Article Submitted to Journal of Symbolic Computation

Incorporating Decision Procedures in Implicit Induction

Alessandro Armando[1], Michaël Rusinowitch[2] and Sorin Stratulat[3]

[1] *DIST – Università di Genova, Viale Causa 13 – 16145 Genova – Italia*
[2] *LORIA-INRIA, 615, rue du Jardin Botanique – 54602 Villers lès Nancy – France*
[3] *INRIA – 2004 route des Lucioles – BP 93 – 06902 Sophia Antipolis – France*

Abstract

In this paper we present an approach to integrating reasoning specialists into Cover Set Induction based on Constraint Contextual Rewriting. The approach has been successfully used to incorporate decision procedures into the SPIKE prover. Computer experiments on non-trivial verification problems illustrating the effectiveness of the proposed technique are given. The generality of the approach allows for the integration of Computer Algebra algorithms and techniques into induction theorem provers. To illustrate this, we discuss the integration of Buchberger algorithm into our framework.

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 [Boyer and Moore, 1985], 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 (see

e.g. [Bouhoula and Rusinowitch, 1995]) is a powerful automated reasoning technique for reasoning about inductively defined objects which combines the advantages of explicit induction and proof by consistency. For instance, Cover Set Induction allows for an easier management of mutual induction and permits the refutation of false conjectures (under suitable conditions) [Bouhoula, 1997]. Constraint Contextual Rewriting [Armando and Ranise, 1998, 2000], CCR or CCR(X)* for short, is an abstract integration schema between rewriting and decision procedures. CCR(X) generalizes contextual rewriting [Zhang and Remy, 1985, Zhang, 1995] 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 [Boyer and Moore, 1985], this feature is crucial to the effectiveness of the integration. A key contribution of this paper is an extension to CCR that enables the use of induction hypotheses (other than the available definitions and lemmas) during the augmentation activity.

An extended version of the induction proof method presented in this paper has been implemented in the SPIKE prover [Bouhoula, 1997] along with a combination of a decision procedure for the union of the quantifier-free theory of equality (UTE) and of a semi-decision procedure for the quantifier-free theory of Presburger Arithmetics over the Integers (UPAI). The SPIKE induction prover has been designed to verify clausal formulas in the initial model of theories presented by first-order conditional rules.

Computer experiments on non-trivial verification problems give evidence of the effectiveness of our approach: the soundness proof of the MJRTY algorithm [Boyer and Moore, 1991] does not need anymore user-defined tactics or lengthy interactions as it is the case with Nuprl [Jackson, 1994] and STeP [Bjørner et al., 1995]. On the other hand the generality of the approach allows for the integration of Computer Algebra algorithms and techniques into induction theorem provers. To illustrate this, we discuss the integration of Buchberger algorithm into our framework.

Structure of the paper. In Section 2 we introduce some notions and notations that we use throughout the paper. In Section 3 we present the concept of reasoning specialist by specifying the required interface functionalities. Constraint Contextual Rewriting and the principle of Cover Set Induction are presented in Section 4 and Section 5 respectively. In Section 6 we first describe the reasoning specialist for the union of the quantifier-free theory of equality and the quantifier-free Presburger Arithmetics as implemented in SPIKE and then we show how it contributes to the proof of the MJRTY algorithm. In Section 7 we introduce Buchberger algorithm as a method for reasoning on non-linear equations and we

*The notation CCR(X) (by analogy with the CLP(X) notation used to denote the Constraint Logic Programming paradigm [Jaffar and Lassez, 1987]) is used to stress the independence of CCR(X) from the theory decided by the decision procedure.

illustrate the advantage of incorporating reasoning specialists capable of dealing with non-linearities into Cover Set Induction provers by means of a worked out example. We conclude in Section 8 with some final remarks and with an outline of the future work.

2. Preliminaries

By Σ (possibly subscripted) we denote finite sets of function symbols (with their arity). V (possibly subscribed) denotes a finite set of variables. $\tau(\Sigma, V)$ is the set of *terms* built out of Σ 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 [Dershowitz and Jouannaud, 1990b]. A Σ-*equation* is an expression of the form $t_1 = t_2$ where $t_1, t_2 \in \tau(\Sigma, V)$. (Σ, V)-*formulae* are built in the usual way using the standard logical connectives (i.e. $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$). A (Σ, V)-*literal* is either a (Σ, V)-equation or a negated (Σ, V)-equation. We write Σ-equation (-literal) instead of $(\Sigma, \emptyset)$-atom (-literal, resp.). A (Σ, V)-*clause* is a disjunction of literals which we indicate as finite set of (Σ, V)-literals. We denote by $\mathtt{true}$ 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 $\overline{\neg a}$ stands for a; similarly, if E is a set of literals, then $\overline{E}$ abbreviates $\{\overline{p} : p \in E\}$. It is convenient to use $(p_1, \ldots, p_n \Rightarrow p)$ as an abbreviation of the clause $\{\overline{p}_1, \ldots, \overline{p}_n, p\}$. Moreover $\bigwedge S$ stands for any conjunction of the literals in S.

In the following we consider two theories $T_c \subseteq T_j$ of signature $\Sigma_c \subseteq \Sigma_j$ respectively. By T_c we indicate the theory decided by the reasoning specialist, whereas T_j is the theory dealt by the induction prover. By $\prec$ we indicate a reduction ordering over the (Σ_j, V)-expressions (i.e. a well-founded relation over the (Σ_j, V)-expressions closed under substitution and replacement) containing the sub-expression relation. We also assume that $\mathtt{true} \prec e$ for all equations $e \neq \mathtt{true}$. Given a congruence relation $\approx$ on terms that is stable (i.e., $s\sigma \approx t\sigma$ if $s \approx t$) and compatible with $\prec$ (i.e., $s' \prec t'$ if $s \prec t$, $s \approx s'$, and $t \approx t'$) we define $\preceq$ as $\prec \cup \approx$. The notation $\prec\!\!\prec$ is used for the multiset extension of $\prec$. A conditional equation $(a_1 = b_1, \ldots, a_n = b_n \Rightarrow l = r)$ can be oriented into the *conditional rewrite rule* $(a_1 = b_1, \ldots, a_n = b_n \Rightarrow l \rightarrow r)$ if, for each substitution σ, $\{r\sigma, a_1\sigma, b_1\sigma, \ldots, a_n\sigma, b_n\sigma\} \prec\!\!\prec \{l\sigma\}$. 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 $t[l\sigma]_u \rightarrow_R t[r\sigma]_u$ iff σ is a substitution and $a_1 = b_1, \ldots, a_n = b_n \Rightarrow l \rightarrow r$ is a conditional rewrite rule in R such that for any $i = 1, \ldots, n$, there exists a term c_i with $a_i\sigma \rightarrow_R^* c_i$ and $b_i\sigma \rightarrow_R^* c_i$. A term t is *R-irreducible* if there is no term s such that $t \rightarrow_R s$. If $s \rightarrow_R^* t$ 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 ordering over clauses $\prec_c$ closed under substitution and well-founded, built from a multiset extension

of $\prec$ [Bouhoula and Rusinowitch, 1995, Naidich, 1996]. Let $\mathcal{S}$ be the set of substitutions, C and C' be clauses, and $\lessdot \in \{\prec_c, \preceq_c\}$, then $C_{\lessdot C'} = \{C\sigma : C\sigma \lessdot C' \text{ and } \sigma \in \mathcal{S}\}$. If E is a set of clauses then $E_{\lessdot C'}$ is the set of clauses $\{C_{\lessdot C'} : C \in E\}$.

