Journal of Universal Computer Science, vol. 7, no. 2 (2001), 124-140
submitted: 1/9/00, accepted: 13/10/00, appeared: 28/2/01 © Springer Pub. Co.

A Practical Extension Mechanism for Decision Procedures: the Case Study of Universal Presburger Arithmetic

Alessandro Armando
(DIST – Università degli Studi di Genova, Genova, Italia
armando@dist.unige.it)

Silvio Ranise
(DIST – Università degli Studi di Genova, Genova, Italia
LORIA – Université Henri Poincaré, Nancy, France
silvio@dist.unige.it)

Abstract: In this paper, we propose a generic mechanism for extending decision procedures by means of a lemma speculation mechanism. This problem is important in order to widen the scope of decision procedures incorporated in state-of-the-art verification systems. Soundness and termination of the extension schema are formally stated and proved. As a case study, we consider extensions of a decision procedure for the quantifier-free fragment of Presburger Arithmetic to significant fragments of non-linear arithmetic.

Key Words: Affinization, Augmentation, Formal Verification, Decision Procedures, Lemma Speculation, Universal Arithmetic over Integers, Universal Presburger Arithmetic over Integers
Category: I.2.3, I.1.1, I.1.2, F.3.1

1 Introduction

The lack of automated support is probably the main obstacle to the application of formal method techniques in the industrial setting. As witnessed by the success of model checking techniques, formal methods are readily employed in industry as soon as automated reasoning tools providing a sufficiently high degree of automation are available. On the other hand, the only way to meet the requirements posed by many industrial applications is to combine the expressiveness of general purpose provers with the efficiency of specialized reasoners (such as, e.g., decision procedures and unification algorithms). Unfortunately this turns out to be a surprisingly difficult task. The main problem is that only a tiny portion of the proof obligations arising in many practical applications falls exactly into the domain the specialized reasoners are designed to solve.

To illustrate, let us assume that a decision procedure for Presburger Arithmetic over Integers is available and consider the problem of proving the formula:

$$(l \leq min(a) \land 0 < k \land a \neq [\,]) \Rightarrow l < max(a) + k \tag{1}$$

where l and k are constants denoting arbitrary integers, a is a constant denoting an arbitrary list of integers, $[\,]$ denotes the empty list of integers, max (min) is a unary function symbol denoting a function which returns the maximum (minimum, resp.) element of the list of integers given as input, and the remaining symbols (namely 0, +, and <) have their usual arithmetic interpretation. The key point here is that the decision procedure for Presburger Arithmetic is only aware of the interpretation of the arithmetics symbols (i.e. 0, +, and <) and treats all the terms whose top-most function symbol is non-arithmetic as uninterpreted. Under such assumptions, the decision procedure can not possibly establish the validity of (1).

Boyer and Moore recognized this difficulty when they incorporated a decision procedure for the quantifier-free fragment of Presburger Arithmetic over naturals into their prover [BM88]. To cope with the problem, they proposed an elaborated incorporation schema between the rewriter and the decision procedure. One of the key ingredient of the effectiveness of such incorporation schema is *augmentation*. This mechanism aims at making the decision procedure aware of properties of function symbols it is otherwise not aware of through the appropriate use of available lemmas. Going back to our example, if the following lemma is available

$$X \neq [\,] \Rightarrow min(X) \leq max(X) \tag{2}$$

(where X is a variable ranging over lists of integers), then augmentation inspects the internal state of the decision procedure (which stores the negation of (1), namely $l \leq min(a) \wedge 0 < k \wedge a \neq [\,] \wedge l \not< max(a) + k$), instantiates (2) with the substitution $\{a/X\}$, and finally extends the internal state of the decision procedure with the resulting instance. The new state is easily found unsatisfiable by the decision procedure (hence we conclude that (1) holds, by refutation). Notice that conditional lemmas can be a source of problems: when trying to establish the conditions of the lemmas the prover can be recursively invoked and special devices must be put in place to ensure termination. In our example, the (instantiated) hypothesis of the lemma, namely $a \neq [\,]$, can be readily relieved by inspecting the initial state of the decision procedure.

In this paper, we address **the problem of lifting decision procedures for a given theory T to proof procedures capable of proving formulae belonging to non trivial extensions of T**. In the example above, T is the theory of Universal Presburger Arithmetic over integers and the extension of T under consideration contains (2). For concreteness, we consider a simplified variant of the Fourier-Motzkin elimination method over rationals as a procedure for the theory of Universal Presburger Arithmetic over integers.[1] The method is sound but incomplete (although it can be extended to a decision procedure over integers, as shown in [KN94]). This situation reflects the common practice in state-of-art-verification systems to adopt incomplete methods for decidable theories on the basis of pragmatical considerations (see, e.g., [NO78] for a discussion of this issue).

This paper makes two contributions.

- First, a *generic extension schema* for decision procedures is presented. The key ingredient of such a schema is a *lemma speculation mechanism* which 'reduces' the validity problem of the extended theory to the validity problem of the base theory T. The schema is generic both in the decision procedure and

[1] However, most of the discussion in this paper carries over to other decision procedures for the theory of Universal Presburger Arithmetic, such as the incremental version of the Simplex method described in [Nel81].

in the lemma speculation mechanism. It enjoys basic theoretical properties such as soundness and termination.

– Second, *three instances* of the extension schema lifting a decision procedure for the theory of Universal Presburger Arithmetic based on (a variant of) the Fourier-Motzkin elimination method are described: augmentation, affinization, and their combination. *Augmentation* copes with user-defined functions whose properties can be expressed by conditional lemmas. It is seen in isolation and not in cooperation with other modules (e.g. a rewriter) as in [BM88, AR98, AR00]. This, we hope, will clarify the idea underlying it and foster the adoption of this powerful technique in other systems. *Affinization* is a mechanism for the 'on-the-fly' generation of lemmas. It is a significant improvement over augmentation, since it relieves the user from the burden of supplying lemmas about multiplication. Finally, a *combination* of augmentation and affinization puts together the flexibility of the former with the automation of the latter. Since the theory of Universal Arithmetic is ubiquitous in mechanical verification, the proposed extended procedures aims at enhancing the power of existing verification systems.

Plan of the paper. The paper begins with the abstract description of our extension schema (Section 2). Then it presents three instances of the extension schema lifting the Fourier-Motzkin based decision procedure for Universal Presburger Arithmetic over integers (Section 3). Some experimental results are presented in Section 4. The related work is discussed in Section 5 and some final remarks are drawn in Section 6.

2 Extending Decision Procedures

We assume the usual notion of quantifier-free first-order language, term, formula, logical consequence (in symbols, $\models$), validity, satisfiability, and theory (see, e.g., [End72]). Let T be a theory. We say that a syntactical object o is in $\mathcal{L}(T)$ to abbreviate that o is in the language of T. Let ϕ and ψ be two formulae of $\mathcal{L}(T)$. ϕ is T-valid if, and only if, $T \models \phi$. ϕ is T-entailed by ψ (in symbols, $\psi \models_T \phi$) if, and only if, $\psi \Rightarrow \phi$ is T-valid, where $\Rightarrow$ is the logical connective for implication. ϕ is T-equivalent to ψ if, and only if, ϕ is T-entailed by ψ and ψ is T-entailed by ϕ.

