A Framework for the Flexible Integration of a Class of Decision Procedures into Theorem Provers *

Predrag Janičić[1], Alan Bundy[2], and Ian Green[2]

[1] janicic@matf.bg.ac.yu
Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Yugoslavia
[2] {A.Bundy,I.Green}@ed.ac.uk
Division of Informatics, University of Edinburgh
Edinburgh EH1 1HN, Scotland

Abstract. The role of decision procedures is often essential in theorem proving. Decision procedures can reduce the search space of heuristic components of a prover and increase its abilities. However, in some applications only a small number of conjectures fall within the scope of the available decision procedures. Some of these conjectures could in an informal sense fall 'just outside' that scope. In these situations a problem arises because lemmas have to be invoked or the decision procedure has to communicate with the heuristic component of a theorem prover. This problem is also related to the general problem of how to flexibly integrate decision procedures into heuristic theorem provers. In this paper we address such problems and describe a framework for the flexible integration of decision procedures into other proof methods. The proposed framework can be used in different theorem provers, for different theories and for different decision procedures. New decision procedures can be simply 'plugged-in' to the system. As an illustration, we describe an instantiation of this framework within the Clam *proof-planning* system, to which it is well suited. We report on some results using this implementation.

1 Introduction

Decision procedures have a very important role in heuristic theorem provers — they can reduce the search space of heuristic components of a prover and increase its abilities. Decision procedure can both close a branch in a proof and reject non-theorems. Some decision procedures are (generally) inefficient, but even they could increase abilities of a prover. Some decision procedures, such as a decision procedure for Presburger arithmetic [18], can have a useful role in proving some typical conjectures occurring in software and hardware verification. In some applications, decision procedures themselves cannot significantly improve the overall efficiency of a prover; for example, when only a small number of conjectures fall within the domain of the available decision procedures.

* The second and the third author are supported in part by EPSRC grant GR/M45030.

However, some (or many) of these conjectures could be in a sense “close” to that domain (see [3]). For instance, the formula

$$\forall l\ \forall\alpha\ \forall k \quad (l \leq min(\alpha)\ \wedge\ 0 < k \quad\rightarrow\quad l < max(\alpha) + k)$$

is not a Presburger arithmetic formula. Besides, the formula $\forall max \forall min \forall l \forall k\ (l \leq min \wedge\ 0 < k \ \rightarrow\ l < max + k)$ obtained by generalising $min(\alpha)$ to min and $max(\alpha)$ to max is a Presburger arithmetic formula, but is not a theorem. The power of decision procedures can be increased by linking them to heuristic components of the prover so they can communicate; by using already proved lemmas and by combining different decision procedures. For instance, in the given example, if the lemma $\forall\xi\ (min(\xi) \leq max(\xi))$ is available, it can be used (with the right instantiation) and lead to $\forall l \forall\alpha \forall k\ (min(\alpha) \leq max(\alpha) \rightarrow (l \leq min(\alpha) \wedge\ 0 < k\ \rightarrow\ l < max(\alpha) + k))$. After generalisation, we get the formula $\forall max \forall min \forall l \forall k\ (min \leq max \rightarrow (l \leq min \wedge\ 0 < k\ \rightarrow\ l < max + k))$ which can be proved by the decision procedure for the Presburger arithmetic. Procedures dealing with such problems are built into most state-of-the-art theorem proving systems. In [3] there is a description of one such system — a procedure for linear arithmetic based on Hodes' algorithm. However, in that paper, implementation details are mixed up with a description of the underlying algorithm. Moreover, the system itself depends to a high degree on certain specific data structures and can hardly be described without these devices. All this makes the system from [3] inapplicable in theories other than linear arithmetic. Besides, even for linear arithmetic, it is not obvious how a procedure other than that due to Hodes' could be integrated into the prover.

In this paper we give a kind of rational reconstruction of some of the approaches presented in [3]. We present a general method that can incorporate different decision procedures into a heuristic theorem prover. This general method is flexible in its structure and can be used for different decision procedures, for different theories and in different theorem provers. Within this modular system, the new decision procedure (with only a syntactical description of a corresponding theory) can be ‘plugged-in’ into a prover without requiring any theoretical analysis of the corresponding theory. This method is rather general and domain specific knowledge is encapsulated in smaller subprocedures specialised for certain theories.

Overview of the paper. In section 2, we give some notation and background; we introduce the notion of a *heaviest non-$\mathcal{T}$-term* in section 3; in section 4 we give a description of modules that make up the extended proof method; in section 5 we define the proposed general method and in section 6 we consider one example from [3] in order to illustrate how the proposed method works. Section 7 discusses the termination, the soundness and some other properties of the proposed method and in section 8 we give some results of the preliminary implementation of the method. Section 9 discusses some possible refinements. In section 10 we discuss related work and in section 11 we draw some final conclusions.