Let Ax be a set of axioms, then we say that ϕ is an *initial consequence* of Ax, in symbols $Ax \models_{ini} \phi$, iff ϕ is valid in the initial model of Ax. In the sequel T_j is the (Σ_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 ϕ is T_j-*unsatisfiable* iff ϕ is not valid in the initial model of Ax and that ϕ is T_c-*unsatisfiable* iff no model of T_c is also a model of ϕ. Similarly, we say that ϕ is T_j-*valid* iff ϕ is an initial consequence of Ax and that ϕ is T_c-*valid* iff all models of T_c are also models of ϕ.

3. Reasoning Specialists

According to the usual definition, a decision procedure for T_c is a procedure which takes a (Σ_c, Π_c)-formula as input and returns a 'yes-or-no' answer indicating whether the input formula is T_c-satisfiable or not. Unfortunately, although simple and conceptually elegant, this definition is seldom adequate in practical applications. Efficiency considerations require the procedure to be *incremental*, i.e. capable of processing parts of the input problem as soon as they become available (see, e.g. Nelson and Oppen [1978] for a discussion on this issue). This generalized notion of decision procedure is captured by the notion of reasoning specialist. A *reasoning specialist* is a state-based procedure whose states (called *constraint stores*) are finite sets of (Σ_c, Π_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.

Constraint Store Simplification. The main functionality of the reasoning specialist is a transition relation over constraint stores, $S \xrightarrow[P]{cs-simp} S'$, which models the activity of adding a finite set of Σ_j-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 $S \xrightarrow[P]{cs-simp} S'$.

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

[†]If C is a constraint store, then $\|C\|$ denotes the set of literals represented by C. To simplify the presentation we will often blur the distinction between constraint stores and the set of formulae they represent. For instance, we will talk about the satisfiability of C meaning the satisfiability of $\|C\|$.

a computationally inexpensive syntactic check. We require that $\texttt{cs-unsat}(S)$ is decidable and that $\texttt{cs-unsat}(S)$ implies the T_c-unsatisfiability of S.

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 $\texttt{cs-init}(S_0)$, $S_0 \xrightarrow[P]{cs-simp} S$, and $\texttt{cs-unsat}(S)$ then P is T_c-unsatisfiable. This observation shows that the functionalities we have presented so far allow us to check the T_c-unsatisfiabily of any given set of literals P.

EXAMPLE 3.1 (A REASONING SPECIALIST FOR TOTAL ORDERS): *Let T_c be a theory for total orders. Constraint stores are finite sets of literals of the form $t_1 \leq t_2$, $t_1 = t_2$, or $t_1 \neq t_2$ closed under the following inference rules:*

$$\frac{t_2 \leq t_2 \quad t_2 \leq t_3}{t_1 \leq t_3} \textit{ (transitivity)} \qquad \frac{t_1 \leq t_2 \quad t_2 \leq t_1}{t_1 = t_2} \textit{ (antisymmetry)}$$

$\texttt{cs-init}(C)$ *holds iff* $C = \emptyset$. $\texttt{cs-unsat}(C)$ *holds iff C contains two literals of the form $t_1 = t_2$ and $t_1 \neq t_2$. In this simple case, the set of literals represented by a constraint store C coincides with the constraint store itself, i.e.* $\|C\| = C$.

Let ν be the function associating sets of Σ_c-literals to Σ_c-literals as defined in Table 1. This function is extended to sets of Σ_c-literals, say P, by $\nu(P) = \bigcup_{c \in P} \nu(c)$. *The* $C \xrightarrow[P]{cs-simp} C'$ *relation holds iff C' is the result of closing $C \cup \nu(P)$*

Table 1: Definition of $\nu(c)$

c	$\nu(c)$	c	$\nu(c)$	c	$\nu(c)$
$t_1 \leq t_2$	$\{t_1 \leq t_2\}$	$t_1 \neq t_2$	$\{t_1 \neq t_2\}$	$t_1 \not< t_2$	$\{t_2 \leq t_1\}$
$t_1 = t_2$	$\{t_1 \leq t_2, t_2 \leq t_1\}$	$t_1 < t_2$	$\{t_1 \leq t_2, t_1 \neq t_2\}$	$t_1 \not\leq t_2$	$\{t_2 \leq t_1, t_1 \neq t_2\}$
$t_1 > t_2$	$\nu(t_2 < t_1)$	$t_1 \not> t_2$	$\nu(t_1 \leq t_2)$	$t_1 \geq t_2$	$\nu(t_2 \leq t_1)$
$t_1 \not\geq t_2$	$\nu(t_1 < t_2)$	*otherwise*	$\emptyset$		

w.r.t. the application of (transitivity) and (antisymmetry). It is easy to verify that $\xrightarrow{cs-simp}$ *enjoys the associated soundness requirement, i.e. that* $C \xrightarrow[P]{cs-simp} C'$ *only if* $T_c \models \bigwedge(P \cup C) \Rightarrow \bigwedge C'$.

Augmentation. It is often the case that the constraint store S is T_j-unsatisfiable but not T_c-unsatisfiable. In such a case, it is not sufficient to reason solely within the theory T_c as the T_j-unsatisfiability of S cannot possibly be detected by the reasoning specialist. The occurrence symbols interpreted in T_j but not in T_c is the main cause of the problem. Some more information from the specification is required for deriving the unsatisfiability. 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 constraint store, the heuristics aims at generating a T_j-equivalent but T_c-unsatisfiable constraint store 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 the set of axioms R (given as inputs and oriented as rewrite rules) or the set of formulas $\mathcal{H}$ that—as we will see in Section 5—contains the available induction hypotheses. To model augmentation we define a new relation $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ as the smallest transitive relation (i.e. such that $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ and $S' \xrightarrow[R;\mathcal{H};P]{decproc} S''$ imply $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ for all R, $\mathcal{H}$, and P) such that:

$$\text{CS-SIMP:} \quad S \xrightarrow[R;\mathcal{H};P]{decproc} S' \qquad \text{if } S \xrightarrow[P]{cs-simp} S'$$

$$\text{AUGMENT:} \quad S \xrightarrow[R;\mathcal{H};P]{decproc} S'$$
$$\text{if } \ (Q \Rightarrow c) \in R \text{ or } (Q \Rightarrow c)\sigma \in \mathcal{H},\ c\sigma \text{ is a } \Sigma_c\text{-literal},$$
$$q\sigma \xrightarrow[R;\emptyset;S]{iccr} \texttt{true} \text{ for all } q\sigma \in Q, \text{ and } S \xrightarrow[R;\mathcal{H};\{c\sigma\}]{decproc} S'$$

where $\xrightarrow{iccr}$ is the constraint contextual rewriting relation we define next.