Let T_c and T_j be two first-order theories s.t. $T_c \subset T_j$ and the predicate symbols of $\mathcal{L}(T_j)$ are those of $\mathcal{L}(T_c)$. Following [NO78], we consider the problem of proving the un-satisfiability of ground conjunctions of literals in $\mathcal{L}(T_j)$.

The *objective* of our extension schema is to build a proof procedure capable of establishing the T_j-validity of a non trivial superset of T_c on top of a decision procedure for T_c. By an 'extensible' decision procedure, we mean a *state-based*, *incremental*, and *resettable* procedure. Its state represents conjunctions of literals stored in some internal form, e.g. a form in which the satisfiability of the set of literals can be detected by a computationally inexpensive check. To be resettable and incremental, the decision procedure must have the capability to add and remove literals from its state without restarting from scratch. These characteristics are of paramount importance to obtain decision procedures of practical interest which are eventually integrated in larger verification systems (see, e.g. [NO78]).

2.1 The Decision Procedure

We assume to work with the following sets of objects. lit is the set of ground literals of $\mathcal{L}(T_j)$. state is the set of states of the decision procedure, representing conjunctions of elements in lit; $l_1 \wedge ... \wedge l_n$ (also denoted with $\bigwedge_{i=1}^{n} l_i$) is in state if l_i $(i = 1, ..., n)$ is in lit. bool is the set of boolean values.
Two interface functions model the (extensible) decision procedure.

- $\mathsf{simp} : \mathsf{lit} \times \mathsf{state} \longrightarrow \mathsf{state}$. $\mathsf{simp}(l, S)$ computes and returns the new state S' resulting from the addition of a literal l to S in such a way that S' is T_c-entailed by $l \wedge S$. For termination it is required that

$$\mathsf{simp}(l, S') \preceq \mathsf{simp}(l, S) \tag{3}$$

  holds, for all $\mathsf{literals}$ l and for all states S and S' s.t. $S' \prec S$, where $\preceq$ is an ordering relation over states whose strict version, $\prec$, is well-founded.
- $\mathsf{unsat} : \mathsf{state} \longrightarrow \mathsf{bool}$. $\mathsf{unsat}(S)$ characterizes a sub-set of the unsatisfiable states whose T_c-unsatisfiability can be checked by means of a computationally inexpensive check. $\mathsf{unsat}(S)$ implies that S is T_c-unsatisfiable.

Example 1. (A Decision Procedure for Universal Presburger Arithmetic.) Consider the first-order language consisting of the numerals $..., -2, -1, 0, 1, 2, ...$, variables, the function symbol $+$, the (infix) binary predicate symbols $<$, $\leq$, $=$, $\geq$, and $>$, and the usual logical connectives. The intended structure of this language interprets numerals as integers,[2] variables range over integers, $+$ is interpreted as addition, $<$, $\leq$, $\geq$, and $>$ are interpreted as the usual ordering relations, and $=$ is interpreted as the identity relation. We call the theory of this structure, the theory of *Universal Presburger Arithmetic over Integers* (UPAI). The theory of Universal Arithmetic over Integers (UAI) is an extension of UPAI with multiplication $*$. T_c (T_j) is a first-order theory containing UPAI (UAI, resp.) and n-ary function symbols other than $+$ (other than $+$ and $*$, resp.) interpreted as functions from n-tuples of integers to integers. Notice that $T_c \subset T_j$.

The Fourier-Motzkin elimination method is based on the idea of eliminating one variable at a time in the hope of obtaining a 'trivially' unsatisfiable inequality, e.g. $0 \leq -1$. It can easily be lifted to a proof procedure for T_c.[3] We assume that $<$, $=$, $\geq$, and $>$ (in the language of UPAI) are preliminary eliminated in favour of $\leq$ (e.g. $x < 0$ can be rewritten to $x \leq -1$ by exploiting the integral property of integers). The literals in lit are inequalities in the normal form

$$c_1 \cdot m_1 + \cdots + c_n \cdot m_n \leq c \tag{4}$$

[2] In rest of the paper, we will use the term 'integer' in place of 'numeral'. The context will make it clear which one we are referring to.

[3] It is well-known that the Fourier-Motzkin method has exponential space complexity and it is incomplete over integers (although a proposal to make it complete is described in [KN94]). Despite these bad theoretical properties, the method is usable in practice as observed, e.g., in [BM88] and it is implemented in some state-of-the-art verification systems (such as ACL2 [KM97], STeP [M+94], and Tecton [KMN94]). The variant of the Fourier-Motzkin method described here simplify the argument of termination for the extended versions of the decision procedure since the ordering over multiplicands is also used by the lemma speculation mechanism (see Examples 2 and 3). Finally, it is worth noticing that other decision procedures for UPAI (such as the Simplex method) can be modelled in our framework.

where $n \geq 0$ (if $n = 0$, then (4) stands for $0 \leq c$), $c, c_1, ..., c_n$ are relatively prime integers (called *coefficients*), $m_1, ..., m_n$ are terms in $\mathcal{L}(T_j)$ (called *multiplicands*) whose top-most function symbols are different from $+$ s.t. $m_{i+1} \prec^m m_i$ where $\prec^m$ is a total ordering over multiplicands ($i = 1, ..., n$), and $c_i \cdot m_i$ ($i = 1, ..., n$) abbreviates the term $m_i + \cdots + m_i$ in which m_i occurs c_i times. m_1 is the *heaviest* multiplicand in (4) w.r.t. the total ordering $\prec^m$. This is lifted to inequalities lexicographically in the obvious way. An element in the set state is a pair $\langle C, M \rangle$ (denoted with C_M), where C is a conjunction of inequalities in lit and M is a set of multiplicands.

Let ι and ι' be two inequalities of the form (4) both having m as their heaviest multiplicand, k (k') is the coefficient of m in ι (ι', resp.), k and k' are of opposite sign, and $elim(\iota, \iota')$ is the normal form of the linear combination of ι and ι' not containing m. The inequality $elim(\iota, \iota')$ is entailed by ι and ι'. Let $C|_M$ denote the conjunction of inequalities in C_M not containing any element of M as heaviest multiplicand. If the heaviest multiplicand m of ι is $\prec^m$-smaller or equal to any element in M, then $\mathsf{simp}(\iota, C_M) = C'_{M \cup \{m\}}$ where C' is the *closure* of $elim$ on $\iota \wedge C|_M$ in the sense that for any ι_0 in C' we have that $elim(\iota, \iota_0) \in C'$ (if $elim$ is defined). Otherwise, $\mathsf{simp}(\iota, C_M) = C_M$. (Therefore, M contains all the multiplicands already eliminated by a Fourier-Motzkin elimination step.) It is trivial to verify that C' is T_c-entailed by $\iota \wedge C$, if $C'_{M'} = \mathsf{simp}(\iota, C_M)$. Under the assumption that the initial state is closed under $elim$, we obtain an incremental version of the decision procedure by restricting the computation of the closure to the input inequality and its consequences only. Given the states C_M and $C'_{M'}$, we define $C_M \prec C'_{M'}$ if, and only if, the heaviest multiplicand in $C|_M$ is $\prec^m$-less than the heaviest multiplicand in $C'|_{M'}$. It is easy to verify that the definitions of simp and of $\prec$ enjoy (3). $\mathsf{unsat}(C_M)$ holds if an inequality $0 \leq c$ is in C s.t. c is a negative integer. This implies that C is T_c-unsatisfiable. $\triangle$

2.2 The Extension Schema

Let lemma be the set whose elements represent (ground) formulae of the form $\bigwedge_{j=1}^m c_j$ or of the form $\bigwedge_{i=1}^n q_i \Rightarrow \bigwedge_{j=1}^m c_j$ ($n \geq 1$ and $m \geq 1$), where $q_1, ..., q_n$ and $c_1, ..., c_m$ are in lit.
The lemma speculation mechanism is modelled by the following function.

