Implementing Reasoning Modules in Implicit Induction Theorem Provers

Sorin Stratulat

Implementing Reasoning Modules in Implicit Induction Theorem Provers

Sorin Stratulat
LITA, Department of Computer Science
Université de Lorraine
Ile du Saulcy, Metz, 57000, FRANCE
Email: sorin.stratulat@univ-lorraine.fr

***Abstract*—We detail the integration in SPIKE, an implicit induction theorem prover, of two reasoning modules operating over naturals combined with interpreted symbols. The first integration schema is *à la* Boyer-Moore, based on the combination of a congruence closure procedure with a decision procedure for linear arithmetic over rationals/reals. The second follows a 'black-box' approach and is based on external SMT solvers. It is shown that the two extensions significantly increase the power of SPIKE; their performances are compared when proving a non-trivial application.**

I. Introduction

In [1] it has been shown that concrete induction-based theorem provers can be built by implementing *abstract* rules with reasoning modules. The abstract rules express the declarative part of the reasoning and can be considered as the logical components of the provers since they define what information can be soundly used during proof derivations, for example the induction hypotheses. On the other hand, the reasoning modules represent their operational components that define how new formulas can be derived from existing ones, based on the information provided by the abstract rules.

In this paper, we discuss two integration schemas of reasoning modules in SPIKE [1]–[3], an automated first-order theorem prover based on implicit induction. The first follows ideas from [4]. It allows for a tight cooperation between the prover and the reasoning modules. Compared with the previous schema, the reasoning modules can provide in addition new information in terms of logical consequences of the given input, which can be further used by the prover in subsequent steps of the proof.

The second integration schema follows a 'black box' approach such that the conjectures to be proved, together with other information (axioms, induction hypotheses) useful for proving the conjecture, are processed by an external tool, in our case an SMT solver. The answers provided by the tool back to the prover are rather limited; they can inform SPIKE only if the conjecture is true, false or has an unknown truth value.

The rest of the paper has 3 sections. Section II introduces the backgrounds of reasoning by implicit induction and describes the SPIKE prover. Section III details two extensions of SPIKE, following the above integration schemas, with reasoning modules operating over naturals combined with interpreted symbols, then compare the performances of their implementations when tested on a non-trivial application. The last section concludes.

II. The SPIKE prover

In the following, we present the syntax and semantics of SPIKE specifications, as well as the underlying induction principle and the SPIKE's inference system.

Syntax. SPIKE can reason on many-sorted conditional specifications built from quantifier-free conditional equalities, denoted by *axioms*, that *define* function symbols. In addition, a (disjoint) set of *constructor* function symbols is attached to each sort. We denote by $\mathcal{F}$ the disjoint union between the sets of constructor and defined function symbols, and by $\mathcal{V}$ a denumerable set of variables. Each variable from $\mathcal{V}$ has a sort and the profile of each function f from $\mathcal{F}$ is sorted, of the form $f : s_1 \times \ldots \times s_n \to s$, where $s_1, \ldots, s_n, s$ are sorts. In this case, we say that f has the sort s. The set of *terms* built from function symbols from $\mathcal{F}$ and variables from $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. If S is a denumerable set of sorts, by $\mathcal{T}(\mathcal{F}, \mathcal{V})_{s \in S}$ we can recursively define the set of terms of sort s, which can be either a variable of sort s or a non-variable term of the form $f(t_1, \ldots, t_n)$, where f has the profile $f : s_1 \times \ldots \times s_n \to s$ and t_1 (resp., $t_2, \ldots t_n$) is in $\mathcal{T}(\mathcal{F}, \mathcal{V})_{s_1 \in S}$ (resp., $\mathcal{T}(\mathcal{F}, \mathcal{V})_{s_2 \in S}$, …, $\mathcal{T}(\mathcal{F}, \mathcal{V})_{s_n \in S}$). Therefore, $\mathcal{T}(\mathcal{F}, \mathcal{V})$ can be defined as $\sqcup \mathcal{T}(\mathcal{F}, \mathcal{V})_{s \in S}$, i.e., the disjoint union of the sets of terms of same sort, by considering all the sorts occurring in S.

An (unconditional) *equality* is a binary relation between two terms l and r of same sort, denoted by $l = r$. A *conditional* equality is represented under the form of the implication $l_1 = r_1 \wedge \ldots \wedge l_n = r_n \Rightarrow l = r$, where the conclusion $l = r$ and each $l_i = r_i$ ($i \in [1..n]$) from the condition part are unconditional equalities. SPIKE accepts specifications built on *free* constructors, for which there is no equality relation between any two different constructor symbols. A term or equality is *ground* if it has no variables. The equality $l_1 = r_1 \wedge \ldots \wedge l_n = r_n \Rightarrow l = r$ can be oriented into the *rewrite rule* $l_1 = r_1 \wedge \ldots \wedge l_n = r_n \Rightarrow l \to r$ if l is greater than r (w.r.t. an ordering over terms) as well as any l_i and r_i, for any $i \in [1..n]$. In SPIKE, the ordering over terms is the recursive path ordering (rpo) with status [5], denoted in the following by $<_t$.

New terms and equalities can be built from replacing (subsets of their) variables with terms of the same sort, by the means of substitutions. A *substitution* is a mapping $\sqcup\{x_i \mapsto t_i\}$, where the variable x_i and the term t_i have the same sort. It is ground if each replacing term is ground. If σ is a substitution and t a term or equality, then $t\sigma$ is an *instance* of t.

Semantics. Let Ax be the set of axioms of a given SPIKE specification and $\mathcal{M}$ a set of Herbrand models of Ax. An (unconditional or conditional) equality is a $\mathcal{M}$-consequence (or just *consequence*) of a set of equalities Φ, denoted by $\Phi \models_{\mathcal{M}} \phi$, if ϕ is valid in the model m whenever ψ is valid in m, for any $\psi \in \Phi$ and $m \in \mathcal{M}$. An equality ϕ is $\mathcal{M}$-valid (or just *valid*), denoted by $\models_{\mathcal{M}} \phi$, iff it is a consequence of Ax.