EXAMPLE 3.2 (AUGMENTATION): *Let T_j be a theory of integer numbers with the usual interpretation of the symbols. Let $\Sigma_c = \Sigma_j$ and let T_c and $\xrightarrow{cs-simp}$ be as in Example 3.1. Under such hypotheses the following formula*

$$X \geq 4 \Rightarrow X^2 \leq 2^X \tag{1}$$

is in T_j but not in T_c and we assume that R contains it. We want to prove that the formula

$$n \geq 5,\ n^2 \geq 2^n \Rightarrow n^2 = 2^n \tag{2}$$

is T_j-valid. To this end we add the literals $n \geq 5$, $n^2 \geq 2^n$, and $n^2 \neq 2^n$ to the empty constraint store, thereby getting the constraint store

$$\{5 \leq n, 2^n \leq n^2, n^2 \neq 2^n\} \tag{3}$$

which is not found T_c-unsatisfiable by `cs-unsat`*. Notice however that* (3) *is T_j-unsatisfiable. Rule* AUGMENT *selects the instance of* (1) *obtained by substituting X with n. This instance is obviously T_j-valid and, since $n \geq 5$ occurs in the constraint store, $n^2 \leq 2^n$ is T_j-entailed by it. Therefore* AUGMENT *adds $n^2 \leq 2^n$ to* (3) *thereby leading to the new constraint store:*

$$\{5 \leq n, n^2 \leq 2^n, 2^n \leq n^2, 2^n = n^2, n^2 = 2^n, n^2 \neq 2^n\}$$

which is readily found T_c-unsatisfiable (and hence T_j-unsatisfiable) by `cs-unsat`. *This allows us to conclude the T_j-unsatisfiability of* (3) *and hence the T_j-validity of* (2).

Notice that the task of establishing the T_j-validity of (2) *falls largely beyond the scope of a decision procedure for total orders. The problem is nevertheless solved thanks to the use of the augmentation heuristics.* △

4. Constraint Contextual Rewriting

Let R be a set of rewrite rules, $\mathcal{H}$ a set of clauses, and S a constraint store. Constraint Contextual Rewriting is modeled by the relation $e \xrightarrow[R;\mathcal{H};S]{iccr} e'$ meaning that e' is the result of (constraint contextual) rewriting e in the rewrite context represented by R, $\mathcal{H}$, and S. We start by observing that a literal e can be rewritten to `true` if the result of extending the S with the negation of e yields an unsatisfiable constraint store. This is formalized by the following rule:

ENTAILMENT CHECK: $e \xrightarrow[R;\mathcal{H};S]{iccr} \texttt{true}$
if e is a Σ_c-literal, $e \neq \texttt{true}$, $S \xrightarrow[R;\mathcal{H};\{\overline{e}\}]{decproc} S'$ and `cs-unsat`(S')

Conditional rewriting is formalized by the rule:

CONDITIONAL REWRITING: $e[l\sigma] \xrightarrow[R;\mathcal{H};S]{iccr} e[r\sigma]$
if $(Q \Rightarrow l \to r) \in R$ or $(Q \Rightarrow l \to r)\sigma \in \mathcal{H}$, and $q\sigma \xrightarrow[R;\mathcal{H};S]{iccr} \texttt{true}$ for all $q \in Q$.

i.e. rewrite the sub-expression $l\sigma$ in e to $r\sigma$ in the rewrite context represented by R, $\mathcal{H}$, and S if a clause of the form $(Q \Rightarrow l \to r)$ is in R or $(Q \Rightarrow l \to r)\sigma$ is in R, where σ is a substitution and all the conditions in $Q\sigma$ can be recursively established in the same rewrite context.

THEOREM 4.1 (SOUNDNESS OF CCR):

1. *If* $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ *then* $R, \mathcal{H}, P, S \models_{ini} \bigwedge S'$;
2. *If* $e \xrightarrow[R;\mathcal{H};S]{iccr} e'$ *then* $R, \mathcal{H}, S \models_{ini} (e \sim e')$ *and* $e' \prec e$;

Proof: Since the definitions of $\xrightarrow{decproc}$ and $\xrightarrow{iccr}$ 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 $\xrightarrow{decproc}$ and $\xrightarrow{iccr}$ in the derivation is 1) amounts to proving the following case:

- $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ results from the application of CS-SIMP and therefore S' is such that $S \xrightarrow[P]{cs-simp} S'$ and hence $T_c \models \bigwedge(P \cup S) \Rightarrow \bigwedge S'$ holds. 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:

- $S \xrightarrow[R;\mathcal{H};P]{decproc} S'$ results from the application of AUGMENT and therefore S' is such that $S \xrightarrow[R;\mathcal{H};\{c\sigma\}]{decproc} S'$ where either $(Q \Rightarrow c) \in R$ or $(Q \Rightarrow c)\sigma \in \mathcal{H}$ and $q\sigma \xrightarrow[R;\emptyset;S]{iccr} \mathtt{true}$ for all $q\sigma \in Q$. From the induction hypothesis we know that $R, S \models_{ini} q\sigma$ for all $q \in Q$. From this and the fact that $(Q \Rightarrow c) \in R$ or $(Q \Rightarrow c)\sigma \in \mathcal{H}$ it readily follows that $R, \mathcal{H}, S \models_{ini} c\sigma$. By induction hypothesis we also know that $R, \mathcal{H}, c\sigma, S \models_{ini} \bigwedge S'$ and therefore we can conclude that $R, \mathcal{H}, S \models_{ini} \bigwedge S'$ and hence 1.
- $e \xrightarrow[R;\mathcal{H};S]{iccr} e'$ results from the application of ENTAILMENT CHECK and hence e is a Σ_c-literal and $e' = \mathtt{true}$. In this case we know that $S \xrightarrow[R;\mathcal{H};\{\overline{e}\}]{decproc} S'$ and $\mathtt{cs\text{-}unsat}(S')$. By induction hypothesis we have $R, \mathcal{H}, \overline{e}, S \models_{ini} \bigwedge S'$ and from the T_c-unsatisfiability of S' it readily follows that $R, \mathcal{H}, S \models_{ini} e$. It is immediate to see that $\mathtt{true} \prec e$ holds.
- $e \xrightarrow[R;\mathcal{H};S]{iccr} e'$ results from the application of CONDITIONAL REWRITING and there exists a substitution σ and a clause $Q \Rightarrow l = r$ such that $l\sigma$ occurs in e, i.e. $e = e[l\sigma]$, $e' = e[r\sigma]$, $Q \prec\!\!\prec \{l\sigma = r\sigma\}$, $r\sigma \prec l\sigma$, and either $(Q \Rightarrow l = r) \in R$ or $(Q \Rightarrow l \rightarrow r)\sigma \in \mathcal{H}$; moreover $q\sigma \xrightarrow[R;\mathcal{H};S]{iccr} \mathtt{true}$ for all $q \in Q$. By induction hypothesis we know that $R, \mathcal{H}, S \models_{ini} q\sigma$ holds for all $q \in Q$. If $(Q \Rightarrow l = r) \in R$ we can conclude that $R, \mathcal{H}, S \models_{ini} l\sigma = r\sigma$, and therefore $R, \mathcal{H}, S \models_{ini} e[l\sigma] \sim e[r\sigma]$. Also, $e[r\sigma] \prec e[l\sigma]$ holds since $r\sigma \prec l\sigma$ and $\prec$ is monotonic and stable. The case in which $(Q \Rightarrow l \rightarrow r)\sigma \in \mathcal{H}$ is proven along the same lines.