- $\mathsf{genlemma} : \mathsf{state} \longrightarrow \mathsf{state} \times \mathsf{lemma}^*$.[4] This function speculates lemmas T_j-entailed by the state of the procedure. If $\mathsf{genlemma}(S) = \langle S', [\gamma_1, ..., \gamma_s] \rangle$ then S' is T_c-equivalent to S and γ_k ($k = 1, ..., s$) is T_j-entailed by S. For termination it is also required that

$$S' \prec S \tag{5}$$

  and for each γ_k ($k = 1, ..., s$) of the form $\bigwedge_{i=1}^n q_i \Rightarrow \bigwedge_{j=1}^m c_j$ it is also required that (for $i = 1, ..., n$)[5]

$$\mathsf{simp}(\neg q_i, S') \prec S \tag{6}$$

$$\mathsf{simp}([c_1, ..., c_m], S') \prec S. \tag{7}$$

[4] If A is a set, A^* denotes the set of (finite) lists of elements in A. $[o_1, ..., o_n]$ denotes an element of A^*, where $o_1, ..., o_n$ are in A.

[5] If l is an atom in lit, then $\neg l$ abbreviates the negation of l in lit. If l is the negation of an atom a in lit, then $\neg l$ stands for a in lit.

where $\mathsf{simp}([c_1, ..., c_m], S)$ abbreviates $\mathsf{simp}(c_m, ..., \mathsf{simp}(c_1, S)...)$. For each γ_k ($k = 1, ..., s$) of the form $\bigwedge_{j=1}^{m} c_j$, only (7) is required to hold.[6]

We are now in the position to precisely define our extension schema.

$$
\begin{array}{l}
\mathsf{extend} : \mathsf{lit}^* \times \mathsf{state} \longrightarrow \mathsf{state} \\
\mathsf{extend}([l_1, ..., l_m], S) \;=\; \\
\quad \mathsf{let}\ S_1 = \mathsf{simp}([l_1, ..., l_n], S)\ \mathsf{in} \\
\quad\quad \mathsf{if}\ \mathsf{unsat}(S_1)\ \mathsf{then}\ S_1 \\
\quad\quad \mathsf{else\ let}\ \langle S_2, [\gamma_1, ..., \gamma_s]\rangle = \mathsf{genlemma}(S_1)\ \mathsf{in} \\
\quad\quad\quad\quad \mathsf{extstate}(\gamma_s, ..., \mathsf{extstate}(\gamma_1, S_2)...)
\end{array}
$$

We add the literals $l_1, ..., l_n$ to the state S (first let construct) and we check whether the resulting state S_1 is T_c-unsatisfiable (test of the conditional). If it is, we simply return S_1 (then branch). Otherwise (else branch), by invoking genlemma, we try to speculate T_j-entailed formulae (second let construct) which will hopefully enable us to extend the state of the procedure. In the last line of extend, we return the nested application of the function extstate over the formulae of the list $[\gamma_1, ..., \gamma_s]$. (If genlemma returns the empty list of formulae, $\mathsf{extstate}(\gamma_s, ..., \mathsf{extstate}(\gamma_1, S_2)...)$ simplifies to S_2.)

$\mathsf{extstate}(\gamma, S)$ attempts to extend S by using formula γ (T_j-entailed by S).

$$
\begin{array}{l}
\mathsf{extstate} : \mathsf{lemma} \times \mathsf{state} \longrightarrow \mathsf{state} \\
\mathsf{extstate}(\bigwedge_{i=1}^{n} q_i \Rightarrow \bigwedge_{j=1}^{m} c_j, S) \;=\; \\
\quad \mathsf{if}\ \mathsf{entailed}(q_1, S)\ \mathsf{and}\ ...\ \mathsf{and}\ \mathsf{entailed}(q_n, S) \\
\quad \mathsf{then}\ \mathsf{simp}([c_1, ..., c_m], S)\ \mathsf{else}\ S \\
\mathsf{extstate}(\bigwedge_{j=1}^{m} c_j, S) \;=\; \mathsf{simp}([c_1, ..., c_m], S)
\end{array}
$$

The conclusions $c_1, ..., c_m$ of the formula $\bigwedge_{i=1}^{n} q_i \Rightarrow \bigwedge_{j=1}^{m} c_j$ are added to the state S of the decision procedure and the resulting new state is returned (then branch), provided that S entails all the conditions $q_1, ..., q_n$ (test of the conditional). If this is not the case, we simply return S (else branch). If the lemma is unconditional (i.e. of the form $\bigwedge_{j=1}^{m} c_j$), we simply add $c_1, ..., c_m$. All the obvious improvements (such as to stop adding $c_1, ..., c_m$ as soon as an unsatisfiable state is obtained) are not considered here for simplicity.

entailed checks whether a literal l is T_j-entailed by the state of the procedure S by adding the negation of l to S and checking whether the resulting state is T_c-unsatisfiable.

$$
\begin{array}{l}
\mathsf{entailed} : \mathsf{lit} \times \mathsf{state} \longrightarrow \mathsf{bool} \\
\quad\quad \mathsf{entailed}(l, S) \;=\; \mathsf{unsat}(\mathsf{extend}([\neg l], S))
\end{array}
$$

It is important to notice that if a formula ϕ has been shown unsatisfiable by the decision procedure for T_c, there is no performance loss when invoking on ϕ the extended version of the decision procedure. This is so because we resort to the lemma speculation mechanism only when the decision procedure for T_c fails to prove the unsatisfiability of ϕ.

---

[6] The intuition underlying these requirements is discussed in Example 2, Section 3.1.

2.3 Properties of the Extension Schema

Since entailed and extend are mutually recursive via extstate, checking for entailment may involve relieving further conditions. This makes the termination of the proposed extension schema not obvious.

Theorem 1 (Termination). $\mathsf{extend}([l_1,...,l_n],S)$ *terminates for all lists of literals* $[l_1,...,l_n]$ *and for all states* S.

