Division of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

A General Setting for Flexibly Combining and Augmenting Decision Procedures

by

Predrag Janicic, Alan Bundy

Informatics Research Report EDI-INF-RR-0095

Division of Informatics **January 2002**
http://www.informatics.ed.ac.uk/

A General Setting for Flexibly Combining and Augmenting Decision Procedures

Predrag Janicic, Alan Bundy

Informatics Research Report EDI-INF-RR-0095

DIVISION *of* INFORMATICS
Centre for Intelligent Systems and their Applications

January 2002

accepted - JAR 2002

Abstract :
The efficient combining and augmenting of decision procedures are often very important for a successful use of theorem provers. There are several schemes for combining and augmenting decision procedures; some of them support handling uninterpreted functions, use of available lemmas, and the like. In this paper we introduce a general setting for describing different schemes for both combining and augmenting decision procedures. This setting is based on the macro inference rules used in different approaches. Some of these rules are abstraction, entailment, congruence closure and lemma invoking. The general setting gives a simple description and the key ideas of one scheme and makes different schemes comparable. Also, it makes easier combining ideas from different schemes. In this paper we describe several schemes via introduced macro inference rules and report on our prototype implementation.

Keywords : theorem proving, decision procedures, combining decision procedures, augmenting decision procedures

A General Setting for Flexibly Combining and Augmenting Decision Procedures

Predrag Janičić (janicic@matf.bg.ac.yu)
Faculty of Mathematics, University of Belgrade,
Studentski trg 16, 11000 Belgrade, Yugoslavia

Alan Bundy* (a.bundy@ed.ac.uk)
Division of Informatics, University of Edinburgh,
80 South Bridge, Edinburgh EH1 1HN, Scotland

Abstract. The efficient combining and augmenting of decision procedures are often very important for a successful use of theorem provers. There are several schemes for combining and augmenting decision procedures; some of them support handling uninterpreted functions, use of available lemmas, and the like. In this paper we introduce a general setting for describing different schemes for both combining and augmenting decision procedures. This setting is based on the macro inference rules used in different approaches. Some of these rules are abstraction, entailment, congruence closure and lemma invoking. The general setting gives a simple description and the key ideas of one scheme and makes different schemes comparable. Also, it makes easier combining ideas from different schemes. In this paper we describe several schemes via introduced macro inference rules and report on our prototype implementation.

Keywords: Theorem proving, decision procedures, combining decision procedures, augmentation of decision procedures

1. Introduction

The role of decision procedures is often very important 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 one decision procedure. Some of these conjectures could, in an informal sense, fall "just outside" that scope. In these situations some additional knowledge is needed, lemmas have to be invoked, and/or decision procedures have to communicate with each other or with the heuristic component of a theorem prover.

In theorem proving, the role of decision procedures for fragments of arithmetic is significant because of its importance in software and hardware verification. The whole of arithmetic is undecidable, but it has decidable fragments such as Presburger arithmetic and strictly multiplication arithmetic [41, 44, 36]. As said, however, a decision procedure

* Supported in part by EPSRC grant GR/M45030.

for Presburger arithmetic itself has limited power: for instance, the formula

$$\forall x \forall y\ x \leq y \wedge y \leq x + car(cons(0,x)) \wedge p(h(x) - h(y)) \rightarrow p(0)$$

is not a Presburger arithmetic formula. Besides, the formula

$$\forall x \forall y\ x \leq y \wedge y \leq x + c \wedge p(h(x) - h(y)) \rightarrow p(0)$$

obtained by generalizing $car(cons(0,x))$ to c is a Presburger arithmetic formula (extended by the theory of equality with uninterpreted function and predicate symbols, i.e., by the pure theory of equality), but is not a theorem. However, if there are decision procedures for the theory of lists and for the pure theory of equality available, it is possible to combine them with a decision procedure for Presburger arithmetic and to prove the above conjecture. This example illustrates situations when several decision procedures have to *cooperate* and to be *combined* into one decision procedure. As another example, consider the formula

$$\forall l\ \forall \alpha\ \forall k\ \ (l \leq minl(\alpha)\ \wedge\ 0 < k\ \ \rightarrow\ \ l < maxl(\alpha) + k)\ \ .$$