2 Background and Notation

Let us define a theory $\mathcal{T}$ within first order logic with equality. Let T be some fixed set of types $\{\tau_i | i \in I\}$. For each type τ_i let V_i be a denumerable set of variables of that type ($V_i \cap V_j = \emptyset$ for $i \neq j$) and let $V = \cup_{i \in I} V_i$. Let Σ be a finite set of function symbols, each having either a type τ, $\tau \in T$ (then we call it a *constant* of type τ) or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \tau$ ($\tau_{i_1}, \ldots, \tau_{i_k}, \tau \in T$). Let Π be a finite set of predicate symbols, each having a type `truth` (with members `true` and `false`) or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow$ `truth` ($\tau_{i_1}, \ldots, \tau_{i_k} \in T$). We define the notion of a $\mathcal{T}$*-term* in the following way: function symbols and variables of type τ are $\mathcal{T}$-terms of type τ; if $t_1, t_2, \ldots, t_n$ are $\mathcal{T}$-terms of types $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}$ respectively and a function symbol $f \in \Sigma$ has a type $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow \tau$, then $f(t_1, t_2, \ldots, t_n)$ is a $\mathcal{T}$-term of type τ; all terms could be obtained using the first two rules. For a term which is a variable x, the set of free variables is $\{x\}$; for a term $f \in \Sigma$ the set of free variables is empty; for $f(t_1, t_2, \ldots, t_n)$ the set of free variables is the union of free variables in $t_1, t_2, \ldots, t_n$. A $\mathcal{T}$*-atomic formula* is either an expression $p(t_1, t_2, \ldots, t_n)$ (where $t_1, t_2, \ldots, t_n$ are $\mathcal{T}$-terms of types $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}$ respectively and $p \in \Pi$ is a predicate symbol of type $\tau_{i_1} \times \tau_{i_2} \times \ldots \times \tau_{i_k} \rightarrow$ `truth`) or an expression $t_1 =_\tau t_2$ (where t_1 and t_2 are $\mathcal{T}$-terms of type τ) or a constant of type `truth`. For $p(t_1, t_2, \ldots, t_n)$ the set of free variables is the union of free variables in $t_1, t_2, \ldots, t_n$; for $t_1 =_\tau t_2$ the set of free variables is the union of free variables in t_1 and t_2. If F_1 is a $\mathcal{T}$-atomic formula, then it is also a $\mathcal{T}$*-formula.* If F_1 and F_2 are $\mathcal{T}$-formulae and $x \in V$, then also $\forall x\ F_1$, $\exists x\ F_1$, $\neg F_1$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ are $\mathcal{T}$-formulae. The set of free variables in $\forall x\ F_1$ and $\exists x\ F_1$ is the set of free variables in F_1 minus $\{x\}$. The set of free variables in $\neg F_1$ is the set of free variables in F_1. The set of free variables in $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ is the union of sets of free variables in F_1 and F_2. A $\mathcal{T}$-formula with no free variables we call a *closed* $\mathcal{T}$*-formula* or a $\mathcal{T}$*-sentence.* Further, by $\mathcal{T}$-formulae we will mean closed $\mathcal{T}$-formulae (unless stated otherwise). A formula F is *ground* if it has no variables. A formula F is in *prenex normal form* if it is of the form $Q_1 x_1\ Q_2 x_2\ \ldots Q_k x_k\ F'$ where $Q_i \in \{\forall, \exists\}$, $x_i \in V$ and there are no quantifiers in F'. A formula F is *universally closed* if it is of the form $\forall x_1\ \forall x_2\ \ldots \forall x_k\ F'$ where $x_i \in V$ and there are no quantifiers in F'. A formula appearing in a formula F has a *polarity* that is either positive (+) or negative (−). The top-level formula F being proved has positive polarity, written F^+. The complement of a polarity p is written $\overline{p}$, defined to be $\overline{+} = -$ and $\overline{-} = +$. Polarity is defined recursively over the structure of formulae: $(\neg F^{\overline{p}})^p$ (if p is the polarity of $\neg F$ then $\overline{p}$ is the polarity of F), $(\forall x\ F^p)^p$, $(\exists x\ F^p)^p$, $(F_1^p \wedge F_2^p)^p$, $(F_1^p \vee F_2^p)^p$, $(F_1^{\overline{p}} \rightarrow F_2^p)^p$.

A *theory* $\mathcal{T}$ (or an *axiom system* $\mathcal{T}$) is some fixed set of $\mathcal{T}$-formulae. If a formula F can be derived from that set using usual classical logic inference system we denote that by $\mathcal{T} \vdash F$ and we call the formula F a $\mathcal{T}$*-theorem* (and we say that F is *valid* in $\mathcal{T}$). Otherwise, we write $\mathcal{T} \not\vdash F$ and we say that F is *invalid* in $\mathcal{T}$.

Let Σ^e, Π^e and V^e be some finite sets of function, predicate and variable symbols over some set of types T^e such that $\Sigma \subseteq \Sigma^e$, $\Pi \subseteq \Pi^e$, $V \subseteq V^e$ and

$T \subseteq T^e$. We define the notions of $\mathcal{T}^e$-term, $\mathcal{T}^e$-atomic formula, $\mathcal{T}^e$-formula and a theory $\mathcal{T}^e$ by analogy. If all formulae of the theory $\mathcal{T}$ belong to the theory $\mathcal{T}^e$, we say that $\mathcal{T}^e$ is an extension of the theory $\mathcal{T}$ (or the theory $\mathcal{T}$ is a subtheory of the theory $\mathcal{T}^e$).[1]

A theory $\mathcal{T}$ is decidable if there is an algorithm (which we call a *decision procedure*) such that for an input $\mathcal{T}$-formula F, it returns `true` if and only if $\mathcal{T} \vdash F$ (and returns `false` otherwise). For a number of decidable theories there are decision procedures that work using the idea of successive elimination of quantifiers from formula being proved ([13]). When all quantifiers are eliminated, the formulae is ground and can be easily reduced to `true` or `false`. In this paper, we will be concerned with this kind of theory.

One of such theories is Presburger Natural Arithmetic (PNA). In this theory, all variables are of type `pnat` (from *Peano NATurals*), $\Sigma = \{0, s, +\}$, ($0 : \texttt{pnat}$, $s : \texttt{pnat} \rightarrow \texttt{pnat}$, $+ : \texttt{pnat} \times \texttt{pnat} \rightarrow \texttt{pnat}$), $\Pi = \{<, >, \leq, \geq\}$ (all the predicate symbols are of the type $\texttt{pnat} \times \texttt{pnat} \rightarrow \texttt{truth}$). We write 1 instead of $s(0)$, etc. Multiplication of a variable by a constant can also be considered as in PNA: nx is treated as $x + \cdots + x$, where x appears n times. The axioms of PNA are those of Peano arithmetic without axioms concerning multiplication.

Similarly, we introduce theories *Presburger Integer Arithmetic* (PIA) and *Presburger Real Arithmetic* (PRA).[2] It was Presburger who first showed that PIA is decidable [18]. The decidability of PNA can be proved in an analogous way. PRA is also decidable [13]. The arithmetic with multiplication only is also decidable. The whole of arithmetic is known to be undecidable.