Proof. All the recursive calls to extend occur in $\mathsf{extstate}(\gamma_s,...,\mathsf{extstate}(\gamma_1,S_2)...)$ and are of the form $\mathsf{extstate}(\gamma_k,S_2^k)$, where

$$\begin{aligned} S_2^0 &= S_2 \\ S_2^k &= \mathsf{if\ unsat(extend}([\neg q_1^k],S_2^{k-1}))\ \mathsf{and}\ ...\ \mathsf{and\ unsat(extend}([\neg q_{n_k}^k],S_2^{k-1})) \\ &\quad\ \mathsf{then\ simp}([c_1^k,...,c_{m_k}^k],S_2^{k-1}) \\ &\quad\ \mathsf{else}\ S_2^{k-1} \end{aligned} \tag{8}$$

where γ_k is of the form $\bigwedge_{i=1}^{n_k} q_i^k \Rightarrow \bigwedge_{j=1}^{m_k} c_j^k$, $n_k \geq 1$, $m_k \geq 1$, and $k = 1,...,s$. (We do not consider lemmas of the form $\bigwedge_{j=1}^{m_k} c_j^k$, since it is trivial to conclude in this case.) We must show that in each (possible) recursive call (see the test of the conditional in (8)) a suitable measure function is decreased according to a well-founded ordering. Let simp be such a measure function and $\prec$ be the associated well-founded ordering. We must prove that

$$\mathsf{simp}(\neg q_j^k, S_2^{k-1}) \prec \mathsf{simp}([l_1,...,l_n],S) \tag{9}$$

holds for $j = 1,...,n_k$ and $k = 1,...,s$.
From (5) and (3), it is easy to see that

$$\mathsf{simp}([l_1,...,l_n],S_2) \preceq \mathsf{simp}([l_1,...,l_n],S). \tag{10}$$

If we can prove for $j = 1,...,n_k$ and $k = 1,...,s$ that

$$\mathsf{simp}(\neg q_j^k, S_2^{k-1}) \prec \mathsf{simp}([l_1,...,l_n],S_2^0), \tag{11}$$

then (9) readily follows from (11), (10), and the fact $S_2^0 = S_2$ in (8).

We prove (11) by induction on s. The induction hypothesis is

$$\mathsf{simp}(\neg q_j^k, S_2^{k-1}) \prec \mathsf{simp}([l_1,...,l_n],S_2^0) \tag{12}$$

for $j = 1,...,n_k$. According to (8) and (7), there are two cases to consider.

– $S_2^k = S_2^{k-1}$. By (12), it is trivial to derive $\mathsf{simp}(\neg q_j^k, S_2^k) \prec \mathsf{simp}([l_1,...,l_n], S_2^0)$, for $j = 1,...,n_k$.

– $S_2^k \prec S_2^{k-1}$. From (3), we have $\mathsf{simp}(\neg q_j^k, S_2^k) \preceq \mathsf{simp}(\neg q_j^k, S_2^{k-1})$. Then, by (12), we derive $\mathsf{simp}(\neg q_j^k, S_2^k) \prec \mathsf{simp}([l_1,...,l_n],S_2^0)$, for $j = 1,...,n_k$. □

Another fundamental property enjoyed by our extension schema is soundness.

Theorem 2 (Soundness). *For all lists of literals $[l_1, ..., l_n]$ and for all states S, if $S' = \mathsf{extend}([l_1, ..., l_n], S)$, then S' is T_j-entailed by $l_1 \wedge ... \wedge l_n \wedge S$.*

Proof. From the assumption $T_c \subset T_j$ and the requirement that S' is T_c-entailed by $l \wedge S$ (where $\mathsf{simp}(l, S) = S'$), for all literals l and for all states S, it is easy to see that

$$S_1 \text{ is } T_j\text{-entailed by } l_1 \wedge ... \wedge l_n \wedge S. \tag{13}$$

There are two cases to consider according to the test of the conditional in extend. If $\mathsf{unsat}(S_1)$, then S_1 is returned and it is trivial to conclude from (13). Otherwise, from (13), the assumption $T_c \subset T_j$, and the fact that $\mathsf{genlemma}(S_1) = \langle S_2, [\gamma_1, ..., \gamma_s]\rangle$ is s.t. S_2 is T_c-equivalent to S_1, we derive

$$S_2 \text{ is } T_j\text{-entailed by } l_1 \wedge ... \wedge l_n \wedge S. \tag{14}$$

If we can prove that

$$S_2^k \text{ is } T_j\text{-entailed by } S_2^0 \tag{15}$$

for all $k = 1, ..., s$, then we conclude from (15), (14), and the base case of (8).

We prove (15) by induction on s. The induction hypothesis is

$$S_2^{k-1} \text{ is } T_j\text{-entailed by } S_2^0. \tag{16}$$

According to (8) and the fact that $\mathsf{simp}(l, S) = S'$ is s.t. S' is T_c-entailed by $l \wedge S$ for all states S and for all literals l, two cases are to be considered.

- $S_2^k = S_2^{k-1}$. It is trivial to conclude by (16).
- S_2^k is T_j-entailed by $c_1^k \wedge ... \wedge c_{m_k}^k \wedge S_2^{k-1}$. From (16), the assumption $T_c \subset T_j$, and the facts that γ_k is T_j-entailed by S_1 (since $\mathsf{genlemma}(S_1) = \langle S_2, [\gamma_1, ..., \gamma_s]\rangle$), that S_2 is T_c-equivalent to S_1, and that $S_2^0 = S_2$ in (8), we derive

$$\gamma_k \text{ is } T_j\text{-entailed by } S_2^{k-1}. \tag{17}$$

If γ_k is of the form $\bigwedge_{j=1}^{m_k} c_j$, then we conclude that S_2^k is T_j-entailed by S_2^0 from (17) and (16). Otherwise, γ_k has the form $\bigwedge_{i=1}^{n_k} q_i \Rightarrow \bigwedge_{j=1}^{m_k} c_j$. It is easy to see (from the definition of entailed) that

$$q_i \text{ is } T_j\text{-entailed by } S_2^{k-1} \tag{18}$$

for all $i = 1, ..., n_k$. From (17) and (18), we derive

$$\bigwedge_{j=1}^{m_r} c_j \text{ is } T_j\text{-entailed by } S_2^{k-1}. \tag{19}$$

Then, by (16) and (19), we conclude S_2^k is T_j-entailed by S_2^0. □

It would be interesting to precisely characterize classes of lemma speculation mechanisms which guarantee the completeness of the extended procedures. Unfortunately, there are three main problems in achieving completeness in the typical verification efforts our extension schema has been designed for. Firstly, it is common practice to adopt an incomplete method to prove formulae in T_c for efficiency reasons (see Example 1). However, this problem is of little concern in practice since experiments show that this kind of incompleteness rarely occurs in real verification efforts (see, e.g., [BM88, NO78]). Secondly, our extension schema does not handle disjunction of literals which are sometimes necessary to design complete reduction mechanisms for the satisfiability problem of certain theories (see Example 3). In practice, this is not a big problem since decision procedures are usually incorporated in larger systems where other reasoning modules are encharged to process the formulae (e.g. by case analysis). These modules put formulae in a form in which they can be decided by the the decision procedures. Finally, it is difficult to give a syntactic characterization of a decidable theory which is also useful in practice (see, e.g. [SJ80], for a discussion of this problem). Preliminary investigations about 'completeness-preserving' lemma speculation mechanisms to extend a decision procedure for ground equality are reported in [ARR00].