□

CCR is used to simplify clauses by rewriting. Clause simplification is modeled by the relation $E \xrightarrow[R;\mathcal{H}]{simplify} E'$ meaning that a set of clauses E simplifies to a set

of clauses E' in a simplification context represented by R and $\mathcal{H}$. We start by observing that a clause can be safely deleted from a set of clauses if $\mathtt{true}$ is among its disjuncts, if it occurs in $\mathcal{H}$, or if it is an instance of one of the clauses in R. This is formalized by the following rule:

DELETE: $\{C\} \cup E \xrightarrow[R;\mathcal{H}]{simplify} E$

if $\mathtt{true} \in C$, $C \in \mathcal{H}$, or there exists σ s.t. $C\sigma \in R$

Furthermore, a clause can be simplified by constraint contextual rewriting any literal in it, say p, into a new literal p', as specified by the following rule:

CCR: $\{\{p\} \cup C\} \cup E \xrightarrow[R;\mathcal{H}]{simplify} \{\{p'\} \cup C\} \cup E$

if $\mathtt{cs\text{-}init}(S_0)$, $S_0 \xrightarrow[R;\mathcal{H};\overline{C}]{decproc} S$ and $p \xrightarrow[R;\mathcal{H};S]{iccr} p'$

Here p is called the *focus literal* and C the *rewriting context.*

THEOREM 4.2 (SOUNDNESS OF CLAUSE SIMPLIFICATION): *If* $E \xrightarrow[R;\mathcal{H}]{simplify} E'$ *and* $\prec_c$ *is monotonic, then for all clauses* $C \in E$ *either* $\mathtt{true} \in C$ *or there exists a clause* $C' \in E'$ *such that* $R, \mathcal{H} \models_{ini} (C \Leftrightarrow C')$ *and* $C' \prec_c C$.

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

EXAMPLE 4.1 (CONSTRAINT CONTEXTUAL REWRITING): *Let* T_j, T_c, *and* $\xrightarrow{decproc}$ *be as in Example 3.2 and let* R *contain* (1) *and the following conditional rewrite rule:*

$$X^2 = \sqrt{X^2},\ X \geq 0\ \Rightarrow \sqrt{2^X} \rightarrow X \tag{4}$$

We want to prove the T_j*-validity of*

$$n \geq 4,\ n^2 \geq 2^n \Rightarrow \sqrt{2^n} = n \tag{5}$$

To this end, we invoke the CCR *rule upon the singleton set of clauses containing* (5). *Let* $\sqrt{2^n} = n$ *be the focus literal and* $\{n \not\geq 4,\ n^2 \not\geq 2^n\}$ *be the rewriting context. Application of* $\emptyset \xrightarrow[R;\mathcal{H};\{n\geq 4,\ n^2\geq 2^n\}]{decproc} C$ *yields* $C = \{4 \leq n, 2^n \leq n^2\}$, *and application of* $(\sqrt{2^n} = n) \xrightarrow[R;\mathcal{H};C]{iccr} p'$ *triggers the rules* CONDITIONAL REWRITING *and* ENTAILMENT. *The former applies* (5) *to rewrite the focus literal to* $n = n$ *which is then simplified to* $\mathtt{true}$ *by the latter. Notice that, preliminary to its application, the conditions of* (2) *are relieved by a recursive application of* $\xrightarrow[R;\mathcal{H};C]{iccr}$ *which in turn invokes the reasoning specialist and the augmentation facility.* △

5. Cover Set Induction

Given a set of conditional rules, cover set induction is based on the idea of computing *covering substitutions*, i.e. a family of substitutions covering all possibles cases for induction variables. These substitutions are applied to conjectures thereby *generating* special instances which are then *simplified* by the available rewrite rules, lemmas, and induction hypotheses. The whole process is then iterated on the resulting set of clauses.

More in detail, 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 γ 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 $Ax \models t\sigma = s$. From a cover set for a conditional rewrite system, we can build cover sets for clauses. Let C be a clause, a *cover substitution for* C instantiates a particular subset of the variables of C (called induction variables) with terms obtained from $CS(R)$ (whose variables are first standardized apart). We will denote by $CS\Sigma(C)$ the set of all possible cover substitutions for C. Then, the set $\{C\sigma \mid \sigma \in CS\Sigma(C)\}$ is a *cover set for* 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 $(E, H) \xrightarrow[Ax]{spike} (E', H')$ which is defined to be the smallest transitive relation such that:

$$\textsc{Generate:}\quad (E \cup \{C\}, H) \xrightarrow[Ax]{spike} (E \cup \textstyle\bigcup_{\sigma \in CS\Sigma(C)} E_\sigma, H \cup \{C\})$$
$$\text{if } \{C\sigma\} \xrightarrow[R;(E\cup H\cup\{C\})_{\prec_c C\sigma}]{simplify} E_\sigma \text{ for } \sigma \in CS\Sigma(C)$$

$$\textsc{Simplify:}\quad (E \cup \{C\}, H) \xrightarrow[Ax]{spike} (E \cup E', H)$$
$$\text{if } \{C\} \xrightarrow[R;H_{\preceq_c C}\cup E_{\prec_c C}]{simplify} E'$$

The set of induction hypotheses available for the simplification of the cover-set instance $C\sigma$ are instances of the current set of E, $\{C\}$, and H strictly smaller (w.r.t. $\prec_c$) than $C\sigma$. The SIMPLIFY inference rule transforms a conjecture into a (possibly empty) set of simpler conjectures.

A clause E_0 is an inductive theorem w.r.t. Ax if there exists a finite derivation of the form $(E_0, \emptyset) \xrightarrow[Ax]{spike} \cdots \xrightarrow[Ax]{spike} (\emptyset, H)$. More in general, we say that E_0 is an *inductive theorem w.r.t.* Ax, in symbols $Ax \vdash_{ini} E_0$, iff there exists a fair derivation $(E_0, \emptyset) \xrightarrow[Ax]{spike} (E_1, H_1) \xrightarrow[Ax]{spike} \cdots$, i.e. iff the set of persisting clauses $\bigcup_{i\geq 0} \bigcap_{j\geq i} E_j$ is empty.

THEOREM 5.1 (SOUNDNESS OF COVER SET INDUCTION): *If* $Ax \vdash_{ini} E_0$ *then* $Ax \models_{ini} E_0$.