3 Heaviest Non-$\mathcal{T}$-Term

A conjecture attempted to be proved by some specific decision procedure for some theory $\mathcal{T}$ would often "fall just outside its domain", i.e., a formula F being proved could be a $\mathcal{T}^e$-formula, but not a $\mathcal{T}$-formula. In that case and if $\Pi^e \setminus \Pi$ is empty, it would mean that F involves *non-$\mathcal{T}$-terms*. For now on, we will assume $\Pi^e \setminus \Pi = \emptyset$. Recall that all function symbols from Σ^e have either a type τ or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \cdots \times \tau_{i_k} \rightarrow \tau$, where $\tau \in T$, so the formula F' obtained in such a manner is a $\mathcal{T}$-formula.

Definition 1. *For $\mathcal{T}^e$-formulae and $\mathcal{T}^e$-atomic formulae, we define sets of* non-$\mathcal{T}$-terms *in the following way:*

- *the set of non-$\mathcal{T}$-terms in $\forall x\ F$, $\exists x\ F$, $\neg F$ is equal to the set of non-$\mathcal{T}$-terms in F;*

[1] Note that it can be determined whether a well-formed formula is a $\mathcal{T}$-formula or a $\mathcal{T}^e$-formula in linear time on the size of a formula.

[2] For identical and related theories a number of different terms are used. For instance, Hodes calls Presburger rational arithmetic a *theory EAR* — "the elementary theory of addition on the reals" [9]. Boyer and Moore [3] describe a universally quantified fragment of Presburger rational arithmetic as *linear arithmetic* (although in fact they work over the integers); that same theory sometimes goes by the name *Bledsoe real arithmetic*.

- *the set of non-$\mathcal{T}$-terms in $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$ is equal to the union of sets of non-$\mathcal{T}$-terms in F_1 and F_2;*
- *in $p(t_1, t_2, \ldots, t_n)$, where $p \in \Pi$, the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms t_j, $(j = 1, 2, \ldots, n)$;*
- *in $t_1 =_\tau t_2$, the set of non-$\mathcal{T}$-terms is the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms t_j, $(j = 1, 2)$;*
- *in* `true` *and* `false` *the set of non-$\mathcal{T}$-terms is empty.*

Definition 2. *For $\mathcal{T}^e$-terms, we define sets of* non-$\mathcal{T}$-terms *in the following way:*

- *if t is a $\mathcal{T}$-term, then its set of non-$\mathcal{T}$-terms is empty;*
- *in $f(t_1, t_2, \ldots, t_n)$, where $f \notin \Sigma$ (and $n \geq 0$), the set of non-$\mathcal{T}$-terms is $\{f(t_1, t_2, \ldots, t_n)\}$;*
- *in $f(t_1, t_2, \ldots, t_n)$, where $f \in \Sigma$ (and $n \geq 0$), the set of non-$\mathcal{T}$-terms is the union of sets of non-$\mathcal{T}$-terms in terms t_j $(j = 1, 2, \ldots, n)$.*

In deciding whether a $\mathcal{T}^e$-formula F is a theorem, our motivation is to use a decision procedure for $\mathcal{T}$ (either by using the decision procedure itself, or in a combination with some lemmas). Thus we have to somehow transform the formula F to some corresponding $\mathcal{T}$-formula. We do it by generalisation: in $\mathcal{T}^e$-formula F, we generalise non-$\mathcal{T}$-terms (from outside in) by new variables in the following way: we substitute each non-$\mathcal{T}$-term of a type τ by a new variable of the same type and then take the universal closure of the formula F. Recall that all function symbols from Σ^e have either a type τ or a type of the form $\tau_{i_1} \times \tau_{i_2} \times \cdots \times \tau_{i_k} \rightarrow \tau$, where $\tau \in T$, so the formula F' obtained in such a manner is a $\mathcal{T}$-formula (with a possible exception of some redundant quantifiers of some types not in T). For instance, the extended Presburger arithmetic formula $\forall \alpha\ (min(\alpha) \leq max(\alpha))$ (where α has type `list of pnats`) can be transformed to $\forall min \forall max \forall \alpha\ (min \leq max)$ (where min and max are of type `pnats`). In deciding whether F' is a theorem, we use a decision procedure for $\mathcal{T}$ by first eliminating all new variables (either by using the decision procedure itself, or in a combination with some lemmas). It is preferable to eliminate variables obtained from the most complicated terms. Thus we have to introduce some total ordering on $\mathcal{T}^e$-terms. First, we define a function $|\ .\ |$ that maps the set of $\mathcal{T}^e$-terms into the set of natural numbers (it will correspond to the *size* of a term).

Definition 3. *If a $\mathcal{T}^e$-term t is a function symbol of some atomic type τ $(\tau \in T)$ or a variable, then $|t| = 1$. If a $\mathcal{T}^e$-term t is of the form $f(t_1, t_2, \ldots, t_n)$, then $|t| = 1 + \sum_{i=1}^{n} |t_i|$.*

Definition 4. *A $\mathcal{T}^e$-term t_1 is* heavier *than a $\mathcal{T}^e$-term t_2 iff*

- *$|t_1| > |t_2|$ or*
- *$|t_1| = |t_2|$ and a dominant symbol of t_1 comes later in the lexicographic ordering than a dominant symbol of t_2 or*
- *$|t_1| = |t_2|$, $t_1 = f(t'_1, t'_2, \ldots, t'_n)$, $t_2 = f(t''_1, t''_2, \ldots, t''_n)$ and there is a value k $(1 \leq k \leq n)$ such that t'_i and t''_i are identical terms (for $i < k$) and t'_k is heavier than t''_k.*

Definition 5. *If terms t_1 and t_2 are identical or t_2 is heavier than t_1, then $t_1 \preceq t_2$.*

Definition 6. *A $\mathcal{T}^e$-term t is* the heaviest non-$\mathcal{T}$-term *in some set S of $\mathcal{T}^e$-terms, if t is a non-$\mathcal{T}$-term and for every non-$\mathcal{T}$-term t' from S it holds $t' \preceq t$.*

The relation $\preceq$ defines a total ordering on the set of $\mathcal{T}^e$ terms and this ordering fulfils the following condition: provided finitely many variables and function symbols, for each term t there are finitely many terms t' such that $t' \preceq t$. (This condition is important for the termination of the method proposed.)