3 Extensions of a Decision Procedure for Universal Presburger Arithmetic over Integers

The extension schema is parametric both in the decision procedure (i.e. simp and unsat) and in the lemma speculation mechanism (i.e. genlemma). Example 1 describes a proof procedure satisfying requirement (3). In what follows, we focus on three instances of genlemma (satisfying requirements (5), (6), and (7)), which allow to obtain three sound and terminating procedures capable of establishing the validity of formulae belonging to non trivial extensions of T_c (where T_c is UPAI extended with arbitrary function symbols, as in Example 1).

3.1 Augmentation

Augmentation [BM88, AR98, AR00] extends the information available to the decision procedure with selected instances of lemmas encoding properties of symbols the decision procedure does not know anything about. For example, $*$ is uninterpreted for T_c and interpreted for T_j (where T_j is UAI extended with arbitrary function symbols). By devising a suitable set of lemmas about $*$, it is possible to enable the decision procedure for T_c to decide formulae whose validity depends on properties of multiplication, e.g. multiplying two positive integers we obtain a positive integer. However, augmentation can cope with user-defined function whose relevant arithmetic properties are expressed by suitable lemmas (e.g. formula (1) in Section 1). Let R be a set of T_j-valid formulae of the form $\bigwedge_i^n q_i \Rightarrow c$, where $c, q_1, ..., q_n$ are (possibly) non-ground literals.[7] We illustrate how augmentation works (in the presence of conditional lemmas) by means of an example.

We assume that R contains the T_j-valid formula encoding a simple property about the signs of multiplicands:

$$(X \leq 0 \wedge Y \geq 0) \Rightarrow X * Y \leq 0, \tag{20}$$

[7] Adapting what follows to unconditional lemmas is trivial and therefore it is not discussed.

where X and Y are implicit universally quantified integer variables. We want to prove that the following formula is $\mathcal{L}(T_j)$-valid:

$$a \leq -1 \wedge b \leq 0 \wedge -c \leq 0 \Rightarrow (a+b)*c \not> 0 \tag{21}$$

(where $A \not> B$ denotes the negation of $A > B$).

By invoking simp on the negation of (21) and the empty initial state, we obtain the state S':[8]

$$a \leq -1 \wedge b \leq 0 \wedge -c \leq 0 \wedge -(a+b)*c \leq -1. \tag{22}$$

It is easy to see (by reasoning about the sign of multiplicands) that S' is T_j-unsatisfiable but the decision procedure for T_c is unable to decide the T_j-unsatisfiability of S' since it does not know anything about multiplication.

The idea underlying augmentation is to extend the state of the decision procedure with the conclusion of selected instances of lemmas in R, so to enable the decision procedure to derive a T_c-unsatisfiable state. To add the instantiated conclusions of the formulae in R, we must first relieve their (instantiated) conditions. Consider the instance of (20) obtained by applying the substitution $\{(a+b)/X, c/Y\}$, i.e. $(a+b \leq 0 \wedge c \geq 0) \Rightarrow (a+b)*c \leq 0$. Conditions $a+b \leq 0$ and $c \geq 0$ are easily found to be T_j-entailed by S'. This can be done by adding their negations to S' with simp and check for the T_c-unsatisfiability of the resulting states with unsat. Then, we are entitled to add $(a+b)*c \leq 0$ to S', obtaining (by invoking simp again) the new state S'', i.e. $a \leq -1 \wedge b \leq 0 \wedge -c \leq 0 \wedge -(a+b)*c \leq -1 \wedge (a+b)*c \leq 0 \wedge 0 \leq -1$, which is readily found unsatisfiable by unsat.

Example 2. (Extending the Decision Procedure of Example 1: Augmentation.) The crucial step for the success of augmentation is the selection of suitable instances of formulae in R. A sensible heuristics is to find instances of the conclusions of formulae in R promoting further Fourier-Motzkin elimination steps when added to the current state of the decision procedure. We define genlemma as follows:

$$\mathsf{genlemma}(C_M) \;=\; \langle C_{\{m\}\cup M}, [\gamma_1, ..., \gamma_s] \rangle \tag{23}$$

where γ_k $(k = 1, ..., s)$ is of the form $\bigwedge_{i=1}^{n} q_i\sigma \Rightarrow c\sigma$, m is the heaviest multiplicand of $C|_M$, $\bigwedge_{i=1}^{n} q_i \Rightarrow c$ is in R, σ is a substitution, and

1. m is also the heaviest multiplicand of $c\sigma$, and its coefficients in $C|_M$ and in $c\sigma$ are of opposite sign,
2. $c\sigma$ and $q_i\sigma$ $(i = 1, ..., n)$ are ground, and
3. $q_i\sigma \prec^m c\sigma$ $(i = 1, ..., n)$.

In order to ensure condition 2 above, we require that for any lemma $\bigwedge_{i=1}^{n} q_i \Rightarrow c$ in R all the variables in $\bigwedge_{i=1}^{n} q_i$ also occur in the heaviest multiplicand of c. For a more general technique allowing extra variables in the conditions of the lemmas in R see [BM88].

[8] We have exploited the following transformation of formulae: for every integer expression A, replace $A > 0$ with $-A \leq -1$.

The intuition underlying abstract requirement (5) is exemplified by the definition above. In fact, genlemma adds the heaviest multiplicand m to the set M of already eliminated multiplicands in order to allow the processing of the conditions of the lemmas which would not be possible otherwise (recall the definition of simp in Example 1 and consider requirement 3 above, which forces each instantiated condition to be $\prec^m$-smaller than $c\sigma$). From (23) and the fact that m is not in M, we have $C_{\{m\}\cup M} \prec C_M$ since m is the heaviest multiplicand in $C|_M$ and the heaviest multiplicand in $C|_{\{m\}\cup M}$ is $\prec^m$-smaller than m. Hence, (5) is satisfied. $\mathsf{simp}(c\sigma, C_{\{m\}\cup M}) \prec C_M$ holds since the heaviest multiplicand in $\mathsf{simp}(c\sigma, C_{\{m\}\cup M})$ is $\prec^m$-smaller than m. Thus (7) is satisfied. Furthermore, we have $\mathsf{simp}(\neg q_i\sigma, C_{\{m\}\cup M}) \prec C_M$ $(i = 1, ..., n)$. There are two cases to be considered. First, the heaviest multiplicand of $\neg q_i\sigma$ is m. Then the heaviest multiplicand of the state resulting from $\mathsf{simp}(\neg q_i\sigma, C_{\{m\}\cup M})$ is $\prec^m$-smaller than m. Second, the heaviest multiplicand of $\neg q_i\sigma$ is $\prec^m$-smaller than m. Then, $\mathsf{simp}(\neg q_i\sigma, C_{\{m\}\cup M}) = C_{\{m\}\cup M}$ (since no multiplicand can be eliminated) and we conclude by (5). Thus (6) is also satisfied. $\triangle$

3.2 Affinization

If a suitable set R of lemmas is defined, augmentation increases dramatically the effectiveness of the decision procedure. Unfortunately, devising a suitable R (especially for multiplication) is a time consuming activity, which requires a good understanding of the internal workings of augmentation. We now describe an instance of genlemma which allows for the 'on-the-fly' generation of lemmas about multiplication. We illustrate how affinization works by means of an example.