In the following, we consider M as being the singleton built from the *initial model* of Ax. The consequence relation is referred to as *inductive*. An equality is a *counterexample* if there is a ground instance of it which is not valid. We say that an equality *has a counterexample* if there is a ground instance of it which is a counterexample. A set of equalities has a counterexample if there is an equality from the set that has a counterexample.

The induction principle. SPIKE implements an instance of the Noetherian induction principle. Given $(\mathcal{E}, <)$ a non-empty well-founded poset of equalities to prove, the formula-based instance of the Noetherian induction principle [6] states that if, for any equality $\delta \in \mathcal{E}, \bigcup_{\gamma \in \mathcal{E}, \gamma < \delta} \{\gamma\} \models_{\mathcal{M}} \delta$ then $\forall \rho \in \mathcal{E}$, $\models_{\mathcal{M}} \rho$. The principle allows that, during the proof of an equality conjecture, smaller formulas can be used in terms of induction hypotheses (IHs), which makes it a natural choice for reasoning on proofs requiring lazy and mutual induction steps. The proof method used by SPIKE is an application of this principle, called *implicit induction*, suggested in [7] and formally presented in [8]. In SPIKE, $<$ is a multiset extension of $<_t$.

The inference system. The inference system is a set of inference rules representing transitions between pairs of sets of equalities of the form $(E, H) \vdash (E', H')$, where E, E' are two sets of *conjectures* and H, H' are two sets of *premises*. The application of such a rule replaces a (current) conjecture ϕ with a potentially empty set of new conjectures Φ. This can be summarized as $(E \cup \{\phi\}, H) \vdash (E \cup \Phi, H')$. Moreover, ϕ may be added as a premise in order to participate to further inference steps. Therefore, H' can be either H or $H \cup \{\phi\}$. An implicit induction *derivation* for a set of equalities E_0 is any sequence of the form $(E_0, H_0) \vdash \cdots \vdash (E_n, H_n) \vdash \ldots$ resulted from successive applications of inference rules starting from (E_0, H_0). A *proof* is a finite derivation $(E_0, H_0) \vdash \cdots \vdash (E_n, H_n)$ for which H_0 and E_n are empty sets.

The induction principle can be applied to validate a proof $(E_0, \emptyset) \vdash \cdots \vdash (\emptyset, H_n)$ by considering $\mathcal{E}$ as the set $\{\phi\sigma \mid \phi\sigma$ is ground and $\phi \in \bigcup_i E_i\}$ and the inference system as *reductive*, i.e., for any ground instance $\phi\tau$ of the current conjecture ϕ from any step $(E \cup \{\phi\}, H) \vdash (E \cup \Phi, H')$ of the proof, either i) $\phi\tau$ is valid, or ii) $\phi\tau$ is a counterexample and there is a smaller or equal counterexample in $E \cup H \cup \Phi$. In addition, we assume that any premise does not have minimal counterexamples in $\mathcal{E}$. It can be easily noticed that any reductive system is *sound*, i.e., $\models_{\mathcal{M}} E_0$. Otherwise, if E_0 is not valid, $\mathcal{E}$ should have a counterexample. Since $<$ is well-founded, there is a minimal counterexample τ in $\mathcal{E}$. The proof ends with an empty set of conjectures, therefore there is a *last* proof step where ϕ is the current conjecture, of the form $(E \cup \{\phi\}, H) \vdash (E \cup \Phi, H')$, and ϕ has a counterexample equal to τ. On the other hand, the inference system is reductive, so there is a counterexample smaller than τ in $E \cup \Phi$, as $E \cup \Phi \cup H$ does not have counterexamples equal to τ, hence in $\mathcal{E}$. Contradiction.

Concrete reductive inference systems, for which the premises do not have minimal counterexamples, can be built using a methodology based on *contextual cover sets* (CCSs) [1]. We denote by the *context* $\mathcal{C}$ the pair of sets of equalities $(\mathcal{C}^1, \mathcal{C}^2)$. We say that the set of equalities Ψ *contextually covers* the set of equalities Φ in $\mathcal{C}$, denoted by $\Phi \sqsubseteq_{\mathcal{C}} \Psi$, iff $\mathcal{C}^1_{\leq \phi\sigma} \cup \mathcal{C}^2_{<\phi\sigma} \cup \Psi^1_{\leq \phi\sigma} \models_{\mathcal{M}} \phi\sigma$, for any $\phi \in \Phi$ and ground instance $\phi\sigma$. When $\Phi = \{\phi\}$, we say that Ψ is a CCS of ϕ. When Φ is empty, the CCS is *empty*, and when $\Psi^1_{\leq \phi\sigma}$ is replaced in the definition by $\Psi^1_{<\phi\sigma}$, then the CCS is *strict* and denoted by $\Phi \sqsubset_{\mathcal{C}} \Psi$.

The core of the methodology is an abstract inference system made of two rules: ADDPREMISE and SIMPLIFY, as shown in Fig. 1.

1-ADDPREMISE : $(E \cup \{\phi\},\ H) \vdash (E \cup \Phi,\ H \cup \{\phi\})$
if $\{\phi\} \sqsubset_{(H,\ E)} \Phi$

1-SIMPLIFY: $(E \cup \{\phi\},\ H) \vdash (E \cup \Phi,\ H)$
if $\{\phi\} \sqsubseteq_{(E \cup H,\ \emptyset)} \Phi$

Fig. 1: Abstract inference rules.

Each rule defines the set of new conjectures Φ as a CCS of the current conjecture ϕ, by explicitly stating the content of the context. It can be noticed that the current conjecture of ADDPREMISE does not have minimal counterexamples, so the rule adds it as a premise. SIMPLIFY does not satisfy this property but allows bigger contexts. A very useful instance of SIMPLIFY is when Φ is empty, also referred to as the DELETE rule. It can be noticed that more complex abstract inference rules can be built. In [1], it was shown that abstract inference rules can build in two steps the CCS of the current conjecture, thanks to the compositional properties of CCSs and that they define the biggest contexts compared to similar abstract rules proposed in the literature.