4 Simplification Procedures

In this section we describe four procedures for simplification (and for reducing the number of quantifiers) of $\mathcal{T}^e$-formula being proved. Each of them can be used in any simplification context, in combination with other components of a prover and, finally, they can together build an extended proof method for $\mathcal{T}$. All of the procedures are applicable just to $\mathcal{T}^e$-formulae.

We assumed that there is a decision procedure for the theory $\mathcal{T}$ based on the idea of successive elimination of quantifiers. Thus, let us suppose that there are available two procedures[3] dealing with $\mathcal{T}$-formulae — $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\texttt{DpGround}_{\mathcal{T}}$:

— let $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ be the procedure which transforms a given non-quantifier free $\mathcal{T}$-formula[4] F to prenex normal form, eliminates the innermost quantifier (and the corresponding variable) and returns a formula F' such that it holds $\mathcal{T} \vdash F$ iff $\mathcal{T} \vdash F'$ and the number of quantifiers in the formula F' is fewer (or one less) than the number of quantifiers in the formula F.

— let $\texttt{DpGround}_{\mathcal{T}}$ be the procedure which for a given ground $\mathcal{T}$-formula F returns `true` if $\mathcal{T} \vdash F$ and returns `false` if $\mathcal{T} \not\vdash F$.

For many decidable theories there are such procedures ([13]).

For the following procedures, we restrict our consideration only to universally quantified conjectures and lemmas.

4.1 Decision Procedure for $\mathcal{T}$

Let the procedure $\texttt{Dp}_{\mathcal{T},A}$ be defined in the following way: if a formula being proved is ground, then apply $\texttt{DpGround}_{\mathcal{T}}$; otherwise, while a formula being proved is not ground, apply $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$.

Note that $\texttt{Dp}_{\mathcal{T},A}$ is a decision procedure for theory $\mathcal{T}$, i.e., $\texttt{Dp}_{\mathcal{T},A}$ reduces a $\mathcal{T}$-formula F to `true` if $\mathcal{T} \vdash F$ and to `false` otherwise.

The decision procedure for the theory $\mathcal{T}$ defined in this way is more flexible, but usually somewhat slower than some more compact procedures which

[3] Often these procedures can be also implemented in a flexible way: as the exhaustive applications of a series of rewrite rule sets (see [4])

[4] In F there could be some redundant quantifiers of some types not in T. This procedure ignores them (but does not eliminate them).

avoid unnecessary repetition of some algorithm steps (usually some unnecessary normalisations). Our extended proof method can also use some other decision procedure for the theory $\mathcal{T}$ instead of $\mathtt{Dp}_{\mathcal{T},A}$.

4.2 Elimination of Redundant Quantifier

The procedure `ElimRedundantQuantifier` for the elimination of a redundant quantifier is very simple: if there is a redundant quantifier (no matter of what type) in a $\mathcal{T}^e$-formula F being proved, eliminate it.

It is good if there are some rules for theory $\mathcal{T}$ such that we can use them to simplify the formula F before we try to apply `ElimRedundantQuantifier`. For instance, in PNA, we can reduce each atomic formula in such a way that it does not include two occurrences of the same term (for example, $k \leq max(a) + k$ can be rewritten to $0 \leq max(a)$).

4.3 Elimination of $\mathcal{T}$-variables

Let a procedure $\mathtt{ElimOneVar}_{\mathcal{T},A}$ be defined in the following way: if F is a $\mathcal{T}^e$-formula and if there is a variable v which does not appear in non-$\mathcal{T}$-terms of F, then generalise all its non-$\mathcal{T}$-terms (from outside in), then use the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ to eliminate the variable v (and the corresponding quantifier) and then substitute new variables by the original generalised terms.[5] (Note that this procedure is applicable to $\mathcal{T}$-formulae, but it is only sensible to use this procedure for formulae which are $\mathcal{T}^e$-formulae and not $\mathcal{T}$-formulae.)

4.4 Elimination of Generalised Non-$\mathcal{T}$-Term Using a Lemma

A procedure $\mathtt{ElimGeneralisedTerm}_{\mathcal{T},A}$ is defined (only) for universally quantified $\mathcal{T}^e$-formulae. It is defined in the following way:

- if a formula F being proved is a $\mathcal{T}^e$-formula and is not a $\mathcal{T}$-formula, find the heaviest non-$\mathcal{T}$-term t in it;
- find a set $\mathcal{A}$ of all (different) atomic formulae (with their polarities) from F in which t occurs;

[5] Note that we consider only universally closed formulae, so we can freely reorder quantifiers (in a formula which is in prenex normal form) and the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ can be used to eliminate any of the quantifiers. In the general case, $\mathtt{ElimOneVar}_{\mathcal{T},A}$ would be defined in the following way: if F is $\mathcal{T}^e$-formula in prenex normal form and if there is a variable v with corresponding quantifier in the *innermost block of the same quantifiers* and which does not appear in non-$\mathcal{T}$-terms of F, then generalise all its non-$\mathcal{T}$-terms, use the procedure $\mathtt{DpElimOneQuantifier}_{\mathcal{T},A}$ to eliminate the variable v and then substitute new variables by the original generalised terms.