Proof (Adapted from [Bouhoula, 1997]): Let us assume that $Ax \vdash_{ini} E_0$ but $Ax \not\models_{ini} E_0$. Since $Ax \vdash_{ini} E_0$, then there exists a fair derivation $(E_0, \emptyset) \xrightarrow[Ax]{spike} (E_1, H_1) \xrightarrow[Ax]{spike} \cdots$. 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

$$CE = \{D\theta : D \in \bigcup_i E_i \text{ and } Ax \not\models_{ini} D\theta \text{ for some ground } R\text{-irreducible substitution } \theta\}.$$

(Notice that CE is not empty since, by assumption, $Ax \not\models_{ini} E_0$; 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 ϕ is ground and R-irreducible, then there exists $\sigma \in CS\Sigma(C)$ and a ground substitution τ such that $\phi = \sigma\tau$. From Theorem 4.2 it follows that there exists a clause $C' \in \bigcup_i E_i$ such that $R, (E \cup H \cup \{C\})_{\prec C\phi} \models_{ini} (C\phi \Leftrightarrow C'\tau)$ with $C'\tau \prec C\phi$.[‡] Notice that $Ax \models_{ini} (E \cup H \cup \{C\})_{\prec C\phi}$ must hold too (otherwise the minimality of $C\phi$ in CE would be contradicted) and hence $Ax \models_{ini} (C\phi \Leftrightarrow C'\tau)$. Since $Ax \not\models_{ini} C\phi$, then also $Ax \not\models_{ini} C'\tau$ which, together with $C'\tau \prec C\phi$, contradicts the minimality of $C'\phi$ in CE.
2. Case: SIMPLIFY. From Theorem 4.2 it follows that there exists a clause $C' \in \bigcup_i E_i$ such that $R, E_{\prec C\phi} \cup H_{\preceq C\phi} \models_{ini} (C\phi \Leftrightarrow C'\phi)$ and $C'\phi \prec C\phi$. 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 $C_1\theta \approx_c C\phi$ for some ground R-irreducible substitution θ, then $Ax \models_{ini} C_1\theta$. Let us assume that $Ax \not\models_{ini} C_1\theta$, 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.

□

EXAMPLE 5.1: *Let* R *be a rewrite system that mutually defines the unary pred-*

[‡]Notice that true cannot possibly be in $C\phi$ as this would contradict the assumption that $C\phi \in CE$.

icates even and odd over unbounded lists:

$$even(nil) \rightarrow \text{T} \tag{6}$$
$$even(cons(A, L)) \rightarrow odd(L) \tag{7}$$

$$odd(nil) \rightarrow \text{F} \tag{8}$$
$$odd(cons(A, L)) \rightarrow even(L) \tag{9}$$

The predicate $even(l)$ (resp. $odd(l)$) is true if the list l contains an even (resp. odd) number of elements. The symbols nil and cons are the list constructors and the cover set for R is $\{nil, cons(x, y)\}$.

Let us consider the problem of proving

$$\forall\ l : list.(even(l) = \text{T} \vee odd(l) = \text{T}) \tag{10}$$

The initial state of the proof is $(\{C\},\ \emptyset)$ where $C = \{even(l) = \text{T},\ odd(l) = \text{T}\}$. Since Simplify *cannot be applied,* Generate *computes the cover set for C which comprises the clauses*

$$\{even(nil) = \text{T},\ odd(nil) = \text{T}\} \tag{11}$$
$$\{even(cons(x, y)) = \text{T},\ odd(cons(x, y)) = \text{T}\} \tag{12}$$

By rewriting with (6) *and* (8), (11) *gets simplified to*

$$\{\text{T} = \text{T},\ \text{F} = \text{T}\} \tag{13}$$

Similarly, by rewriting with (7) *and* (9), *12 gets simplified to*

$$\{odd(y) = \text{T},\ even(y) = \text{T}\} \tag{14}$$

Clause C is then stored as an induction hypothesis and the new state of the proof is $(E, \{C\})$, where E is the set of clauses comprising (13) *and* (14) *which are then eliminated by a double application of* Delete *thereby leaving us with the state $(\emptyset, \{C\})$. This concludes the proof of* (10). △

6. Integrating a Reasoning Specialist for the Union of UTE and UPAI

We present a reasoning specialist for the union of UTE and UPAI obtained by combining a decision procedures for the two component theories. Therefore in this section T_c = UPAI ∪ UTE. The interface functionalities of the compound decision procedure are as described in Section 3 and we show that they comply with the requirements stated in the same section. Constraint stores are of the

form $\langle U \parallel G \mid I \rangle$ where U is a set of ground rewrite rules[§], G is a set of ground equations and disequations, and I is a set of linear inequalities of the form

$$c_1 \cdot m_1 + \cdots + c_n \cdot m_n \leq c \tag{15}$$

where $n \geq 0$ (if $n = 0$, then (15) stands for $0 \leq c$), $c, c_1, \ldots, c_n$ are relatively prime integers, $m_1, \ldots, m_n$ are terms whose top-most function symbols are different from $+$ s.t. $m_{i+1} \prec m_i$, and $c_i \cdot m_i$ $(i = 1, \ldots, n)$ abbreviates the term $m_i + \cdots + m_i$ in which m_i occurs c_i times. A ground rewrite rule in U is an oriented equation from G which in turn can be an instance of an axiom, a lemma, or an induction hypothesis. Such rules are used to normalize the literals in I. The field G represents the state of a congruence closure procedure, modeled by the function *congr*, whose main activity is to derive new ground equalities from those already in G. Similarly, I represents the state of a (semi-)decision procedure for linear arithmetics, modeled by the function *arith*, whose main activity is to derive new linear inequalities along with implicit equations from the inequalities in I. `cs-init`$(\langle U \parallel G \mid I \rangle)$ holds iff all the component fields are empty.

Constraint store simplification, modeled by the relation $\langle U \parallel G \mid I \rangle \xrightarrow[P]{\text{cs-simp}} \langle U' \parallel G' \parallel I' \rangle$, amounts to exchanging information among the fields of the constraint store. Literals in P are initially distributed to G and I as specified by the following rule:

$$\textsc{Init}: \quad \langle U \parallel G \parallel I \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \parallel G \cup P_G \mid I \cup P_I \rangle$$

where P_G comprises the equalities and the negated equalities of P and P_I is a set of inequalities of the form (15) obtained from the inequalities of P by eliminating $<, =, \geq$, and $>$ in favor of $\leq$ (e.g. $x < 0$ can be rewritten to $x \leq -1$ by exploiting the integral property of the integers).

Equations in G are then processed by an algorithm for ground completion [Huet and Lankford, 1978] and the rewrite rules derived by the algorithm are added to U:

$$\textsc{Congr}: \quad \langle U \mid G \mid I \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \cup E \parallel \mathit{congr}(G) \parallel I \rangle \quad \text{where } E = \{a \to b : a = b \in \mathit{congr}(G) \text{ and } b \prec a\}$$

The inequalities in I are processed by a (semi-)decision procedure for linear arithmetics based on the Fourier-Motzkin variable elimination method [Lassez and Maher, 1992] modeled by *arith*. This function takes a set of linear inequalities as input and returns a set of inequalities and a set of entailed equations which are added to I and E respectively.