According to the methodology, any concrete inference rule is built as an instance of one of the abstract rules by implementing its elementary CCSs, i.e. the CCSs that are not built with composition operations, by the means of *reasoning modules*. A reasoning module can produce a CCS with a particular reasoning technique using as IHs formulas from the context defined by the instantiated abstract rule. A reasoning module M can be characterised by a function f_M defined as $f_M(\phi, \mathcal{C}) = \Phi$, where Φ is a CCS of ϕ in the context $\mathcal{C}$. The main reasoning techniques used by SPIKE are based on rewriting, case analysis and variable instantiations. Since any proof ends with an empty set of conjectures, a class of interest is represented by the reasoning modules that build empty CCSs.

III. IMPLEMENTING REASONING MODULES FOR NATURALS COMBINED WITH INTERPRETED SYMBOLS

We will detail the implementation of two reasoning modules for naturals combined with interpreted symbols using: i) a tight integration schema *à la* Boyer-Moore [4] with home-made components, and ii) a 'black-box' approach based on an

external SMT solver. Both reasoning modules take as input an equality, as well as information from the context of the CCS defined by implemented abstract rule, negate the equality and validate it if an inconsistency is detected. We further point out the major role they played during the validation of the conformity algorithm for a telecommunications protocol.

A. *Integrating the reasoning module* à la *Boyer-Moore*

The reasoning module is built from the cooperation between a decision procedure for the quantifier-free theory of equality, T_{ge}, and a decision procedure for linear arithmetic over rationals/reals, T_{la}. It served as a case study to validate an approach integrating decision procedures in a proof environment mixing induction and rewriting techniques [9]. The equality given as input is valid if, after its negation, one of the decision procedures returns **inconsistent**.

The components. The cooperation schema is defined by transitions between states, called $\overline{C}$-structures. A $\overline{C}$-structure consists of constraint stores containing equalities, rewrite rules and linear inequalities issued from the negation of the equality given as input.

Definition 1 ($\overline{C}$-structure): Let C be a conditional equality. A $\overline{C}$-structure is a data structure consisting of the quadruple $\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \parallel \mathsf{L} \rangle$, where

- CR is built from unconditional rewrite rules;
- A consists of atomic formulas issued from the negation of C;
- G contains unconditional equalities and disequalitites;
- L is the pair $(\mathsf{P} \bullet \mathsf{IE})$, where P has linear inequalities and IE stores the (implicit) equalities issued from P after applying the decision procedure for T_{la}.

The variables from the negation of C are existentially quantified and will not be instantiated during the cooperation process but treated as constants. The terms (resp. equalities) from a constraint store can be considered as ground terms (resp. ground equalities) defined over the initial set of function symbols extended with these constants.

Example 1: Let C be the conjecture from Example 5.2 of [10] (also presented in [11]):

$$(p(x) = True \ \wedge \ z \leq f(max(x,\ y)) = True \ \wedge \\ 0 < min(x,\ y) = True \ \wedge x \leq max(x,\ y) = True \ \wedge \\ max(x,\ y) \leq x = True) \ \Rightarrow \ z < g(x) + y = True.$$

The $\overline{C}$-structure is initialized by

$$\begin{array}{lcl} \langle\ \mathsf{CR} & : & \emptyset \\ \quad \mathsf{A} & : & \{p(x) = True,\ z \leq f(max(x,\ y)) = True,\ 0 < \\ & & min(x,\ y) = True, x \leq max(x,\ y) = \\ & & True,\ max(x,\ y) \leq x = True,\ z < g(x) + y = \\ & & False\} \\ \quad \mathsf{G} & : & \emptyset \\ \quad \mathsf{L} & : & (\emptyset \bullet \emptyset)\rangle \end{array}$$

The cooperation schema allows to normalize monomials from P and equalities from IE with rewrite rules from CR. We denote by linearize the procedure to transform an atomic formula into a conjunction of linear inequalities. It returns the empty set if no transformation has been operated on the input; in this case, we say that the input is *non-linearizable*, otherwise it is *linearizable*. The decision procedure for T_{la} will be applied on the linear inequalities from P based on the Fourier-Motzkin variable elimination method [12] over rationals and implemented by the function arith. It returns **inconsistent** if an unsatisfiable linear inequality (for example, $1 \leq 0$) is generated. A set of *implicit* equalities is returned if the inequality $0 \leq 0$ is produced by the method. The decision procedure is *incomplete* because a consistency reported with an interpretation over rationals can be an inconsistency with an interpretation over naturals. In spite of this limitation and its doubly exponential complexity, it has been chosen for its simple implementation. The disjunction operations, mainly resulted when processing disequalities, can be avoided as explained in the following. It can be noticed that the negation of the equality $l_1 = r_1 \wedge \ldots \wedge l_n = r_n \Rightarrow l = r$ can be represented as the conjunction $l_1 = r_1 \wedge \ldots \wedge l_n = r_n \wedge l \neq r$. For the case when the linearization of $l \neq r$ yields the disjunction $l - r + 1 \leq 0 \ \vee \ r - l + 1 \leq 0$, the disequality $l \neq r$ is not processed (but can be processed by the decision procedure for T_{ge}). This is another source of incompleteness but guarantees that there is always only one set of conjunctions of monomials on which the Fourier-Motzkin method operates. An example where the disjunctive representation can be avoided is when one of the sides of the disequality is a boolean value ($True$ or $False$), for example $f(x) \leq g(x) \neq False$ can be linearized to $f(x) - g(x) \leq 0$.

The monomials from a linear inequality can be ordered by a *total* ordering, similar to [13] and defined as follows.

Definition 2 (total ordering over terms): Let t_1 and t_2 be two terms.

We write $t_1 <_{gt} t_2$ iff

- the number of variable positions in t_1 is smaller than that for t_2, or
- the number of variable positions in both terms is the same, but the number of strict positions in t_1 is smaller than that for t_2, or
- the number of variable and strict positions in both terms is the same, but t_1 is smaller than t_2, according to some lexicographic ordering [5, p. 19].