– For each member f of the set $\mathcal{A}$, try to find a lemma[6] $\forall x_1 \forall x_2 \dots \forall x_n L_i$ such that:[7]
  (i) if t' is the heaviest non-$\mathcal{T}$-term[8] in L_i, and if it occurs in an atomic formula l, there exists a (most general) substitution ϕ_i such that dominant predicates symbols of f and l match, f and l have the same polarity (in F and L_i respectively), terms t and t' occur in the same argument positions (in f and l respectively) and $t = t'\phi_i$.
  (ii) all variables in L_i are substituted for some $\mathcal{T}$-terms in F (with corresponding types) by the substitution ϕ_i;
  (iii) the heaviest non-$\mathcal{T}$-term in $L_i\phi_i$ is $t'\phi_i$.

– The new current goal is the formula[9] $F \vee \neg L_1\phi_1 \vee \dots \vee \neg L_j\phi_j$; generalise all its non-$\mathcal{T}$-terms and then use the procedure `DpElimOneQuantifier`$_{\mathcal{T},A}$ to eliminate the variable v which corresponds to the term t; then substitute new variables for the original generalised terms; return the resulting formula as a current goal.

5 Extended Proof Method

We restrict our consideration only to universally quantified conjectures and lemmas. The extended proof method for the theory $\mathcal{T}$ (based on its decision procedure) we are proposing is defined in the following way:

(1) if possible (i.e., if there is a redundant quantifier), apply the procedure `ElimRedundantQuantifier` and go to step (1); otherwise, go to step (2).
(2) if possible (i.e., if a formula being proved is a $\mathcal{T}$-formula), use the procedure `Dp`$_{\mathcal{T},A}$:
  - if it returns `true`, then the original conjecture is valid,
  - if it returns `false` and
    ∗ the step (4) has not been applied, then the original conjecture is invalid,

[6] We use instances of substitutivity axioms as lemmas, but we can also use them in the following way: if f is the dominant function symbol of t, for each pair $f(t_1, t_2, \dots, t_m)$ and $f(u_1, u_2, \dots, u_m)$ of different terms occurring in a formula being proved we use $t_1 = u_2 \wedge t_2 = u_2 \wedge \dots \wedge t_m = u_m \rightarrow f(t_1, t_2, \dots, t_m) =_\tau f(u_1, u_2, \dots, u_m)$ as a lemma. This approach would have a wider scope, but would probably be less efficient.

[7] The motivation is that such lemmas contain information sufficient for proving the conjecture (although there are no guarantees of that kind).

[8] It is only sensible to search for lemmas which are $\mathcal{T}^e$ and are not $\mathcal{T}$-formulae; any lemma which is $\mathcal{T}$-formulae cannot contain any information that could not be derived by `Dp`$_{\mathcal{T},A}$; thus, we search just for lemma with non-$\mathcal{T}$-terms.

[9] Note that j does not have to be equal to the number of elements of $\mathcal{A}$: for some $\mathcal{T}^e$ conjectures to be proved no lemmas are required — for instance $\forall\alpha\ \forall k\ \ (max(\alpha) \leq k \vee max(\alpha) > k)$ can be proved without any lemma. Therefore, even if some of the lemmas with given properties are not found, it is still sensible to try to prove the conjecture by means of other lemmas and the theory $\mathcal{T}$ itself.

- $*$ the step (4) has been applied, then return to that point and try to apply $\texttt{ElimGeneralisedTerm}_{\mathcal{T},A}$ in another way; if it is not possible, the extended proof method stops failing to prove or disprove the conjecture and returns the current goal.

otherwise, go to step (3).

(3) if possible (i.e., if a formula F being proved is a $\mathcal{T}^e$-formula and if there is a variable v which does not appear in non-$\mathcal{T}$-terms of F), apply the procedure $\texttt{ElimOneVar}_{\mathcal{T},A}$ and go to step (1); otherwise, go to step (4).

(4) if possible (i.e., if a formula being proved is a $\mathcal{T}^e$-formula and is not a $\mathcal{T}$-formula), apply the procedure $\texttt{ElimGeneralisedTerm}_{\mathcal{T},A}$ and go to step (1) (if $\texttt{ElimGeneralisedTerm}_{\mathcal{T},A}$ can be applied in more than one way, then keep this position for possible backtracks).

6 Worked Example

Here we consider an example from [3]. We use (PNA) as a theory $\mathcal{T}$ and Cooper's algorithm [6] as an algorithm A.[10] Let $\{i, j, k, l, \ldots\}$ be a set of variables of type `pnat` and $\{\alpha, \beta, \gamma, \ldots, \}$ be a set of variables of type `list of pnats`. We consider the conjecture:

$$\forall l\ \forall \alpha\ \forall k \quad (l \leq min(\alpha)\ \wedge\ 0 < k \quad \rightarrow \quad l < max(\alpha) + k) \quad .$$

(1a) There are no redundant quantifiers in the conjecture, so go to step (2);

(2a) The conjecture is not a PNA-formula, so go to step (3);

(3a) There is a variable (k) which does not appear in not-PNA-terms ($min(\alpha)$ and $max(\alpha)$); generalize $min(\alpha)$ to min, $max(\alpha)$ to max and then use the procedure $\texttt{ElimOneVar}_{PNA,Cooper}$ to eliminate k from $\forall min \forall max \forall l \forall \alpha \forall k$ $(l \leq min\ \wedge\ 0 < k\ \rightarrow\ l < max + k)$. We get $\forall min \forall max \forall l \forall \alpha$ $(1 + min \leq l \vee l \leq max)$, and after substituting max by $max(\alpha)$ and min by $min(\alpha)$ we get $\forall l \forall \alpha$ $(1 + min(\alpha) \leq l \vee l \leq max(\alpha))$.

(3b) There is a variable (l) which does not appear in not-PNA-terms ($min(\alpha)$ and $max(\alpha)$); generalize $min(\alpha)$ to min, $max(\alpha)$ to max and then use the procedure $\texttt{ElimOneVar}_{PNA,Cooper}$ to eliminate l from $\forall min \forall max \forall \alpha \forall l$ $(1 + min \leq l \vee l \leq max)$. We get $\forall min \forall max \forall \alpha$ $(min \leq max)$, and after substituting max by $max(\alpha)$ and min by $min(\alpha)$ we get $\forall \alpha$ $(min(\alpha) \leq max(\alpha))$.

