \) be one of the last clauses in the derivation containing a minimal counterexample (w.r.t. \( \prec \)) from the set

\[
CE = \{ D\theta : D \in \bigcup_i E_i \text{ and } Ax \not\vdash_{\text{ind}} D\theta \text{ for some ground } R\text{-irreducible substitution } \theta \}.
\]

(Notice that \( CE \) is not empty since, by assumption, \( Ax \not\vdash_{\text{ind}} E_0 \); moreover \( CE \) has a minimal element w.r.t. \( \prec \) since \( \prec \) is well-founded.) Since the derivation is fair, there must exist a pair \( (E \cup \{C\}, H) \) in the derivation such that either Generate or Simplify apply to it. It suffices to show that neither Generate nor Simplify may affect \( C \), since this trivially contradicts the fairness assumption.

1. Case: Generate. As \( \phi \) is ground and \( R\)-irreducible, then there exists \( \sigma \in \text{CSΣ}(C) \) and a ground substitution \( \tau \) such that \( \phi = \sigma \tau \). From Theorem 2 it follows that there exists a clause \( C' \in \bigcup_i E_i \) such that \( R, (E \cup \{C\} \prec C' \prec C\phi \prec C'\tau \prec C\phi \). Notice that \( Ax \vdash_{\text{ind}} (E \cup \{C\} \prec C\phi \prec C'\tau \prec C\phi \) and \( C'\tau \prec C\phi \) which, together with \( C'\tau \prec C\phi \), contradicts the minimality of \( C'\phi \) in \( CE \).

2. Case: Simplify. From Theorem 2 it follows that there exists a clause \( C' \in \bigcup_i E_i \) such that \( R, E \prec C\phi \cup H \prec C\phi \vdash_{\text{ind}} (C'\phi \prec C'\phi \prec C\phi) \). This case is similar to the previous one with the additional proof obligation of showing that if a clause \( C_1 \) from \( H \) is such that \( C_1\theta \prec \prec C\phi \) for some ground \( R\)-irreducible substitution \( \theta \), then \( Ax \vdash_{\text{ind}} C_1\theta \). Let us assume that \( Ax \not\vdash_{\text{ind}} C_1\theta \); then \( C_1\theta \) must also be minimal in \( CE \). But \( C_1 \) can be put in \( H \) only by a previous application of Generate which is in contradiction with the previous case.

---

\(^2\) Notice that true cannot possibly be in \( C\phi \) as this would contradict the assumption that \( C\phi \in CE \).
6 A Reasoning Specialist for the union of the quantifier-free theory of Equality and quantifier-free Presburger Arithmetics

We present a reasoning specialist for the union of quantifier-free Presburger Arithmetic, $T_{QF}$, and the quantifier-free theory of equality, $T_{eq}$, obtained by combining a decision procedures for $T_{QF}$ and one for $T_{eq}$. The interface functionalities of the compound decision procedure are as described in Section 2 and we show that they comply with the requirements stated in the same section.

The decision procedure manipulates constraint stores of the form $(A \cup U \mid G \mid (P \bullet I))$ where $A$ is a set of literals, $U$ is a set of ground rewrite rules $^3$, $G$ is a set of ground equations and disequations, $L = (P \bullet I)$, where $P$ is a set of linear inequalities and $I$ is a set of equations entailed by $P$. $cs$-init $(A \cup U \mid G \mid (P \bullet I))$ holds if all the fields are empty. $cs$-unsat $(A \cup U \mid G \mid (P \bullet I))$ holds whenever there exists either an impossible inequation in $P$ or a disequation of the form $a \neq a$ in $G$. $(A \cup U \mid G \mid (P \bullet I)) \xrightarrow{cs\text{-}simp} (A' \mid U' \mid G' \mid (P' \bullet I))$ is defined by $(A \cup L \mid U \mid G \mid (P \bullet I)) \xrightarrow{data\text{-}flow} (A' \mid U' \mid G' \mid (P' \bullet I))$. $data\text{-}flow$ is the smallest transitive relation such that:

$$A2L: (A \cup U \mid G \mid (P \bullet I)) \xrightarrow{cs\text{-}simp} (A' \mid U \mid G \mid (P' \bullet I))$$

if $A' = \{a \in A : linearize(a) = \emptyset\}$ and

$$(I', P') = arith(\cup_{a \in A} linearize(a) \cup P)$$

$$A2G: (A \cup U \mid G \mid (P \bullet I)) \xrightarrow{cs\text{-}simp} (A \setminus A' \mid U \mid G \cup A' \mid (P \bullet I))$$

if $A' = \{a \in A : linearize(a) = \emptyset\}$

$$G2U: (A \cup U \mid G \mid (P \bullet I)) \xrightarrow{cs\text{-}simp} (A \cup U \cup E \mid congr(G) \mid (P \bullet I))$$

if for all $a \rightarrow b \in E$ we have that $a = b \in congr(G)$ and $b < a$

$$L2G: (A \cup U \mid G \mid (P \bullet I)) \xrightarrow{cs\text{-}simp} (A \cup U \mid G \cup I \mid (P \bullet \emptyset))$$

The effect of the application of the rules is that of moving information from the fields of the constraint store as depicted in Figure 1.

A2L initializes the component $P$ with linear inequalities resulting from the linearization of literals from $A$, then it applies $arith$ on the current set of linear inequalities from $P$. $linearize$ is a function that maps literals into sets of linear inequalities, according to the mapping specified in Table 1.$^4$ $linearize$ returns the empty set if no transformation can be performed on the input literal (and in such a case the input literal is said to be non-linearizable). $arith$ models the functionality of

<table>
<thead>
<tr>
<th>literal</th>
<th>$s \geq t$</th>
<th>$s &gt; t$</th>
<th>$s &lt; t$</th>
<th>$s = t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear inequalities</td>
<td>${t \leq s}$</td>
<td>${t + 1 \leq s}$</td>
<td>${s + 1 \leq t}$</td>
<td>${s \leq t, t \leq s}$</td>
</tr>
</tbody>
</table>

$^3$ From now on, by ground rewrite rule (resp. equation) we mean a rewrite rule (resp. equation) whose variables are forbidden to be instantiated.

$^4$ In the last column, the members of the equality $s = t$ must be terms of arithmetic sort. For brevity we omit the treatment of disequations.
models the application of an algorithm for ground completion [HL78]. These equations are oriented in unconditional rewrite rules using the ordering $\prec$. The rewrite system is used to normalize the left and right hand sides of the disequations. The L2G rule transfers the implicit inequalities from $L$ to $G$. normalize normalizes the monomials in $P$ and the ground equations in $I$ by means of the rewrite rules in $U$.

**Theorem 4 (Soundness of the Reasoning Specialist).** Let $\text{wffs}(S)$ be the set of literals stored in the constraint store $S$, then the following facts hold:

- if $\text{cs-unsat}(S)$ then $\text{wffs}(S)$ is $(T_{la} \cup T_{eq})$-unsatisfiable.
- if $S \xrightarrow{a \text{-imp}} A S'$ then $T_{la} \cup T_{eq} \models \text{wffs}(S') \Leftrightarrow \text{wffs}(S)$.

A proof of this result can be found in [Str00b].

7 Proving the Soundness of MJRTY

Given a multiset of elements as input, the MJRTY algorithm computes in an efficient way its majority element (if any), i.e. the element occurring more than the half of the multiset cardinality. The algorithm scans the elements in real time, without additional storage of elements for further operations, and eliminates the counting phase specific to other similar (trivial) algorithms. MJRTY has been devised in 1980 by Boyer and Moore who have also proved its soundness by means of their prover NQTHM [BM79]. Coded in Fortran, the algorithm has a rather difficult soundness proof that demands the use of five lemmas to check the 61 verification conditions issued by a Fortran verification condition generator. Besides NQTHM, several interactive theorem provers also succeeded to prove it, for example PVS and Nuprl [How93] and STeP [Bjo98].