The function symbols and variables from the above definition are uniquely interpreted as naturals by using their internal representation into SPIKE. On the other hand, $<_{gt}$ does not need to be well-founded, hence it may be different from $<_t$.

The input for the decision procedure for T_{ge}, denoted by congr, is a finite set of equalities and disequalities. The equalities are oriented into rewrite rules using the $<_{gt}$ ordering. New equalities are produced by a rewrite-based ground completion algorithm, similar to [14], and the new rewrite system helps to normalize the members of the disequalities. The output of congr is either i) **inconsistent**, if a trivial disequality of the form $s \neq s$ is derived, or ii) the new set of equalities and disequalities.

The cooperation schema. We show in the dotted box from Fig. 2 the transitions between the components of a $\overline{C}$-structure,

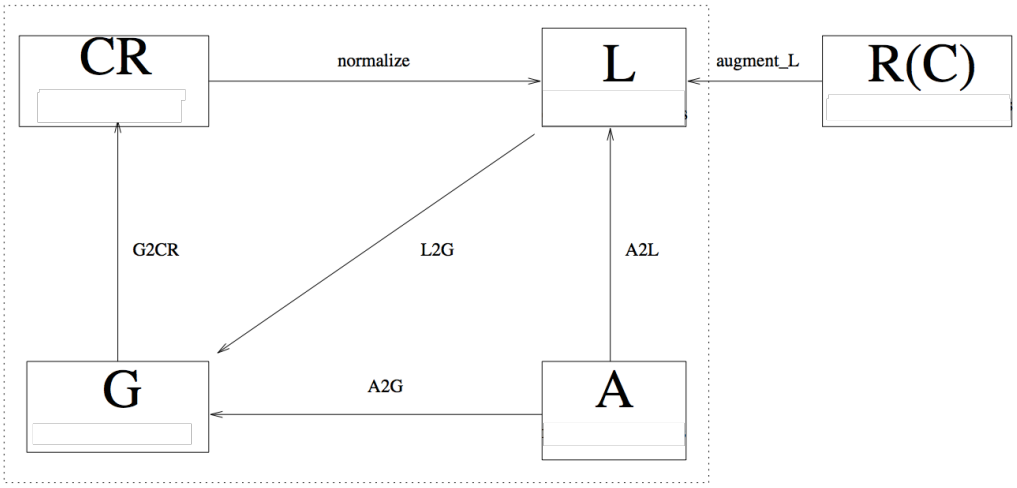

Fig. 2: The cooperation schema.

and from the proof environment to the reasoning module. The definition of each transition is given in Fig. 3.

The atomic formulas from A will be dispatched between L and G, using the first two transitions, A2L and A2G, such that any atomic formula that cannot be transformed into a conjunction (seen as a set) of linear inequalities will be sent to G, otherwise to P. The transitions initialize respectively the P and G components with linear inequalities and equalities/disequalities resulted from the atomic formulas from A. The transition A2L applies in addition arith on the new set of linear inequalities. Any new rewrite rule, resulted from the application of congr on G, are transfered to CR by G2CR. The L2G rule transfers to G all equations from IE. The monomials of linear equalities from P and the equations from IE are normalized by rules from CR, using the normalize procedure.

The proof environment interacts with the reasoning module by the means of the *augmentation operation*, inspired from [4]. Defined in Fig. 4 and denoted by Oracle_A, it is based on the R(C) set of formulas which may contain (instances of) conditional equations issued from axioms, lemmas, or the context of the CCS given as input to the reasoning module. Oracle_A firstly assumes that R(C) has an instance of a conditional equality ϕ whose conclusion is linearizable and, after the linearization process, i) the set of inequalities has a maximal monomial of an inequality from P but with a coefficient with opposite sign, and ii) no new monomials are introduced in P. If the conditions of ϕ are consequences of R(C) and the conditions of C, denoted by $cond(C)$, the operation Oracle_A returns the set of equalities from the condition part of ϕ. The last rule from Fig. 3, augment_L, augments the set of linear inequalities by using rewrite rules from R(C); it applies afterwards arith on the new set of inequalities.

The strategy for applying the above transitions can be implemented with a list of transition labels, initialized by [A2G, A2L]. The first element from the list determines the transition to be executed. When the current transition finished to be executed, its name is replaced with the names of the transitions displayed by its **Next** field from Fig 3. The cooperation schema *succeeds* if **inconsistent** is returned by arith or congr procedures. The cooperation schema *failed* if the list becomes empty.

Properties. The cooperation schema of the reasoning module is sound and well-founded.

Theorem 1 (soundness): Let ϕ be a conditional equality and $\mathcal{C}$ the context given as input to the reasoning module. Then, the empty set is an empty CCS of ϕ in C whenever arith or congr return **inconsistent** during the execution of the cooperation schema.[1]

Theorem 2 (termination): The cooperation schema is well-founded.

Proof: Let us assume that there is a chain of $\overline{C}$-structure transformations $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n \rightarrow \ldots$. It can be noticed that the execution graph of the rules from Fig. 2 has two minimal cycles: (A2L, L2G, G2CR, normalize) and (A2L, augment_L). Since i) the set of monomials from P can be modified only by rewriting with rewrite rules from CR, and ii) $<_t$ is well-founded, there is a step j in the chain such that normalize is no longer applied in further steps to modify the set of monomials from P. This breaks the cycle (A2L, L2G, G2CR, normalize). If no new monomials are produced, Oracle_A will return an empty set of equalities, hence breaking the cycle (A2L, augment_L). ■

The reasoning module is an incomplete decision procedure because of the incompleteness of the decision procedure for T_{la}.

A detailed example. Let C be the conjecture from Example 1. We will show how the reasoning module can help producing an inconsistency from its negation, assuming that R(C) consists of the set of two equalities $\{max(x',\ y') = x' \Rightarrow min(x',\ y') = y',\ p(u) = True \Rightarrow f(u) \leq g(u) = True\}$, and that there is no additional knowledge on f and g. The initial $\overline{C}$-structure is (the empty fields are not listed in the following):

[1]For lack of space, the detailed proof can be found in [15].