Consider again formula (21). By invoking simp on the negation of (21) and the empty initial state, we obtain (22). The last literal in the conjunction, namely $(a+b)*c \leq -1$, is a *hyperbolic inequality*, i.e. an inequality of the form $s*t \leq k$ where s and t are terms in $\mathcal{L}(T_c)$ and k is an integer. By resorting to its geometrical interpretation, it is easy to verify that $s*t \leq -1$ is T_j-equivalent to $(s \geq 1 \wedge t \leq -1) \vee (s \leq -1 \wedge t \geq 1)$. Unfortunately, this formula is a disjunction and the decision procedure for T_c can handle only conjunctions of literals. However, since the semi-planes represented by $s \geq 1$ and $s \leq -1$ as those represented by $t \leq -1$ and $t \geq 1$ are non-intersecting, we can derive the following four T_j-entailed formulae: $s \geq 1 \Rightarrow t \leq -1$, $s \leq -1 \Rightarrow t \geq 1$, $t \geq 1 \Rightarrow s \leq -1$, and $t \leq -1 \Rightarrow s \geq 1$. After instantiating $s \leq -1 \Rightarrow t \geq 1$ with the substitution $\{(a+b)/s, c/t\}$, we are entitled to add its conclusion (namely $c \geq 1$) to S', since the instantiated condition (namely $a+b \leq -1$) is found T_j-entailed by S'. Thus, we obtain a new state S'':

$$a \leq -1 \wedge b \leq 0 \wedge -c \leq 0 \wedge (a+b)*c \leq -1 \wedge c \leq -1 \wedge 0 \leq -1$$

which is readily found T_c-unsatisfiable by unsat.

In the general case, it turns out that any inequality of the form $s*t \leq k$ (where s and t are multiplicands and k is a strictly positive integer) is T_j-equivalent to a formula of the form

$$(s \geq 1 \wedge t \geq 1 \wedge \bigwedge_{i=1}^{m} \alpha_i) \vee (s \leq -1 \wedge t \leq -1 \wedge \bigwedge_{j=1}^{n} \beta_j) \tag{24}$$

where α_j $(i = 1, ..., m)$ and β_j $(j = 1, ..., n)$ are inequalities which are linear in s and t, m and n are natural numbers that depends on the value of k. More precisely, the number of literals generated is $O(k^{1/2})$. Fortunately, k is small for inequalities found in practical verification efforts. To be useful, (24) must be "compiled" into the following four lemmas:

$$\begin{array}{cc} s \geq 1 \Rightarrow \bigwedge_{j=1}^{m} \alpha_j & s \leq -1 \Rightarrow \bigwedge_{j=1}^{n} \beta_j \\ t \geq 1 \Rightarrow \bigwedge_{j=1}^{m} \alpha_j & t \leq -1 \Rightarrow \bigwedge_{j=1}^{n} \beta_j, \end{array} \tag{25}$$

by exploiting the disjointness of the semi-planes $s \geq 1$ and $s \leq -1$, as well as $t \geq 1$ and $t \leq -1$. Similar reductions are possible for *elliptical inequalities*, i.e. inequalities of the form $a * s * s + b * t * t \leq k$ where s and t are integer expressions, a, b, and k are integers. A general affinization theorem for convex areas described by inequalities in two variables and its application to classes of inequalities (among which hyperbolic ones) is described in [MP94].

Example 3. (Extending the Decision Procedure of Example 1: Affinization.) Below, we consider hyperbolic inequalities only. Extensions to other classes of inequalities are trivial. The implementation of genlemma is in three steps, i.e.

$$\mathsf{genlemma}(C_M) \;=\; \langle C_{\{m\} \cup M}, \mathsf{compile}(\mathsf{affinize}(\mathsf{normalize}(C|_M))) \rangle$$

where m is the heaviest multiplicand in $C|_M$ s.t. it is not in M and normalize selects all inequalities with heaviest multiplicand m in $C|_M$ and transforms them into hyperbolic inequalities if possible (for instance $a * c + b * c \not\geq 0$ is transformed into $(a + b) * c \not\geq 0$); affinize takes a hyperbolic inequality and returns the application of the affinization theorem to such an inequality, i.e. it returns an instance of (24); compile transforms the affinized inequality into a list of T_j-entailed formulae of the form $s \geq 1 \Rightarrow \bigwedge_i \alpha_i$ or $t \geq 1 \Rightarrow \bigwedge_j \beta_j$, where m is $s * t$.

Let us assume that $\prec^m$ is s.t. $s \prec^m s + t$, $s \prec^m s + t$, $s \prec^m s * t$, and $s \prec^m s * t$ for any two terms s and t. Then, (5) holds for the same argument given in Example 2. (6) and (7) hold because the heaviest multiplicand in α_i $(i = 1, ..., m)$, β_j $(i = 1, ..., n)$, $\neg(s \geq 1)$, and $\neg(t \geq 1)$ is $\prec^m$-smaller than m (since $s \prec^m m$ and $t \prec^m m$) and the heaviest multiplicand in $C|_{\{m\} \cup M}$ is $\prec^m$-smaller than m.

The crucial step is normalize, i.e. the transformation of an inequality into a hyperbolic inequality. This transformation can be seen as the problem of finding two (non-constant) multivariate polynomials s and t s.t. the polynomial obtained by multiplying s and t is equal to the l.h.s. of ι. Indeed, if we are able to factorize multivariate polynomials, we can easily solve the problem of finding s and t. Unfortunately, no algorithm is known that runs in time polynomial in the length of the adopted (sparse) representation (see [vzGG99] for a discussion). One way to cope with this problem is to use some heuristic manipulations to transform an inequality into the desired form (see [MP94] for more on this). $\triangle$

3.3 Combining Augmentation and Affinization

If on the one hand affinization can be seen as a significant improvement over augmentation since it does not require any user intervention, on the other hand it fails to apply when inequalities cannot be transformed into a form suitable for affinization. It turns out that augmentation and affinization can profitably be combined to obtain the flexibility of the former with the automation of the latter. We illustrate the combination on a well-known example drawn from the literature.

In [BM79], Boyer and Moore present the mechanical verification of a propositional tautology checker for conditional expressions. A key ingredient in the definition of such a checker is the recursive function *norm*, which puts conditional expressions into normal form. Since Boyer and Moore's logic admits a recursive function definition only if its termination is proved, we must verify that *norm* terminates on all possible conditional expressions. A well-known argument for proving the termination of *norm* is based on exhibiting a measure function that decreases (according to a given ordering) at each *norm*'s recursive call. One such a function, called ms, is reported in [Pau86]. Then, the task is to check whether ms is decreasing (w.r.t. the $<$ relation over integers) at each *norm*'s recursive call. $0 < ms(u) * ms(y) + ms(u) * ms(z)$ is one of the proof obligations formalizing the fact that ms is decreasing.