The idea of the algorithm is to pair off the values and to erase pairs of different values such that the returned value at the end of the erasing process is the potential majority value. MJRTY can be easily converted from an imperative program to a recursive function $m(p,i)$ that returns a pair $(mcv, mlv)$, where $mcv$ is the majority candidate at a certain moment and $mlv$ is its lead over the other candidates knowing that the $i$ votes are stored in a poll $p$ given as input (see Algorithm 1).

The SPIKE specification of MJRTY is in Figure 2. It consists of four main parts. The first part is devoted to the specification of the sorts: nat for the naturals, bool for the booleans, cand for the candidates, and list for the lists of candidates. The prover is instructed to apply the
Algorithm 1 \( m(p, i) \): the MJRTY algorithm

Require: a poll \( p \) of \( i \) votes
Ensure: the majority candidate and its lead over the other candidates
1: if \( i > 0 \) then
2: \((\text{mcv}, \text{mlv}) \leftarrow m(p, i - 1)\)
3: if \( p[i] = \text{mcv} \) then
4: return \((\text{mcv}, \text{mlv} + 1)\)
5: else if \( \text{mlv} > 0 \) then
6: return \((\text{mcv}, \text{mlv} - 1)\)
7: else
8: return \((p[i], 1)\)
9: end if
10: else
11: \((\text{No}, 1)\)
12: end if

cooperation schema by the instruction use: \texttt{nat}$. The second part of the specification contains the declaration of function symbols. First we declare the constructor symbols 0 and \( s \) for the naturals, next True and False for the booleans, Nil and Cons for the lists of candidates and, finally, \( Cd \) and No for the candidates. No is a pseudo candidate returned by the algorithm to indicate that there is no majority candidate already computed. Then, we declare the remaining function symbols. The function \( m \) is divided in two mutually recursive functions, \( m_{c} \) and \( ml \), which compute respectively the majority candidate and its lead over the other candidates. \( count(p, i, c) \) counts the number of votes for a given candidate \( c \) from a poll \( p \) containing \( i \) votes. The other defined functions are: \( access(p, n) \) which returns the \( n \)-th element of the list \( p \) and the 4-argument conditional function \( if \). The third part of the specification consists of the axiomatic definitions for the defined function symbols. The well-founded ordering over the terms \( \prec \) is a recursive path-ordering [Der82] built on the precedence over the function symbols presented in the last part.

The main conjecture states that \( mc(p, i) \) always returns the majority candidate whenever such a candidate exists in the poll \( p \) containing \( i \) votes:

\[
\forall p : \text{list}.\forall i : \text{nat}.\forall a : \text{cand}. (i < 2 * count(p, i, c) \Rightarrow c = mc(p, i))
\]

(1)

An important lemma, provided by N. Shankar (according to [How93]), simplifies in a major way the proof of (1):

\[
2 * (if(c_1, mc(p_1, i_1), 0, ml(p_1, i_1)) + count(p_1, i_1, c_1)) < s(i_1 + ml(p_1, i_1))
\]

(2)

A detailed account of the proof of the lemma by SPIKE can be found in [Str98,Str00a]. Here we focus on steps in which the the reasoning specialist plays a key role.

SPIKE starts by applying a case analysis on the literal \( \text{if}(c_1, mc(p_1, i_1), 0, ml(p_1, i_1)) \). According to the definition of \( if \), we consider the two cases, namely \( c_1 \neq mc(p_1, i_1) \) and \( c_1 = mc(p_1, i_1) \). After rewriting \( \text{if}(c_1, mc(p_1, i_1), 0, ml(p_1, i_1)) \) with the corresponding \( if \)-axiom, we get the new conjectures:

\[
c_1 \neq mc(p_1, i_1) \Rightarrow 2 * (ml(p_1, i_1) + count(p_1, i_1, c_1)) < s(i_1 + ml(p_1, i_1))
\]

\[
c_1 = mc(p_1, i_1) \Rightarrow 2 * (0 + count(p_1, i_1, c_1)) < s(i_1 + ml(p_1, i_1))
\]

(3)