(4) The heaviest non-PNA-term in $\forall \alpha$ $(min(\alpha) \leq max(\alpha))$ is $min(\alpha)$. Suppose that there is a lemma $L \equiv \forall \xi$ $(min(\xi) \leq max(\xi))$ available. There is a substitution $\phi = \{\xi \mapsto \alpha\}$ such that $(min(\alpha)) = (min(\xi))\phi$. All preconditions of the procedure $\texttt{ElimGeneralisedTerm}_{PNA,Cooper}$ are fulfilled, so generalise all non-PNA-terms in the formula $\forall \alpha$ $min(\alpha) \leq max(\alpha) \vee \neg(min(\alpha) \leq max(\alpha))$ — generalise $min(\alpha)$ to min and $max(\alpha)$ to max and then use the

[10] In the system described in [3] Hodes' algorithm [9] is used, which is incomplete and is sound only for universally closed PIA formulae. Experimental results show [10] that Cooper's decision procedure for PNA, despite its $2^{2^{2^n}}$ worst-case complexity, is, for practical purposes, no worse than one due to Hodes', so we use Cooper's procedure here.

procedure $\texttt{DpElimOneQuantifier}_{PNA,Cooper}$ to eliminate the variable min. We get $\forall max\ \forall\alpha(0 \leq 0)$, and after substituting max by $max(\alpha)$ we get $\forall\alpha\ (0 \leq 0)$.

(1b) We can eliminate the quantifier $\forall\alpha$ as α does not occur in $0 \leq 0$ (using `ElimRedundantQuantifier`) and we get $0 \leq 0$.

(2b) $0 \leq 0$ is the PNA-formula, so we can use the procedure $\texttt{Dp}_{PNA,Cooper}$ which returns `true`. Thus, the conjecture is valid.

7 Properties of Extended Proof Method

Termination. Each of the simplification procedures returns either a formula with fewer variables than the original formula or the number of variables are the same, but the heaviest non-$\mathcal{T}$-term in the original formula is heavier than the heaviest non-$\mathcal{T}$-term in the resulting formula. Since there are finitely many variables in the formula being proved and since for each non-$\mathcal{T}$-term there are finitely many non-$\mathcal{T}$-terms over that set of variables for which it is heavier than, the simplification procedures can be applied only finitely many times (provided a finite set of lemmas). Therefore, the described method terminates.

Soundness. We assume that the available procedures $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\texttt{DpGround}_{\mathcal{T}}$ are complete and sound. Besides, if a formula with non-$\mathcal{T}$-terms generalised to variables is proved valid, then the initial formula is valid too. Therefore, the procedures $\texttt{Dp}_{\mathcal{T},A}$, `ElimRedundantQuantifier` and $\texttt{ElimOneVar}_{\mathcal{T},A}$ are sound.

Let us prove that the procedure $\texttt{ElimGeneralisedTerm}_{\mathcal{T},A}$ is also sound. Let $(\forall\overrightarrow{x})F$ be a formula being proved, let $(\forall\overrightarrow{u})L$ be a lemma and let ϕ be a substitution that meets the conditions of the procedure $\texttt{ElimGeneralisedTerm}_{\mathcal{T},A}$ (we consider just the simple case with one lemma used; the general case can be handled similarly). From $\mathcal{T}^e \vdash (\forall\overrightarrow{u})L$ it follows[11] $\mathcal{T}^e \vdash (\forall\overrightarrow{x})L\phi$. Let us suppose that we proved $\forall\overrightarrow{x}(F \vee \neg L\phi)$. Thus, $\mathcal{T}^e \vdash (\forall\overrightarrow{x})L\phi$ and $\mathcal{T}^e \vdash \forall\overrightarrow{x}(F \vee \neg L\phi)$ imply $\mathcal{T}^e \vdash \forall\overrightarrow{x}(L\phi \wedge (F \vee \neg L\phi))$ and, further, $\mathcal{T}^e \vdash \forall\overrightarrow{x}(L\phi \wedge F)$. Finally, it holds $\mathcal{T}^e \vdash \forall\overrightarrow{x}F$, i.e., $(\forall\overrightarrow{x})F$ is valid, which is what we wanted to prove.

(For the soundness of the presented extended proof method, it is sufficient that procedures $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\texttt{DpGround}_{\mathcal{T}}$ are sound. For instance, Hodes' algorithm can be used for the PNA procedure which would be sound (and incomplete) for universally closed formulae.)

(In)completeness. We do not make any claims about the completeness of our extended proof method: we do not see this as a severe weakness since we intend to exploit this approach in undecidable theories — we are trying to build a proof method that is successful outside the realm of a decision procedure. In a certain sense, our approach is complete since it is strictly an extension of some underlying decision procedure: those formula falling with the scope of that procedure will be decided correctly.

[11] We assume that $(\forall\overrightarrow{u})L$ is proved valid in $\mathcal{T}^e$.

Efficiency. In the proposed extended proof method there are some unnecessary repetitions due to the flexible combination of independent modules. However, these steps (generalisation of non-$\mathcal{T}$-terms, substitutions, some normalisations etc) can usually be executed in linear time on the size of a current goal and therefore do not significantly affect the efficiency of the system. Moreover, usually all steps of the method can be executed in linear time on the size of the formula being proved. Thus, the complexity of the presented method is dominated by the complexity of the procedures $\texttt{DpElimOneQuantifier}_{\mathcal{T},A}$ and $\texttt{DpGround}_{\mathcal{T}}$.

Besides, although the proposed method may seem a bit complicated, it is generally intended to be used for some simpler conjectures, so the described steps would be simple and fast. Additionally, we could restrict the use of the method just to some conjectures for which it is likely that the method will be successful (for this restriction we could use different heuristic scores or stochastic scores) and we could use some other techniques (e.g. induction) in other cases.