$\langle \mathsf{CR} \mid \mathsf{A} \cup \mathcal{E} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle \xrightarrow{\mathsf{A2L}} \langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P}' \bullet \mathsf{IE} \cup \mathsf{IE}') \rangle$

where
1) $\mathcal{E}$ is a maximal set such that $\forall e \in \mathcal{E}$, e is linearizable, and
2) $(\mathsf{IE}',\ \mathsf{P}') = \mathsf{arith}(\{\cup_{e' \in \mathcal{E}} \mathsf{linearize}(e')\} \cup P)$

Next:
if $\mathcal{E} = \emptyset$ then [] else (if $IE' = \emptyset$ then [augment_L] else [L2G])

$\langle \mathsf{CR} \mid \mathsf{A} \cup \mathcal{E} \mid \mathsf{G} \mid \mathsf{L} \rangle \xrightarrow{\mathsf{A2G}} \langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \cup \mathcal{E} \mid \mathsf{L} \rangle$

where $\mathcal{E}$ is a maximal set such that $a = b$ is non-linearizable, $\forall a = b \in \mathcal{E}$

Next: [G2CR]

$\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid \mathsf{L} \rangle \xrightarrow{\mathsf{G2CR}} \langle \mathsf{CR} \cup \mathcal{E} \mid \mathsf{A} \mid \mathsf{congr}(\mathsf{G}) \mid \mathsf{L} \rangle$

where $\forall a \to b \in \mathcal{E}$, $a = b \in \mathsf{congr}(\mathsf{G})$ and $b <_t a$

Next: if $\mathcal{E}$ is empty then [] else [normalize]

$\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle \xrightarrow{\mathsf{L2G}} \langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \cup \mathsf{IE} \mid (\mathsf{P} \bullet \emptyset) \rangle$

Next: [G2CR]

$\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle \xrightarrow{\mathsf{normalize}} \langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G}' \mid (\mathsf{P}' \bullet \mathsf{IE}') \rangle$

where
1) P' is P normalized with rules from CR, and
2) IE' is IE normalized with rules from CR

Next: [A2L]

$\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle \xrightarrow{\mathsf{augment\_L}} \langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P}' \bullet (\mathsf{IE} \cup \mathsf{IE}')) \rangle$

where
1) $\mathcal{E} = \mathsf{Oracle\_A}(\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle)$, and
2) $(\mathsf{P}',\ \mathsf{IE}') = \mathsf{arith}(\{\cup_{e \in \mathcal{E}} \mathsf{linearize}(e)\} \cup \mathsf{P})$

Next: if $\mathcal{E} = \emptyset$ then [] else (if $\mathsf{IE}' = \emptyset$ then [A2L] else [L2G])

Fig. 3: The transition rules for the cooperation schema.

$\mathsf{Oracle\_A}(\langle \mathsf{CR} \mid \mathsf{A} \mid \mathsf{G} \mid (\mathsf{P} \bullet \mathsf{IE}) \rangle)$,

where
1) $p_1 \wedge \ldots \wedge p_n \Rightarrow l = r$ is an instance of a conditional equality from $\mathsf{R}(\mathsf{C})$,
2) u is a monomial of $\mathsf{linearize}(l = r)$ having c, as coefficient,
3) the rest of the monomials of $\mathsf{linearize}(l = r)$ are in the $\overline{C}$-structure,
4) u is a maximal monomial of an inequality of de P whose coefficient is c' such that $c * c' < 0$, and
5) $R(C) \cup cond(C) \models_{\mathcal{M}} p_i$, $\forall i \in [1..n]$

Returns: l=r

Fig. 4: The Oracle_A augmentation operation.

$$\langle\ \mathsf{A} \ :\ \{p(x) = True,\ z \leq f(max(x,\ y)) = True,\ 0 < min(x,\ y) = True,\ x \leq max(x,\ y) = True,\ max(x,\ y) \leq x = True,\ z < g(x) + y = False\}\ \rangle$$

The transition list is initiated by [A2G, A2L]. Firstly, the non-linearizable and non-orientable equality $p(x) = True$ is transfered to G, by A2G. A2G is replaced by G2CR in the transition list, but its application on the non-orientable equality does not create new rewrite rules. Next, the rest of the (linearizable) equalities from A are dispatched to L, by A2L. The equalities from L are linearized and the new state becomes:

$$\begin{array}{rcl}\langle\ \mathsf{G} & : & \{p(x) = True\} \\ \mathsf{L} & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \leq 0,\ g(x) + y - z \leq 0\} \bullet \emptyset)\ \rangle\end{array}$$

By adding the inequalities $x - max(x,\ y) \leq 0$ and $max(x,\ y) - x \leq 0$, we get the inequality $0 \leq 0$. Therefore, the decision procedure for T_{la} can derive the equality $max(x,\ y) = x$. On the other hand, the inequality $g(x) + y - f(max(x,\ y)) \leq 0$ is the result of the addition between $g(x) + y - z \leq 0$ and $z - f(max(x,\ y)) \leq 0$. The new state after applying A2L is:

$$\begin{array}{rcl}\langle\ \mathsf{G} & : & \{p(x) = True\} \\ \mathsf{L} & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x,\ y)) \leq 0\} \bullet \{max(x,\ y) = x\})\rangle\end{array}$$

The next transition is L2G, so $max(x, y) = x$ of IE is transfered to G, to give:

$$\begin{array}{rcl}\langle\ \mathsf{G} & : & \{p(x) = True, max(x,\ y) = x\} \\ \mathsf{L} & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x,\ y)) \leq 0\} \bullet \{\emptyset\})\rangle\end{array}$$

The execution of congr on CR within the next G2CR transition produces no changes. However, since $max(x,\ y) >_t x$, a copy

of the equality $max(x,\ y) = x$ is transformed into the rewrite rule $max(x,\ y) \to x$ and stored to CR:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x\} \\ \mathsf{L} & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x,\ y)) \leq 0\} \bullet \{\emptyset\})\rangle \end{array}$$

Next, we apply normalize to convert $max(x, y)$ to x in all inequalities from P:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x\} \\ \mathsf{L} & : & (\{z - f(x) \leq 0,\ 1 - min(x,\ y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