[§]From now on, by ground rewrite rule (resp. equation) we mean a rewrite rule (resp. equation) where variables are considered as new constants and therefore cannot be instantiated.

$$\textsc{Arith}: \quad \langle U \mid G \mid I \rangle \xrightarrow[P]{\mathsf{cs-simp}} \langle U \parallel G \cup E \parallel I \cup I' \rangle$$
$$\text{where } (I', E) = \mathit{arith}(I)$$

The rewrite rules in U are used to normalize the literals in G and the inequalities in I:

$$\textsc{Normalize}: \quad \langle U \mid G \parallel I \rangle \xrightarrow[P]{\mathsf{cs-simp}} \langle U \mid G' \mid I' \rangle$$
where I' and G' are the results of
normalizing I and G (resp.) with the rules in U.

Finally $\texttt{cs-unsat}(\langle U \parallel G \mid I \rangle)$ holds iff there exist either a disequation of the form $a \neq a$ in G or a trivially T_c-unsatisfiable inequality (e.g. $0 \leq -1$) in I.

Theorem 6.1 (Soundness of the Reasoning Specialist): *The following facts hold:*

- *If* $\texttt{cs-unsat}(S)$ *then* $\|S\|$ *is* T_c*-unsatisfiable.*
- *If* $S \xrightarrow[A]{cs-simp} S'$ *then* $T_c \models \bigwedge \|S'\| \Leftrightarrow \bigwedge(A \cup \|S\|)$.

A proof of this result can be found in [Stratulat, 2000].

Example 6.1 (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. MJRTY has been devised in 1980 by Boyer and Moore who have also proved its soundness by means of their prover NQTHM [Boyer and Moore, 1979]. 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 [Owre et al., 1992], Nuprl [Howe, 1993] and STeP [Bjørner, 1998].*

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. Let p be a poll of i votes. MJRTY computes a pair (mcv, mlv)*, where mcv is the majority candidate and mlv is its lead over the other candidates. If the poll has no votes, then the majoritary candidate is an arbitrary value no with the lead* 1*. Otherwise, if the poll has* $i > 0$ *votes, we compute recursively the majoritary candidate for the first* $i-1$ *votes and its lead over the other candidates. If the i vote is for the majoritary candidate then its lead is incremented by* 1*. Otherwise, if its lead is positive, it is decremented by* 1*. If the lead is zero, the new majoritary candidate is pointed out by the i vote and its lead is set to* 1*. MJRTY can be specified by means of two mutually recursive*

functions, mc and ml, which compute the majority candidate and its lead over the other candidates respectively:

$$\begin{aligned} P[I] = mc(P,I) &\Rightarrow mc(P,s(I)) = mc(P,I) \\ P[I] \neq mc(P,I),\ 0 < ml(P,I) &\Rightarrow mc(P,s(I)) = mc(P,I) \\ P[I] \neq mc(P,I),\ 0 \not< ml(P,I) &\Rightarrow mc(P,s(I)) = P[I] \\ mc(P,0) &= no \end{aligned}$$

$$\begin{aligned} P[I] = mc(P,I) &\Rightarrow ml(P,s(I)) = s(ml(P,I)) \\ P[I] \neq mc(P,I),\ 0 < ml(P,I) &\Rightarrow ml(P,s(I)) = ml(P,I) - 1 \\ P[I] \neq mc(P,I),\ 0 \not< ml(P,I) &\Rightarrow ml(P,s(I)) = 1 \\ ml(P,0) &= 1 \end{aligned}$$

where $p[i]$ is the binary function (in mix-fix notation) that returns the i-th element of the list p. 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:

$$\forall p : list. \forall i : nat. \forall a : cand. (i < 2 * count(p,i,c) \Rightarrow c = mc(p,i)) \qquad (16)$$

where $count(p,i,c)$ counts the number of votes for a given candidate c from a poll p containing i votes and is defined by:

$$\begin{aligned} C = P[I] &\Rightarrow count(P,s(I),C) = s(count(P,I,C)) \\ C \neq P[I] &\Rightarrow count(P,s(I),C) = count(P,I,C) \\ count(P,0,Y) &= 0 \end{aligned}$$

The following lemma, first proposed by N. Shankar (according to [Howe, 1993]), simplifies in a major way the proof of (16)*:*

$$2 * (if(c, mc(p,i), 0, ml(p,i)) + count(p,i,c)) < s(i + ml(p,i)) \qquad (17)$$

where if is the 4-ary function defined by:

$$A = B \Rightarrow if(A,B,X,Y) = X \qquad (18)$$

$$A \neq B \Rightarrow if(A,B,X,Y) = Y \qquad (19)$$

Here we restrict our attention to the proof of (17) *and in doing so we focus on the steps in which the reasoning specialist plays a key role.*[¶] *The* SPIKE *prover starts with an application of case analysis suggested by* (18) *and* (19) *and leads to the following two new conjectures:*

$$c \neq mc(p,i) \Rightarrow 2 * (ml(p,i) + count(p,i,c)) < s(i + ml(p,i)) \qquad (20)$$

$$c = mc(p,i) \Rightarrow 2 * (0 + count(p,i,c)) < s(i + ml(p,i)) \qquad (21)$$

[¶]A detailed account of the proof of the lemma as carried out by SPIKE can be found in [Stratulat, 1998, 2001].

Application of GENERATE *to* (20) *moves it to the set of induction hypotheses and leads to the following sub-goal:*

$$c \neq mc(p, i_1), p[i_1] \neq mc(p, i_1), 0 < ml(p, i_1), no = p[i_1] \Rightarrow$$
$$2 * ((ml(p, i_1) - 1) + s(count(p, i_1, c))) < s(s(i_1) + (ml(p, i_1) - 1))$$

Application of the SIMPLIFY *rule invokes CCR which in turn invokes the reasoning specialist via the* ENTAILMENT CHECK *rule. Application of the* INIT *rule initializes the constraint store to:*

$$\langle \emptyset \parallel$$
$$\{c \neq mc(p, i_1), p[i_1] \neq mc(p, i_1), c = p[i_1]\} \parallel$$
$$\{-1 \cdot ml(p, i_1) \leq -1, \quad -2 \cdot count(p, i_1, c) - 1 \cdot ml(p, i_1) + 1 \cdot i_1 \leq -1\}\rangle$$

Since a contradiction has not been derived yet, the AUGMENT *rule is then applied using the induction hypothesis* (20) *to promote further inference with the inequality* $-2 \cdot count(p, i_1, c) - 1 \cdot ml(p, i_1) + 1 \cdot i_1 \leq -1$. SPIKE *instantiates* (20) *with the substitution* $\{c \mapsto i_1\}$ *thereby obtaining the following instance:*

$$c \neq mc(p, i_1) \Rightarrow 2 * (ml(p, i_1) + count(p, i_1, c)) < s(i_1 + ml(p, i_1)) \qquad (22)$$

(The condition $c \neq mc(p, i_1)$ *is readily proved by the prover since it is a trivial consequence of the constraint store.) The extension of the constraint store with the conclusion of* (22) *yields a constraint store containing the impossible inequality* $0 \leq -1$. *The unsatisfiability of the constraint store is then easily detected by* `cs-unsat`.