Flexibility. There is a trade-off between generality and efficiency in building an extended proof method. The proposed method is built by combining independent modules which could decrease efficiency, though hopefully not significantly. On the other hand however, the proposed method has a high degree of generality and flexibility: in a uniform way it can be used in different theorem provers, for different theories and for different decision procedures for these theories. It also does not require any specific data structures (for example, a database of polynomials). We claim that that potential losses in efficiency are dominated by these advantages.

8 Implementation and Results

We have made a preliminary implementation of the proposed extended proof method within the Clam system [5]. We have implemented procedures for PIA based on Hodes' algorithm[12] and for PIA and PNA based on Cooper's algorithm.[13] These implementations are also flexible and based on the idea of the exhaustive applications of a series or rewrite rule sets (see [4]). This, preliminary version of the extended proof method successfully proved a number of conjectures. Some results (obtained on examples from [3]) are given in Table 1.[14] We applied the extended proof method with three different algorithms for Presburger arithmetic (so, the variable l is of type `integer` in 1a and 1b and of type `pnat` in 1c etc.). The lemmas are formulae valid over natural numbers (but can be used in an adapted form with procedures working over the integers). It can be seen from the table that all three variants had problems with the third conjecture (the

[12] Hodes' procedure [9] is the decision procedure for PRA and the algorithm for PIA based on it is incomplete and is sound only for universally closed PIA formulae.

[13] Cooper's procedure [6] is the decision procedure for PIA, but can be adapted to the decision procedure for PNA.

[14] The extended proof method is implemented in the Clam proof-planning system under SWI Prolog. Tests are made on a PC486 16Mb, running under Linux. CPU time is given in seconds.

#	$\mathcal{T}$	A	lemmas	CPU time (s)
1	$\forall l \forall \alpha \forall k\ (l \leq min(\alpha) \wedge 0 < k \rightarrow l < max(\alpha) + k)$			
1a	PIA	Hodes'	$\forall \xi\ (min(\xi) \leq max(\xi))$	3.56
1b	PIA	Cooper's	$\forall \xi\ (min(\xi) \leq max(\xi))$	5.32
1c	PNA	Cooper's	$\forall \xi\ (min(\xi) \leq max(\xi))$	4.11
2	$\forall lp \forall lt \forall i \forall pat \forall c\ lp + lt \leq maxint \wedge i \leq lt \rightarrow i + delta(pat, lp, c) \leq maxint$			
2a	PIA	Hodes'	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	1.74
2b	PIA	Cooper's	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	2.52
2c	PNA	Cooper's	$\forall x \forall y \forall z\ delta(x, y, z) \leq y$	12.47
3	$\forall a \forall b \forall c ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$			
3a	PIA	Hodes'	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	32.35
3b	PIA	Cooper's	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	108.25
3c	PNA	Cooper's	$\forall x\ 0 < ms(x)$, $\forall i \forall j\ 0 < i \rightarrow j \leq i \cdot j$	?
4	$\forall a \forall b \forall c ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$			
4a	PIA	Hodes'	$\forall i \forall j\ j \leq ms(i) \cdot j$	9.77
4b	PIA	Cooper's	$\forall i \forall j\ j \leq ms(i) \cdot j$	35.49
4c	PNA	Cooper's	$\forall i \forall j\ j \leq ms(i) \cdot j$	3.41
5	$\forall k \forall l\ 0 < k \wedge 0 < l \wedge 2 \cdot k + 1 \leq 2 \cdot l \rightarrow 2 \cdot k + 2 \leq 2 \cdot l$			
5a	PIA	Hodes'		/
5b	PIA	Cooper's		11.27
5c	PNA	Cooper's		1.14

Table 1. Results of the preliminary implementation of the extended proof method

variant with Cooper's algorithm for PNA even ran out of the standard stack size in our system). In this example, lemmas have to be invoked six times and this leads to very complex intermediate formulae. However, if we change two lemmas from the third example with one lemma that is sufficient for a proof (the fourth example), the situation is changed and the system is more efficient. The explanation is the following: the facts $0 < ms(a)$ and $0 < ms(b)$ could not have been used while the terms $ms(a)$ and $ms(b)$ were not the heaviest non-$\mathcal{T}$-terms, so the intermediate formulae were very complex (they involved a large number of formulae $0 < ms(a)$ and $0 < ms(b)$). This was not a problem in the fourth example. This problem can be avoided in some cases: if the lemma we use is of the form $\forall \vec{x}(H_1 \wedge \cdots \wedge H_k \rightarrow C)$ then, after instantiation, we can try to prove the atomic formulae H_i $(i = 1, \ldots, k)$ separately and then, in the main proof we can use just the atomic formula C (the system [3] uses lemmas in this way). This approach can significantly increase the performances of the system in some cases (for instance, for the formula $\forall a \forall b \forall c\ ms(c) + ms(a)^2 + ms(b)^2 < ms(c) + ms(b)^2 + 2 \cdot ms(a)^2 \cdot ms(b) + ms(a)^4$, a speed-up similar to that obtained in the fourth example would be obtained).

The fourth example also illustrates the fact that Cooper's procedure for PNA can be much more efficient than the one for PIA or Hodes' procedure for PIA. Thus, it seems that it is sensible to have all these procedures available (while their ordering could be based on some heuristic scores and adjusted dynamically).

The fifth conjecture is invalid over reals and cannot be proved by Hodes' algorithm (and hence by the system presented in [3]), but can be proved by Cooper's algorithms for PIA and PNA. Thus, this example demonstrates the utility of having available different procedures for 'Presburger arithmetic'.