By applying A2L, arith is again called, to give the new state:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x\} \\ \mathsf{L} & : & (\{z - f(x) \leq 0,\ 1 - min(x,\ y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

No new equalities are produced this time, so augment_L is called to add new inequalities to P. By instantiating the first equality of $R(C)$ with the substitution $\{x' \mapsto x; y' \mapsto y\}$, we get $\{max(x,\ y) = x \Rightarrow min(x,\ y) = y\}$ which can eliminate the maximal monomial $min(x,\ y)$ from $1 - min(x,\ y) \leq 0$. By adding the set of inequalities $\{min(x,\ y) - y \leq 0,\ y - min(x,\ y) \leq 0\}$ resulted from the linearization of $min(x,\ y) = y$, the transition A2L is fired to get the equality $min(x,\ y) = y$. It will be transfered afterwards to G by L2G, to give:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x,\ min(x,\ y) = y\} \\ \mathsf{L} & : & (\{z - f(x) \leq 0,\ 1 - min(x,\ y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0,\ 1 - y \leq 0,\ min(x,\ y) - y \leq 0,\ y - min(x,\ y)\} \bullet \emptyset)\rangle \end{array}$$

As for $max(x,\ y) = x$, the equality $min(x,\ y) = y$ can be oriented from left to right, and a copy of it is transformed into a rewrite rule, then added to CR by applying G2CR:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x,\ min(x,\ y) \to y\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x,\ min(x,\ y) = y\} \\ \mathsf{L} & : & (\{z - f(x) \leq 0,\ 1 - min(x,\ y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0,\ 1 - y \leq 0,\ min(x,\ y) - y \leq 0,\ y - min(x,\ y)\} \bullet \emptyset)\rangle \end{array}$$

The inequalities from P are normalized with $min(x,\ y) \to y$ to get:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x,\ min(x,\ y) \to y\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x,\ min(x,\ y) = y\} \\ \mathsf{L} & : & (\{z - f(x) \leq 0,\ 1 - y \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

The transition augment_L is applied a second time, now with the instance $p(x) = True \Rightarrow f(x) \leq g(x) = True$ of the second equality of $R(C)$ using the substitution $\{u \mapsto x\}$. The condition $p(x) = True$ is satisfied, so the inequality $f(x) - g(x) \leq 0$ is added to P:

$$\begin{array}{rcl} \langle\ \mathsf{CR} & : & \{max(x,\ y) \to x,\ min(x,\ y) \to y\} \\ \mathsf{G} & : & \{p(x) = True,\ max(x,\ y) = x,\ min(x,\ y) = y\} \\ \mathsf{L} & : & (\{f(x) - g(x) \leq 0,\ z - f(x) \leq 0,\ 1 - y \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0,\ 1 \leq 0\} \bullet \emptyset)\rangle \end{array}$$

Finally, arith yields the unsatisfiable inequality $1 \leq 0$ after adding $1 - y \leq 0$, $f(x) - g(x) \leq 0$ and $g(x) + y - f(x) \leq 0$. Hence, the procedure returns **inconsistent**.

Related works. Boyer and Moore [4] have been the first to define a heuristics using the augmentation operation. They integrated it into a decision procedure for linear arithmetic that can also manipulate disjunctions of linear inequalities and conditional linear inequalities. After conducting many experiments, they concluded that the augmentation operation increases significantly the performance of the decision procedure, hence improving the automatisation degree of their prover [16]. However, their integration schema is informal and lacks of termination proof. Several works have been inspired from Boyer-Moore's integration schema. [10] presented an enriched schema with a congruence closure procedure, implemented in the Tecton prover [17]. An informal termination proof of their integration schema is given in [11]. To our knowledge, the first formalisation of the Boyer-Moore's integration schema was given in [18], where it instantiated a more general 'plug and play' reasoning framework. [19] abstracts the interactions between rewriting and decision procedures via *constraint contextual rewriting (CCR)* rules parameterized by decision procedures. In the same line, [20] presents a flexible environment to integrate decision procedures into heuristic theorem provers.

Different cooperation schemas could have been used instead, as those inspired by Nelson-Oppen [21], Shostak [22] or, more recently, based on Delayed Theory Combination [23].

B. Integrating the external SMT solver

SPIKE can translate conditional specifications into SMT specifications following the SMT 2.0 format. By using a 'black-box' integration approach, the calls to the previous reasoning module have been replaced in SPIKE by calls to the Z3 SMT solver [24], version 4.3.2. For any call, the axioms and the negated conjecture are translated one-to-one into **assert** constructions. They are saved in a separate .smt2 file which is finally tested for satisfiability by an external Z3 process. If Z3 returns **unsat**, the conjecture is interpreted by SPIKE as valid and deleted from the current set of conjectures.

Z3 is an efficient SMT solver that combines different first-order theories using the Model Based Theory Combination approach [25], among which the theories for equality reasoning and arithmetics. It integrates a decision procedure for linear

arithmetic over integers that can be activated by interpreting the naturals from the SPIKE specification as non-negative integers. In order to do this, i) the natural sort is translated to the built-in integer sort 'Int', and ii) any SPIKE equality including a set of natural variables V will add the constraints $x \geq 0$, for any $x \in V$, as conditions in the corresponding **assert** construction.[2] The translation process is automatic, excepting for the user-defined sorts which should be manually translated.

C. Applications

We have previously used the reasoning module *à la* Boyer-Moore (BM) on several non-trivial applications [3], [9], [26]. Here, we detail our experience with the validation proof of the conformity algorithm for a telecommunications protocol, fully developed with the PVS [27] system in [28]. The proof is about showing the equivalence between two functions defined over naturals and lists. It mixes induction reasoning, case analysis and arithmetic reasoning. It also requires non-trivial user interaction, among which 79 user-defined lemmas.