The proof of (21) *is analogous to that of* (20). △

7. Integrating a Reasoning Specialist for Non-Linear Equalities

In this section we show how to extend our induction prover with a procedure for handling non-linear polynomials in the same way as we extended the prover with linear arithmetics. Gröbner bases, invented by Bruno Buchberger [Buchberger, 1965], are a particular type of basis for polynomial ideals. Among other applications, they allow to check whether a non-linear equation is satisfied by the solutions of a given system of non-linear equations. Buchberger also provided an algorithm for computing Gröbner bases.

Let F be a field, $R = F[x_1, \ldots, x_n]$ a polynomial ring in n variables over F, and $f_1, \ldots, f_s \in R$. We say that the polynomials $f_1, \ldots, f_s$ form a *basis* of the ideal $\langle f_1, \ldots, f_s\rangle = \{q_1 f_1 + \ldots + q_s f_s \mid q_i \in R, i = 1, \ldots, s\}$. The Gröbner basis is a special basis for $\langle f_1, \ldots, f_s\rangle$ which gives solutions to, among other problems, the *ideal membership problem*, i.e. given $f \in R$, is $f \in \langle f_1, \ldots, f_s\rangle$? It is easy to see that $f = 0$ whenever $f_1 = \ldots = f_s = 0$, for any $f \in \langle f_1, \ldots, f_s\rangle$. Thus, the problem of checking whether $f = 0$ is entailed by a set of equations $f_1 = 0, \ldots,$

Algorithm 1 Buchberger algorithm

Require: $f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$, and an admissible order $\prec$
Ensure: A Gröbner basis $G \subseteq R$ for the ideal $I = \langle f_1, \ldots, f_s \rangle$ with respect to $\prec$, with $f_1, \ldots, f_s \in G$

```
1: G ← {f_1, ..., f_s}
2: repeat
3:    S ← ∅
4:    order the elements of G somehow as g_1, ..., g_t
5:    for 1 ≤ i < j ≤ t do
6:       r ← S(g_i, g_j) rem(g_1, ..., g_t)
7:       if r ≠ 0 then S ← S ∪ {r}
8:    end for
9:    G ← G ∪ {S}
10: until S = ∅
11: return G
```

$f_s = 0$ can be reduced to determining whether $f \in \langle f_1, \ldots, f_s \rangle$ which in turn can be reduced to checking whether $f \operatorname{rem} G = 0$, where G is the Gröbner basis for $\langle f_1, \ldots, f_s \rangle$.

The construction of Gröbner bases is based on two main operations:

1. the reduction of a polynomial by another polynomial, and
2. the addition in the basis of new polynomials from the ideal.

The first operation assumes that the monomials of the polynomials are sorted decreasingly according to an admissible term ordering, like lexicographic ordering [Kamin and Lévy, 1980]. A reduction is performed if the first polynomial has terms that are multiple of the highest (leading) term of the second polynomial. In this case, we subtract from the first polynomial the appropriate multiple of the second polynomial. The second operation, called *S-polynomial* construction, is also based on two polynomials. The smallest term which is a multiple of the leading terms of both of them is used to build the terms with which each polynomial is multiplied. The polynomial to be added is obtained by subtracting them.

We denote by $S(f, g)$ the *S-polynomial* built from the polynomials f and g, and by $f \operatorname{rem}(g_1, \ldots, g_s)$ the multivariate division with remainder algorithm for polynomials of f with $(g_1, \ldots, g_s)$, such that $f = q_1 g_1 + \cdots + q_s g_s + (f \operatorname{rem}(g_1, \ldots, g_s))$. Algorithm 1 computes such Gröbner bases [von zur Gathen and Gerhard, 1999]. More details about Gröbner bases can be found in [Buchberger and Winkler, 1998].

Example 7.1 ([von zur Gathen and Gerhard, 1999]): *Let us consider the problem of showing that*

$$\underbrace{3v - y}_{\gamma} = 0 \tag{23}$$

is entailed by

$$\overbrace{uy - v(x+1)}^{\varphi} = 0 \tag{24}$$
$$\underbrace{(u-1)y - v(x-2)}_{\psi} = 0 \tag{25}$$

To this end we first compute the Gröbner basis G of the ideal $I = \langle \varphi, \psi \rangle$ and then we check whether $g \operatorname{rem} G = 0$. We start by computing a Gröbner basis for I with respect to a lexicographic order over the variables, $\prec$, such that $u \succ v \succ x \succ y$. The normal forms of φ and ψ are $\varphi' = uy - vx - v$ and $\psi' = uy - vx + 2v - y$ respectively. By applying Algorithm 1 step by step, we obtain $S(\varphi', \psi') = \varphi' - \psi' = -3v + y$ and $-\gamma \operatorname{rem}(\varphi', \psi') = -\gamma$. Thus we add g to the basis and after a new iteration, we get:

$$S(\varphi', g) \operatorname{rem}(\varphi', \psi', g) = -uy^2 - 3v^2 x - 3v^2 \operatorname{rem}(\varphi', \psi', g) = 0$$
$$S(\varphi', \psi') \operatorname{rem}(\varphi', \psi', g) = -uy^2 - 3v^2 x + 6v^2 - 3vy \operatorname{rem}(\varphi', \psi', g) = 0$$

and therefore $\{\varphi', \psi', \gamma\}$ is the required Gröbner basis G. It is immediate to see that $\gamma \operatorname{rem} G = 0$ and from this we can thus conclude that (23) *is entailed by* (24) *and* (25).

△

The reasoning specialist of Section 6 can be readily extended so to incorporate Buchberger algorithm. To this end, it suffices to add a extra field to the constraint store, which thus become of the form $\langle U \mid G \parallel I \parallel N \rangle$ where U, G, and I have the same meaning as in Section 6 and N is a set of equations containing non-linearities and represents the state of Buchberger algorithm. The rules specifying the inner workings of the reasoning specialist are adapted from the corresponding ones in Section 6 so that the manipulation of the fields U, G, and I are preserved while the new field, N, is manipulated by Buchberger algorithm.

$$\textsc{InitN}: \quad \langle U \mid G \parallel I \parallel N \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \parallel G \cup P_G \parallel I \cup P_I \parallel N \cup P_G \rangle$$

where P_G and P_I are as in Section 6.

$$\textsc{CongrN}: \quad \langle U \mid G \parallel I \mid N \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \cup E \parallel \mathit{congr}(G) \mid I \mid N \rangle$$
$$\text{where } E = \{a \to b : a = b \in \mathit{congr}(G) \text{ and } b \prec a\}$$

$$\textsc{ArithN}: \quad \langle U \mid G \parallel I \mid N \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \parallel G \cup E \parallel I \cup I' \mid N \rangle$$
$$\text{where } (I', E) = \mathit{arith}(I)$$

NL-ARITH: $\langle U \parallel G \parallel I \parallel N \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \mid G \cup N' \mid I \mid N \cup N' \rangle$
where N' is the set of the equations obtained
by applying Buchberger algorithm to N