9 Future Work

Some of the ways in which the proposed method could be improved by making it more general are as follows:

- deal with predicates from $\Pi^e \setminus \Pi$; this could be done by using lemmas and generalising to variables of type `truth` (in a very similar manner we use for atomic formulae with non-$\mathcal{T}$-terms);
- use substitutivity axioms not only as lemmas;
- broaded scope to non-universally closed formulae;
- use definitions or lemmas about $\mathcal{T}^e$ functions and predicates expressed in terms of the theory $\mathcal{T}$ (e.g., we can always rewrite $double(x)$ to $2x$ by using a lemma $\forall x\ (double(x) =_{\texttt{pnat}} 2x)$ and we can always rewrite $x \neq y$ to $x < y \vee y < x$ by using a lemma $\forall x\, \forall y\ (x \neq y \leftrightarrow x < y \vee y < x)$);

Some of the ways in which the efficiency of the proposed method could be improved are the following:

- make the condition (i) in 4.4 more restricted — try to match whole atomic formulae f and l (not just the heaviest non-$\mathcal{T}$-terms); this approach would be more efficient for come conjectures, but would have smaller scope: e.g., within this approach $\forall\alpha\forall k\ min(\alpha) \leq max(\alpha) + k$ could not be proved using the lemma $\forall\xi(min(x) \leq max(x))$;
- preprocess lemmas or use lemmas having some fixed structure (e.g., those of the form $\forall\overrightarrow{x}(H_1 \wedge \cdots \wedge H_k \rightarrow C)$);

In future work we will investigate different ways for improving the presented extended proof method. We intend to make a more systematic study in order to find an optimum between generality and efficiency. In future work we also intend to explore possible combinations of this approach with other ones.

10 Related Work

In the last two decades a lot of effort has been invested into efficient and flexible integration of decision procedures (in particular those for arithmetic) into general-purpose theorem provers or domain specific systems. Our method is mostly related to a procedure for linear arithmetic integrated within Boyer and Moore's NQTHM [3]. This system is rather efficient, but involves a lot of special data structures and the description of the procedure is often given in terms of these special data-structures rather than in terms of their logical meaning. Besides, this system is adjusted for Hodes' algorithm and cannot be used on some other theory or some other algorithm. It is also incomplete for PNA. Our approach is a kind of rational reconstruction of Boyer and Moore's linear arithmetic procedure and is also its generalisation in some aspects. There are some losses in efficiency, but gains are in generality and flexibility.

Several systems are based on [16], including Stanford Pascal Verifier [14] and STeP [15]. In this approach, decision procedures for disjoint theories are combined by abstracting terms which fall outside a certain theory and by propagating deduced equalities from one theory to another. Several other systems are based on [19], including PVS [17] and EHDM [8]. In this approach, an efficient congruence closure procedure is used to handle combinations of ground equalities involving uninterpreted function symbols, linear arithmetic and arrays. Congruence closure is used in combination with decision procedures (solvers) for specific theories (for instance, with a solver for Presburger inequalities). There is an analysis of Shostak's algorithm and a partial analysis of an algorithm due to Nelson and Oppen in [7]. These approaches focus on combinations of (disjoint) theories in contrast to extensions of theories (which are not disjoint, but move outside some syntactically defined class). These systems are rather efficient over their intended domains, however, the method presented in this paper is more syntactical in its nature and more suited to a proof-planning environment (such as embodied in Clam [5]). There is also work on incorporating an arithmetic decision procedure into a rewrite based prover [11] and work in equational reasoning in resolution and paramodulation based provers [2, 1].

11 Conclusions

We have presented a method for the flexible integration certain decision procedures into theorem provers. It is partly based on [3], but is more general and more flexible. The method can be used in different theorem provers, for different theories and for different decision procedures for these theories. Specific knowledge is encapsulated in smaller submodules and decision procedures can be simply 'plugged-in' to the system. This framework is well-suited to the proof-planning paradigm. We have made a preliminary implementation within the Clam system and the results are most encouraging. In future work we propose to investigate different refinements of the method (outlined in section 9), including possible combination with other approaches.

References

1. L. Bachmair and H. Ganzinger. Strict basic superposition. In Kirchner and Kirchner [12], pages 160–174.
2. L. Bachmair, H. Ganzinger, and A. Voronkov. Elimination of equality via transformation with ordering constraints. In Kirchner and Kirchner [12], pages 175–190.
3. R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In J. E. Hayes, J. Richards, and D. Michie, editors, *Machine Intelligence 11*, pages 83–124, 1988.
4. A. Bundy. The use of proof plans for normalization. In R. S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991.

5. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.
6. D. C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*, pages 91–99. Edinburgh University Press, 1972.
7. D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. McRobbie and J. Slaney, editors, *13th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Vol. 1104, New Brunswick, NJ, USA, 1996. Springer-Verlag.
8. User guide for the EHDM specification language and verification system, version 6.1. Technical report, SRI Computer Science Laboratory, 1993.
9. L. Hodes. Solving problems by formula manipulation in logic and linear inequalities. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 553–559, Imperial College, London, England, 1971. The British Computer Society.
10. P. Janičić, I. Green, and A. Bundy. A comparison of decision procedures in Presburger arithmetic. In R. Tošić and Z. Budimac, editors, *Proceedings of the VIII Conference on Logic and Computer Science (LIRA '97)*, pages 91–101, Novi Sad, Yugoslavia, September 1–4 1997. University of Novi Sad.
11. D. Kapur and X. Nie. Reasoning about numbers in Tecton. In *Proceedings of 8th International Symposium on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, October 1994.
12. C. Kirchner and H. Kirchner, editors. *15th International Conference on Automated Deduction*, Lindau, Germany, July 1998.
13. G. Kreisel and J. L. Krivine. *Elements of mathematical logic: Model theory.* North Holland, Amsterdam, 1967.
14. D. C. Luckham, S. M. German, F. W. von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis. Stanford Pascal verifier user manual. CSD Report STAN-CS-79-731, Stanford University, 1979.
15. Z. Manna et al. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR;94-1518, Stanford University, 1994.
16. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.
17. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 1996 Conference on Computer-Aided Verification*, number 1102 in LNCS, pages 411–414, New Brunswick, New Jersey, U. S. A., 1996. Springer-Verlag.
18. M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa 1929*, pages 92–101, 395. Warsaw, 1930. Annotated English version also available [20].
19. R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.
20. R. Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR 84-639, Department of Computer Science, Cornell University, September 1984.