Here we focus on the proof of (3). Application of \texttt{GENERATE} to (3) moves it to the set of induction hypotheses and leads to the following sub-goal:

\[
No \neq mc(p_1, i_2), access(p_1, i_2) \neq mc(p_1, i_2), 0 < ml(p_1, i_2), No = access(p_1, i_2) \Rightarrow \text{No} = access(p_1, i_2) \Rightarrow 2 * ((ml(p_1, i_2) - 1) + s(count(p_1, i_2, No))) < s(s(i_2) + (ml(p_1, i_2) - 1))
\]

(4)
The application of Constraint Contextual Rewriting invokes the reasoning specialist (via Entailment Check) whose constraint store gets initialized to:

\[
\begin{align*}
\langle A : \{ & No \neq mc(p_1, i_2), access(p_1, i_2) \neq mc(p_1, i_2), 0 < ml(p_1, i_2), \\
& No = access(p_1, i_2), \\
& 2 \ast ((ml(p_1, i_2) - 1) + s(count(p_1, i_2, No))) \geq s(i_2) + (ml(p_1, i_2) - 1)) \} \rangle
\end{align*}
\]

The application of \texttt{A2G} moves \( No \neq mc(p_1, i_2), access(p_1, i_2) \neq mc(p_1, i_2), No = access(p_1, i_2) \) to the \( G \)-field and application of \texttt{A2L} linearizes \( 0 < ml(p_1, i_2), 2\ast((ml(p_1, i_2) - 1) + s(count(p_1, i_2, No))) \geq s(i_2) + (ml(p_1, i_2) - 1)) \) and then adds the results to the \( P \)-field. This results in the constraint store:

\[
\begin{align*}
\langle G : \{ & No \neq mc(p_1, i_2), access(p_1, i_2) \neq mc(p_1, i_2), No = access(p_1, i_2) \} \mid \\
P : (\{ & 1 + -1 \ast ml(p_1, i_2) \leq 0, -1 \ast ml(p_1, i_2) \leq 0, -1 \ast count(p_1, i_2, No) \leq 0, \\
& -1 \ast i_2 \leq 0, 1 + -2 \ast count(p_1, i_2, No) + -1 \ast ml(p_1, i_2) + 1 \ast i_2 \leq 0 \} \bullet \emptyset ) \rangle
\end{align*}
\]

Since contradiction has not been derived, the \texttt{augment} rule is then applied using the induction hypothesis (3) to promote further inference with the inequality \( 1 + -2 \ast count(p_1, i_2, No) + -1 \ast ml(p_1, i_2) + 1 \ast i_2 \leq 0 \). SPIKE instantiates (3) with the substitution \( \{ c_1 \mapsto No, i_1 \mapsto i_2 \} \) thereby obtaining the following instance:

\[
No \neq mc(p_1, i_2) \Rightarrow 2 \ast (ml(p_1, i_2) + count(p_1, i_2, No)) < s(i_2 + ml(p_1, i_2)) \tag{5}
\]

(The condition \( No \neq mc(p_1, i_2) \) is readily proved by the prover since it is a trivial consequence of the constraint store.) The extension of the constraint store with the linearization of \( 2 \ast (ml(p_1, i_2) + count(p_1, i_2, No)) < s(i_2 + ml(p_1, i_2)) \) yields a constraint store containing the impossible inequality \( 1 \leq 0 \) in the \( P \)-field. The unsatisfiability of the constraint store is then easily detected by \texttt{cs-unsat}.

8 Conclusions and Future Work

We have presented a general scheme for the integration of decision procedures with an implicit induction prover. The integration scheme is effective since when applied with Spike and decision procedures for equality and arithmetic it has given positive results on several non-trivial problems. Moreover, the soundness of our integration has been formally derived; this task is not obvious since we allow some interleaving between induction hypothesis application and decision-procedure application.

We plan to apply the integration scheme to new decision procedures for new decidable theories such as those for lists and arrays [ARR01]. At the same time, we will apply the theorem-prover to the verification of protocols like ABR [RSK00]. We would like also to exploit better for efficiency built-in AC operators by using AC-matching as in [BBR96].

References


\(^{5}\) To simplify the notation we represent only the non-empty fields of the structure and we tag the non-empty field with the corresponding name.