Later on [26], a previous version (p.v.) of SPIKE integrating BM has proved completely automatically 48 user defined lemmas. To measure the impact of decision procedures on the automatisation degree of the prover, TABLE I indicates that a number of 69 lemmas was required by the successfully proved lemmas using the current version (c.v.) without the integration of reasoning modules.[3] The current version, this time integrating the reasoning modules, helped to completely automatically prove 46 lemmas by using the same proof strategy; only the last 'final' lemma required one additional lemma. It can be noticed that the overall BM-based proofs have been done 20 times faster than the overall Z3-based proofs. This can be explained by the non-negligeable time needed to launch the Z3 processes. On the other hand, the incompleteness of BM didn't penalize its effectiveness on this example, the two reasoning modules being able to prove the same conjectures. Notice that, in general, the two sets of conjectures proved with BM and Z3, respectively, may differ. For example, we can imagine divergent BM-derivations of BM false negatives that cannot be automatically proved by induction, i.e., without providing additional lemmas, but which can be successfully conducted by Z3.

Only one successful proof, done without the help of reasoning modules, was unsuccessful when using reasoning modules (see 'null_wind2'). In the other direction, there are 7 lemmas proved with reasoning modules but not proved using lemmas; in fact, we have found difficulties to provide the required lemmas leading to a successful proof. Also, it can be noticed that very few conjectures are not proved by SPIKE integrating reasoning modules. This is due to the slight differences between the specifications, mainly in the definition of the induction orderings and the parameterization of the inference rules. The implementation of some inference rules also changed, which explains why conjectures like 'null_insin' and 'null_insat' are proved by the previous version and not by the current version. For example, the implementation of some augment-like inference rule from [26] unsoundly adds to the conditions of the processed conjecture the conclusion of the involved conditional lemma. This error has been fixed in the current version.

#	lemma	p.v. w/ BM	c.v.		
			lemmas w/o r.m.	time (s) w/ r.m.	
				BM	Z3
01	firstat_timeat	0	1	0.052	0.749
02	firstat_progat	0	1	0.049	0.226
03	sorted_sorted	0	0	0.034	1.265
04	sorted_insat1	0	4	0.054	0.399
05	sorted_insin2	0	4	0.082	1.626
06	sorted_e_two	0	0	0.037	1.258
07	sorted_e_insin	0	3	0.049	1.633
08	member_t_insin	0	2	0.060	5.182
09	member_t_insat	no	2	0.057	4.164
10	member_firstat	0	2	0.046	6.172
11	timel_insat_t	0	0	0.037	3.121
12	erl_insin	0	0	0.035	2.134
13	erl_insat	0	0	0.036	2.134
14	erl_prog	0	2	0.058	7.340
15	time_progat_er	0	1	0.041	1.085
16	timeat_tcrt	0	0	0.042	4.147
17	timel_timeat_max	0	3	0.075	1.116
18	null_insat	0	no	no	no
19	null_insin	0	no	no	no
20	null_listat	0	1	0.035	5.164
21	null_listat1	0	0	0.031	0.063
22	cons_insat	0	0	0.031	3.115
23	cons_listat	0	0	0.028	0.058
24	progat_two_timeat	0	no	no	no
25	progat_timel_erl	0	3	0.068	2.145
26	progat_insat	0	4	0.935	11.410
27	progat_insat1	0	3	0.116	4.183
28	progat_insin_timeat	no	no	0.417	17.544
29	progat_insin	0	no	no	no
30	listat_insin_tcrt	0	no	no	no
31	progat_insin_t	0	3	0.508	15.511
32	listupto1_erl	0	no	0.819	1.164
33	null_listupto	0	no	0.068	1.147
34	listupto_t_insat	0	4	0.061	4.178
35	sorted_e_listupto	no	no	0.168	10.385
36	timel_listupto	0	0	0.049	1.094
37	sorted_listupto	0	4	0.054	6.206
38	progat_listupto	0	no	0.539	13.511
39	leftmax	0	no	no	no
40	leftmax_max	0	4	0.104	1.167
41	right_wind	0	4	0.064	1.108
42	time_listat	0	1	0.049	1.119
43	listat_listupto	0	4	0.218	13.46
44	sorted_cons_listat	0	no	0.110	11.345
45	null_wind1	0	1	0.048	1.135
46	null_wind2	0	1	no	no
47	member_t_timel	no	no	0.074	8.229
48	timel_insin1	0	1	0.040	1.077
49	null_listupto1	0	0	0.030	2.203
50	sorted_e_cons	0	0	0.034	1.118
51	erl_cons	0	0	0.036	1.115
52	no_time	0	2	0.130	7.245
53	final	1	3	0.042	5.183
Total		1	69	< 10 s	> 3 min

TABLE I: Statistics about the proof with (w/) and without (w/o) reasoning modules (r.m.).

The experiments have been performed on a MacBook Pro notebook featuring a 2.6 GHz Intel Core i5 processor and 8 Go RAM. [4]

IV. Conclusions and Future Work

We detailed the implementation in SPIKE of two reasoning modules integrating components for arithmetic and equality reasoning. The integration schema *à la* Boyer-Moore was

[2]This solution was suggested by Pascal Fontaine in a private communication.

[3]'no' means that the proof was not successful.

[4]For the reviewers, the full specifications and proofs can be found at http://lita.sciences.univ-metz.fr/~stratula/synasc2014.zip

firstly presented in [15] and served later as example for an approach that combined CCR and implicit induction techniques [9]. Compared to [4], it manages neither conditional linear inequalities nor disjunctions of linear inequalities. Moreover, it does not use abstraction variables as in [21], which makes impossible the application of the decision procedure for linear arithmetic on subterms of monomials or non-linearizable equality and disequality sides. On the other hand, it is fast and has been successfully used to automatize the validation proof from [28], representing one of the most challenging case studies ever tested with a reasoning specialist based on the Boyer-Moore's integration schema. However, the cooperation schema is rather complex and error-prone. Relaunching the proofs using the Z3-based reasoning module can help checking the (implementation) soundness of the cooperation schema. Moreover, the completeness property of the arithmetic component in Z3 is an added-value to SPIKE; we expect to prove more conjectures in a completely automatic way.