We assume that R contains the formula $0 < ms(E)$, where E is a variables ranging over conditional expression. At the beginning, the state of the decision procedure is S, i.e. $ms(u) * ms(y) + ms(u) * ms(z) \leq 0$, which is T_j-unsatisfiable but unsat is unable to detect it. First, we resort to affinization. The factorization of the literal in S is $ms(u) * (ms(y) + ms(z)) \leq 0$. Then, it is possible to generate a list of T_j-entailed formulae containing $0 \leq ms(u) \Rightarrow ms(y) + ms(z) \leq 0$. To relieve the condition of the formula, namely $0 \leq ms(u)$, augmentation is required. In fact, it is immediately relieved if we consider $0 < ms(u)$ as an instance of the available lemma in R. Then, we are entitled to add the conclusion of the speculated fact, namely $ms(y) + ms(z) \leq 0$, to S and we obtain the state S', i.e. $ms(u) * ms(y) + ms(u) * ms(z) \leq 0 \wedge ms(y) + ms(z) \leq 0$. We resort to augmentation by instantiating twice the available lemma with substitutions $\{z/E\}$ and $\{y/E\}$, obtaining the state S'', i.e. $ms(u) * ms(y) + ms(u) * ms(z) \leq 0 \wedge ms(y) + ms(z) \leq 0 \wedge -ms(z) \leq -1 \wedge -ms(y) \leq -1$ (notice that $ms(y) > 0$ and $ms(z) > 0$ have been transformed into $-ms(y) \leq -1$ and $-ms(z) \leq -1$, respectively, as suggested in footnote 8). Finally, we eliminate the multiplicands $ms(y)$ and $ms(z)$ in S'' obtaining the state $ms(u) * ms(y) + ms(u) * ms(z) \leq 0 \wedge ms(y) + ms(z) \leq 0 \wedge -ms(z) \leq -1 \wedge -ms(y) \leq -1 \wedge ms(z) \leq -1 \wedge 0 \leq -2$, which is readily found T_c-unsatisfiable by unsat.[9]

The realization of this instance for genlemma can easily be gleaned from Examples 2 and 3, and the observation that the top-most function symbol of the heaviest multiplicand (in the state of the procedure) triggers the invocation of either affinization or augmentation.

[9] It is worth noticing that if the following lemma $(X \geq 0 \wedge Y \geq 0) \Rightarrow X * Y \geq 0$ is also in R, then $0 < ms(u) * ms(y) + ms(u) * ms(z)$ can be decided by augmentation alone.

#	S	Problem	Results		
			Au	Af	AA
1	[Pau86]	$\vdash\ ms(x) > 0 \wedge ms(y) > 0 \wedge ms(z) > 0 \Rightarrow ms(x) + ms(x) * ms(y) + m(x) * ms(z) > 0$	–	$\surd$	$\surd$
2	[Pau86]	$ms(E) > 0\ \vdash\ 0 < 1 + m(z)$	$\surd$	–	$\surd$
3	[Pau86]	$ms(E) > 0\ \vdash\ ms(u) * ms(y) + ms(u) * ms(z) > 0$	–	–	$\surd$
4	[BM88]	$0 < I \Rightarrow J \leq I * J,\ ms(X) > 0\ \vdash$ $ms(a)^2 < 2 * ms(a)^2 * ms(b) + ms(a)^4$	$\surd$	–	$\surd$
5	[BM88]	$ms(X) > 0\ \vdash\ ms(a)^2 < 2 * ms(a)^2 * ms(b) + ms(a)^4$	–	–	$\surd$
6	[Bjø98]	$\vdash\ m \leq l + d * r \wedge r < 0 \wedge x + t \leq d \wedge t > 0 \Rightarrow m \leq l + r * x + r * t$	–	$\surd$	$\surd$
7	[Har96]	$\vdash\ x \geq 0 \Rightarrow x^2 - x + 1 \neq 0$	–	$\surd$	$\surd$
8	[Har96]	$\vdash\ x \leq 0 \Rightarrow x^2 - x + 1 \neq 0$	–	$\surd$	$\surd$
9	[Har96]	$\vdash\ x > 4 \Rightarrow x^2 - 5 * x + 6 \geq 0$	–	$\surd$	$\surd$
10	[BM88]	$\delta_1(PAT, LP, C) \leq LP\ \vdash\ lp + lt \leq maxint \wedge i \leq lt \Rightarrow i + \delta_1(pat, lp, c) \leq maxint$	$\surd$	–	$\surd$

Table 1: Experimental Results

4 Experimental Results

Our extension schema must be judged w.r.t. its effectiveness in enabling a decision procedure to decide proof obligations arising in practical verification efforts. We discuss the instances described in Section 3 by running three prototype implementations on a set of proof obligations extracted from the literature. The prototypes are coded in **RDL**, a system for simplifying formulae in (extensions of) quantifier-free first-order logic with equality.[10] The functionalities simp and unsat implement a rational based version of the Fourier-Motzkin elimination method (as described in Example 1).

Table 1 reports the results of our computer experiments. S refers to the source of the problem. Results records the successful ($\surd$) or the unsuccessful (–) attempt to solve a problem. Au, Af, and AA refer to our implementation of augmentation, affinization, and their combination (respectively). Problem lists the available lemmas[11] (if any) and the formula to be decided. $\vdash$ is the binary relation characterizing the deductive capability of the extended decision procedure (we have that $\vdash$ is contained in $\models_{T_j}$). The selected problems are difficult for decision procedures integrated in current state-of-the-art verification systems. As a matter of fact, the online version of STeP fails to solve all of them. However, most of the problems are successfully solved by the improved version of STeP described in [Bjø98]. Furthermore, the selected problems are representative of various verification scenarios. Problems 1, 2, 3, 4, and 5 are proof obligations arising in establishing the termination of recursive functions

[10] **RDL** distribution can be accessed via the Constraint Contextual Rewriting Home Page, `http://www.mrg.dist.unige.it/ccr`.

[11] Capitalized letters denote implicitly universally quantified variables.

and of term rewriting systems using the method of polynomial orderings (see, e.g., [DJ90]). Problem 6 is the by-product of solving differential equations (this relevant to the verification of hybrid systems, see e.g. [Bjø98]). Problems 7, 8, and 9 involve quadratic forms which can be found in routine algebraic manipulations. Problem 10 is a proof obligation extracted from the verification of a string matching algorithm.

Our experiments show the flexibility of augmentation (cf. problems 2, 4, and 10), the high degree of automation achieved by affinization (cf. problems 1, 6, 7, 8, and 9), and the power of their combination (cf. problems 3 and 5). Let us compare problems 4 and 5. The lemma about multiplication (i.e. $0 < I \Rightarrow J \leq I * J$) is supplied in problem 4 but it is not in problem 5. Augmentation solves problem 4 thanks to the supplied lemmas, but affinization fails to do so since it is not able to use the 'positivity lemma' $ms(X) > 0$. Only the combination of augmentation and affinization can solve problem 5. This suggests that ACL2 [KM97] and its predecessor NQTHM featuring only augmentation can greatly be enhanced by our affinization technique. Let us turn to problem 6. STeP [M+94] solves it by resorting to a partial method for quantifier elimination (see [Bjø98] for details). Instead, our extension schema is able to prove the formula with simpler mathematical techniques. The comparison is somewhat difficult since the method used by STeP works over the rationals and our affinization technique only work over integers. However, our affinization technique can be used also over rationals to approximate classes of non-linear inequalities. Finally, without $x \geq 0$ ($x \leq 0$) affinization would not be able to solve problem 7 (8, resp.). This shows the importance of the context in which proof obligations are proved.