NORMALIZEN: $\langle U \parallel G \mid I \parallel N \rangle \xrightarrow[P]{\text{cs-simp}} \langle U \mid G' \parallel I' \parallel N' \rangle$
where I', G', and N' are the results of
normalizing I, G, and N (resp.) with the rules in U

EXAMPLE 7.2: *Let us consider the problem of proving the following two conjectures:*

$$max(a,b) = b \Rightarrow min(a,b) = a \tag{26}$$

$$max(u,t) = t \wedge \overbrace{min(u,t)y - v(x+1)}^{\xi} = 0 \wedge \overbrace{uy - vx + 2v - y}^{\psi'} = 0 \Rightarrow \underbrace{3v - y}_{\gamma} = 0 \tag{27}$$

where min and max are defined by:

$$\begin{array}{lcl} X \geq Y = \text{T} & \Rightarrow & min(X,Y) = Y \\ X \geq Y = \text{F} & \Rightarrow & min(X,Y) = X \end{array}$$

$$\begin{array}{lcl} X \geq Y = \text{T} & \Rightarrow & max(X,Y) = X \\ X \geq Y = \text{F} & \Rightarrow & max(X,Y) = Y \end{array}$$

A direct application of Buchberger algorithm to (26) *is not successful. It is easy to show that γ is not in the ideal of the Gröbner basis of $\langle \xi, \psi' \rangle$.*

New information about the interpreted symbols min and max is provided by (26)*. Its instance, namely $max(u,t) = t \Rightarrow min(u,t) = u$, can be used as induction hypothesis since it is smaller than* (27) *w.r.t. $\prec\!\!\!\prec$, where $\prec\!\!\!\prec$ is a multiset extension of a* rpo *(Recursive Path Ordering Dershowitz and Jouannaud [1990a]) using the following precedence over the function symbols: $min \succ max \succ \geq \succ * \succ + \succ - \succ s \succ 0$. This instance can be considered as a conditional rewrite rule which transforms $min(u,t)$ in u in the first conjecture because its condition is discarded from the context of the first conjecture. Therefore, ξ is rewritten to φ (cf.* (24) *in Example 7.1) and application of Buchberger algorithm allows us to conclude.*

The other conjecture, i.e. (26)*, is readily eliminated by a simple case analysis using the rewrite rules defining min and max.* △

8. Conclusions and Future Work

We have presented a general schema for the integration of reasoning specialists with an implicit induction prover. The integration scheme is effective since when

applied with SPIKE and decision procedures for equality and arithmetics it has given positive results on several non-trivial problems. Moreover, the soundness of our integration has been formally derived. This task is not trivial since we allow some interleaving between induction hypothesis application and decision-procedure application. The reasoning specialist for the union of UTE and UPAI can be easily extended to deal with wider fragments of arithmetics which non-linear equations.

We plan to apply the integration scheme to new decision procedures for new decidable theories such as those for lists and arrays [Armando et al., 2001]. At the same time, we will apply the theorem-prover to the verification of protocols like ABR [Rusinowitch et al., 2000]. We would like also to exploit better the efficiency built-in AC operators by using AC-matching as in [Berregeb et al., 1996].

References

A. Armando and S. Ranise. Constraint Contextual Rewriting. In Ricardo Caferra and Gernot Salzer, editors, *Proc. of the 2nd Intl. Workshop on First Order Theorem Proving (FTP'98). Vienna, Austria, November 23-25, 1998*, pages 65–75, 1998.

A. Armando and S. Ranise. Termination of Constraint Contextual Rewriting. In *Proceedings of 3rd International Workshop on Frontiers of Combining Systems (FroCoS'2000)*, Lecture Notes in Artificial Intelligence 1794, pages 47–61, Nancy, France, March 2000.

A. Armando, S. Ranise, and M. Rusinowitch. Uniform derivation of decision procedures by superposition. In *In the Proceedings on the Annual Conference on Computer Science Logic (CSL01)*, number 2142 in Lecture Notes in Computer Science, pages 513–527. Springer-Verlag, September 2001.

N. Berregeb, A. Bouhoula, and M. Rusinowitch. Automated verification by induction with associative-commutative operators. In *Proceedings 8th International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 220–231, New Jersey, USA, 1996.

N. Bjørner. Private communication, 1998.

N. Bjørner et al. *STeP. The Stanford Temporal Prover*, November 1995. Version 1.0.

A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–78, January 1997. ISSN 0747-7171.

A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, April 1995.

R. S. Boyer and J S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. ICSCA-CMP-44, University of Texas at Austin, 1985. Also published in *Machine Intelligence 11*, Oxford University Press, 1988.

R. S. Boyer and J S. Moore. MJRTY - a fast majority vote algorithm. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, volume 1 of *Automated Reasoning*, pages 105–117. Kluwer Academic Publishers, 1991.

B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Philosophische Fakultät an der Leopold-Franzens-Universität, Insbruck, Austria, 1965.

B. Buchberger and F. Winkler, editors. *Gröbner Bases and Applications.* Cambridge University Press, 1998.

N. Dershowitz and J.P. Jouannaud. *Rewrite systems*, volume B:Formal Methods and Semantics of *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. North-Holland, Amsterdam, 1990a.

N. Dershowitz and J.P. Jouannaud. Rewriting systems. In *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier Publishers, Amsterdam, 1990b.

D. J. Howe. Reasoning about functional programs in Nuprl. In *Functional Programming, Concurrency, Simulation and Reasoning*, number 693 in Lecture Notes in Computer Science, pages 145–164, 1993.

G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, Laboria, France, 1978.

P. B. Jackson. *The Nuprl Proof Development System. Version 4.1 Reference Manual and User's Guide.* Cornell University, 1994.

J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished note, Department of computer Science, University of Illinois, Urbana, IL, 1980.

J.-L. Lassez and M.J. Maher. On Fourier's Algorithm's for Linear Arithmetic Constraints. *Journal of Automated Reasoning*, 9:373–379, 1992.

D. Naidich. On generic representation of implicit induction procedures. Technical Report CS-R9620, CWI, 1996.

G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford Computer Science Department, April 1978.

S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer Verlag, 1992.

M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental ABR conformance algorithm. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of 12th International Conference on Computer Aided Verification (CAV'2000)*, number 1855 in Lecture Notes in Computer Science, pages 344–357. Springer Verlag, July 2000.

S. Stratulat. Applying semantic subsumption rules in the context of inductive proofs. In *Workshop on Integration of Deductive Systems, CADE-15*, pages 85–95, 1998.

S. Stratulat. *Preuves par récurrence avec ensembles couvrants contextuels. Applications à la vérification de logiciels de télécommunications.* PhD thesis, Université Henri Poincaré, Nancy I, 2000.

S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32(4):403–445, 2001.

J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.

H. Zhang. Contextual Rewriting in Automated Reasoning. *Fundamenta Informaticae*, 24(1/2):107–123, 1995.

H. Zhang and J. L. Remy. Contextual rewriting. In Jean-Pierre Jouannaud, editor, *Proc. of the 1st International Conference on Rewriting Techniques and Applications*, volume 202 of *LNCS*, pages 46–62, Dijon, France, May 1985. Springer. ISBN 3-540-15976-2.