In the future, we intend to tweak around Z3 for speeding up its performance when checking for unsatisfiability. It would be interesting to combine induction with reasoning modules for other first-order theories that Z3 integrates (fixed-sized bit-vectors, arrays, etc). We also intend to try other STM solvers compatible with the SMT format (Simplify, CVC3, Yices, etc).

Acknowledgments. The author would like to thank Pascal Fontaine and Christophe Ringeissen for useful discussions. Anisia Maria Magdalena Tudorescu has partially implemented the integration of Z3 in SPIKE during a three-month INRIA internship.

REFERENCES

[1] S. Stratulat, "A general framework to build contextual cover set induction provers," *J. Symb. Comput.*, vol. 32, no. 4, pp. 403–445, 2001.

[2] A. Bouhoula, E. Kounalis, and M. Rusinowitch, "Automated mathematical induction," *Journal of Logic and Computation*, vol. 5, no. 5, pp. 631–668, 1995.

[3] G. Barthe and S. Stratulat, "Validation of the JavaCard platform with implicit induction techniques," in *RTA*, ser. Lecture Notes in Computer Science, R. Nieuwenhuis, Ed., vol. 2706. Springer, 2003, pp. 337–351.

[4] R. S. Boyer and J. S. Moore, "Integrating decision procedures into heuristic theorem provers: a case study of linear arithmetic," in *Machine Intelligence*. Oxford University Press, Inc. New York, NY, USA, 1988, pp. 83–124.

[5] F. Baader and T. Nipkow, *Term Rewriting and All That*. Cambridge University Press, 1998.

[6] S. Stratulat, "A unified view of induction reasoning for first-order logic," in *Turing-100 (The Alan Turing Centenary Conference)*, ser. EPiC Series, A. Voronkov, Ed., vol. 10. EasyChair, 2012, pp. 326–352.

[7] E. Kounalis and M. Rusinowitch, "Mechanizing inductive reasoning," in *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, ser. AAAI'90. AAAI Press, 1990, pp. 240–245.

[8] F. Bronsard, U. Reddy, and R. Hasker, "Induction using term orderings," in *Automated Deduction —CADE-12*, ser. LNCS, vol. 814. Springer, 1994, pp. 102–117.

[9] A. Armando, M. Rusinowitch, and S. Stratulat, "Incorporating decision procedures in implicit induction," *J. Symb. Comput.*, vol. 34, no. 4, pp. 241–258, 2002, a previous version appeared in Calculemus 2001 (9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning), pages 65-75.

[10] D. Kapur and X. Nie, "Reasoning about numbers in Tecton," in *Intl. Symp. Methodologies for Intelligent Systems*, 1994, pp. 57–70.

[11] D. Kapur, "Rewriting, induction and decision procedures: A case study of presburger arithmetic," in *Symbolic-Algebraic Methods and Verification Methods — Theory and Applications*, 2001, pp. 129–144.

[12] J.-L. Lassez and M. Maher, "On Fourier's algorithm for linear arithmetic constraints," *Journal of Automated Reasoning*, vol. 9, pp. 373–379, 1992.

[13] M. Kaufmann and J. S. Moore, *ACL2 Version 2.4 - The User's Manual*, 1999.

[14] G. Huet and D. S. Lankford, "On the uniform halting problem for term rewriting systems," Laboria, Tech. Rep. 283, 1978.

[15] S. Stratulat, "Preuves par récurrence avec ensembles couvrants contextuels. applications à la vérification de logiciels de télécommunications." Ph.D. dissertation, Université Henri Poincaré, Nancy I, November 2000, also published as a book at 'Editions Universitaires Européennes' in 2012, ISBN 978-3841794901.

[16] R. S. Boyer and J. S. Moore, *A computational logic handbook*. Academic Press Professional, 1988.

[17] D. Kapur, X. Nie, and D. R. Musser, "An overview of the Tecton proof system," *Theor. Comput. Sci.*, vol. 133, no. 2, pp. 307–339, 1994.

[18] F. Giunchiglia, P. Pecchiari, and C. Talcott, "Reasoning Theories-Towards an Architecture for Open Mechanized Reasoning Systems," in *First International Workshop on Frontiers of Combining Systems (FroCoS).*, ser. Kluwer Series on Applied Logic. Kluwer Academic Publishers, 1996, pp. 157–174, also, published as the technical report 9409-15, IRST, November 1994.

[19] A. Armando and S. Ranise, "Constraint Contextual Rewriting," *Proceedings of the 2nd International Workshop on First Order Theorem Proving,FTP'98, Vienna (Austria)*, pp. 65–75, 1998, an extended version appeared in Journal of Symbolic Computation, 36(2003), 193-216.

[20] P. Janičić, A. Bundy, and I. Green, "A framework for the flexible integration of a class of decision procedures into theorem provers," in *Automated Deduction — CADE-16*, ser. Lecture Notes Computer Science, vol. 1632, 1999, pp. 127–141.

[21] G. Nelson and D. C. Oppen, "Simplification by cooperating decision procedures," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 2, pp. 245–257, 1979.

[22] R. Shostak, "Deciding Combinations of Theories," *Journal of the ACM*, vol. 31, no. 1, pp. 1–12, 1984.

[23] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P. V. Rossum, and R. Sebastiani, "Efficient satisfiability modulo theories via delayed theory combination," in *In Proc. CAV 2005*, ser. LNCS, vol. 3576. Springer, 2005, pp. 335–349.

[24] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[25] L. de Moura and N. Bjørner, "Model-based theory combination," *ENTCS*, vol. 198, no. 2, pp. 37–49, May 2008.

[26] M. Rusinowitch, S. Stratulat, and F. Klay, "Mechanical verification of an ideal incremental ABR conformance algorithm," *J. Autom. Reasoning*, vol. 30, no. 2, pp. 53–177, 2003.

[27] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert, *PVS prover guide - version 2.4*, SRI International, November 2001.

[28] M. Rusinowitch, S. Stratulat, and F. Klay, "Mechanical verification of an ideal incremental ABR conformance algorithm," in *CAV*, ser. Lecture Notes in Computer Science, E. A. Emerson and A. P. Sistla, Eds., vol. 1855. Springer, 2000, pp. 344–357.