The timings are all within 1.7 sec and in most cases the order of magnitude is 1/10 of a second.[12] Problem 4 takes about 1.7 sec since augmentation is repeatedly invoked. Problem 5 takes about 1 sec since affinization speculates lemmas which are readily used by the decision procedure. In some situations, the combination of augmentation and affinization may result in better performances besides a higher degree of automation.

5 Related Work

In ACL2 (as in its predecessor, NQTHM [BM79]) the augmentation mechanism is tailored to a decision procedure for Presburger Arithmetic, which closely resembles that described in Example 1. The termination of the resulting schema is not discussed in [BM88]. In [AR98, AR00], the authors describe Constraint Contextual Rewriting (CCR), a terminating schema for the incorporation of rewriting with a decision procedure and augmentation. In CCR, augmentation is not seen in isolation and it is the only possible form of lemma speculation. Both in ACL2 and in CCR, it is possible to relieve hypotheses falling outside the domain of the extended decision procedure since a full-fledged rewriter is invoked. *Simplify* [DNS96] features cooperating decision procedures (following the paradigm proposed in [NO78]). It implements a form of augmentation based on a heuristic matching algorithm which finds suitable instances of universally quantified formulae but it is not guaranteed to terminate.

[12] Benchmarks run on a 400 MHz Pentium II running Linux. **RDL** was compiled using Sicstus Prolog, version 3.8.

PVS features tightly coupled decision procedures (following the paradigm proposed in [Sho84]). A mechanism to access a data base of lemmas containing a large number of lemmas about multiplication is implemented. However, PVS—in most situations—must be manually guided to find suitable instances of these lemmas. This activity can be frustrating since the user is asked to scan through quite a large number of available formulae. Our affinization technique provides the desired degree of automation. The version of STeP described in [Bjø98] implements a rational based version of the Fourier-Motzkin method, extended to handle multiplication by (partial) quantifier elimination and reasoning about the sign of multiplicands. A similar procedure can be obtained as an instance of our extension schema by using 'compilation' techniques to eliminate disjunctions (in the same spirit of compile in Section 3.2).

Theory reasoning (see [BFP92] for an overview) is an elegant deduction paradigm which gives a unified view of various logical calculi. Our extension schema can be seen as an instance of incremental literal level theory reasoning where the background reasoner plays the role of our extensible decision procedure and the speculated lemmas are added to the formula to be proved. However, the goal of our work is pragmatic in that the proposed extension schema aims at lifting decision procedures currently available in state-of-the-art verification systems. In [Har96], Harrison describes the LCF implementation of a quantifier elimination procedure for the elementary theory of reals (including multiplication). Since the theoretical complexity of such a procedure greatly restricts its usability, Harrison devises an optimized version for the fragment of Universal Presburger Arithmetic. Our affinization technique (and more in general our extension schema) takes the opposite approach to lift a decision procedure for Universal Presburger Arithmetic to a proof procedure for Universal Arithmetic.

SoleX [MR98] is a mechanism for the domain independent extension of constraint solvers so to deal with programmer-defined constraints. This work resembles augmentation in allowing interpreted function symbols by means of guarded rewrite rules and in being domain independent. More precisely, SoleX can be seen as an instance of augmentation where no recursive invocation of the mechanism is allowed to relieve conditions of lemmas.

6 Conclusions

We have presented an extension mechanism which enables decision procedures to tackle problems falling outside the scope they have been originally designed for. We have shown that the schema is both sound and terminating. We have presented three instances of the mechanism and shown that they enable a decision procedure for the universal fragment of Presburger Arithmetic over integers to tackle non-linear problems of significant difficulty. Computer experiments confirm the validity of the proposed approach. We plan to incorporate the extension mechanism in CCR so to enhance the deductive capabilities of the resulting simplification process.

References

[AR98] A. Armando and S. Ranise. Constraint Contextual Rewriting. In *Proc. of the 2nd International Workshop on First Order Theorem Proving (FTP'98)*, Vienna, Austria, November 1998.

[AR00] A. Armando and S. Ranise. Termination of Constraint Contextual Rewriting. In *Proc. of the 3rd International Workshop on Frontiers of Combining Systems (FroCos'2000)*, Nancy, France, March 2000. Lecture Notes in Computer Science, Volume 1794, pages 47–61.

[ARR00] A. Armando, S. Ranise, and M. Rusinowitch. Generic Derivation of Congruence Closure Algorithms. Technical report, LORIA-INRIA-Lorraine & DIST-Università di Genova, 2000. In preparation.

[BFP92] P. Baumgartner, U. Furbach, and U. Petermann. A Unified Approach to Theory Reasoning. Technical report, Inst. für Informatik (Koblenz), 1992.

[Bjø98] N. S. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Computer Science Department, Stanford University, 1998.

[BM79] R.S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979.

[BM88] R.S. Boyer and J S. Moore. Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic. *Machine Intelligence*, 11:83–124, 1988.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewriting systems. In *Handbook of Theoretical Computer Science*, pages 243–320. 1990.

[DNS96] D. L. Detlefs, G. Nelson, and J. Saxe. Simplify: the ESC Theorem Prover. Technical report, DEC, 1996.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Pr., 1972.

[Har96] J. R. Harrison. *Theorem Proving with the Real Numbers*. PhD thesis, Computer Laboratory, University of Cambridge, 1996.

[KM97] M. Kaufmann and J S. Moore. Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. on Software Engineering*, 23(4):203–213, 1997.

[KMN94] D. Kapur, D.R. Musser, and X. Nie. An Overview of the Tecton Proof System. *Theoretical Computer Science*, Vol. 133, October 1994.

[KN94] D. Kapur and X. Nie. Reasoning about Numbers in Tecton. In *Proc.* 8^{th} *Inl. Symp. Methodologies for Intelligent Systems*, pages 57–70, 1994.

[MP94] V. Maslov and W. Pugh. Simplifying Polynomial Constraints Over Integers to Make Dependence Analysis More Precise. Technical Report CS-TR-3109.1, Dept. of Computer Science, University of Maryland, 1994.

[MR98] E. Monfroy and C. Ringeissen. SoleX: a Domain-Independent Scheme for Constraint Solver Extension. In 4^{th} *Intl. Conf. on Artificial Intelligence and Symbolic Computation (AISC'98)*, pages 222–233, 1998.

[M+94] Z. Manna and the STeP Group. STeP: The Stanford Temporal Prover. Technical Report CS-TR-94-1518, Stanford University, June 1994.

[Nel81] G. Nelson. Techniques for Program Verification. Technical Report CSL-81-10, Xerox Palo Alto Research Center, June 1981.

[NO78] G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford Computer Science Department, April 1978.

[Pau86] L. C Paulson. Proving termination of normalization functions for conditional expressions. *J. of Automated Reasoning*, pages 63–74, 1986.

[PB94] G. Pesant and M. Boyer. QUAD-CLP($\mathcal{R}$): Adding the power of Quadratic Constraints . In *Proc. of PPCP'94: 2nd Workshop on Principles and Practice of Constraint Programming*, 1994.

[Sho84] R.E. Shostak. Deciding Combination of Theories. *J. of the ACM*, 31(1):1–12, 1984.

[SJ80] N. Suzuki and D. Jefferson. Verification Decidability of Presburger Array Programs. *J. of the ACM*, 27(1):191–205, January 1980.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.