It is not a Presburger arithmetic formula, and, besides, the formula

$$\forall maxl\ \forall minl\ \forall l\ \forall k\ \ (l \leq minl \wedge\ 0 < k\ \rightarrow\ l < maxl + k)$$

obtained by generalizing $minl(\alpha)$ to $minl$ and $maxl(\alpha)$ to $maxl$ is a Presburger arithmetic formula but is not a theorem. However, if the lemma $\forall \xi\ (minl(\xi) \leq maxl(\xi))$ is available, it can be used (with the right instantiation) and lead to

$$\forall l \forall \alpha \forall k\ (minl(\alpha) \leq maxl(\alpha) \rightarrow (l \leq minl(\alpha) \wedge\ 0 < k\ \rightarrow\ l < maxl(\alpha) + k))\ .$$

After the generalization, we get the formula

$$\forall maxl \forall minl \forall l \forall k\ (minl \leq maxl \rightarrow (l \leq minl \wedge\ 0 < k\ \rightarrow\ l < maxl + k))\ ,$$

which can be proved by a decision procedure for Presburger arithmetic. This example illustrates how the realm of the decision procedure for Presburger arithmetic can be extended by the use of available lemmas. In situations like this one, a decision procedure has to communicate with other components of a prover, such as a lemma-invoking mechanism, a rewriting mechanism, or simplification techniques. Then we say we deal with *augmenting* a decision procedure (or we say we deal with *integrating* or *incorporating* a decision procedure into a theorem prover).

Several influential approaches are used in handling the problem of efficiently combining and augmenting decision procedures. The research concerning these issues has gone mostly along different two lines: one concerning combining decision procedures and one concerning augmenting decision procedures. Combination schemes typically rely on some local, specific data structures [38, 43, 19, 7, 9, 42, 8] and are focused on combining decision procedures, but some of them also use additional techniques (such as the use of additional rules and lemmas). Augmentation schemes use different functionalities of heuristic theorem provers such as rewriting techniques, lemma-invoking mechanisms, or a variety of simplifications [11, 27, 29, 1, 25, 3]. Combination schemes are typically aimed at decidable combinations of theories, while augmentation schemes are primarily intended for use in (often undecidable) extensions of decidable theories. The research on combination of decision procedures is rich in theoretical and practical results, while the research on the augmentation of decision procedures is still less well developed theoretically.

Nelson and Oppen's scheme for combination of theories [37] is used in several systems, including the Stanford Pascal Verifier [32], ESC [21], and EVES [18]. In this approach, decision procedures for disjoint theories are combined by abstracting terms that fall outside a certain theory and by propagating deduced equalities from one theory to another. Shostak's scheme for combination of theories [43] is used in several other systems, including EHDM [22], PVS [40], STeP [33, 9], and SVC [7]. In this approach, *solvers* for specific theories (for instance, solvers for equational real arithmetic or theory of lists) are tightly combined by an efficient underlying congruence closure algorithm for ground equalities.

One of the most influential methods in the incorporation of decision procedures into theorem provers is a procedure for linear arithmetic integrated within Boyer and Moore's NQTHM [11]. This system involves a lot of special data structures, and the description of the procedure is given in terms of these special data structures rather than in terms of their logical meaning. There is also an important work on incorporating an arithmetic decision procedure into a rewrite-based prover TECTON [27].

In this paper we intend to capture the key ideas shared by these systems, and we formulate a general setting (GS) for both the combining and the augmenting decision procedures. As all of the mentioned systems, the setting we present in this paper addresses quantifier-free (that is, universal) background theories and only quantifier-free (that is, universally quantified) conjectures.

We won't go into much detailed discussion about the efficiency of some modules of a system, but we will focus on modularity and generality enough to cover different variants of the mentioned systems. The presented general setting also provides the possibility for easily combining ideas from different approaches (e.g., for combining Nelson and Oppen's scheme with a lemma-invoking mechanism) and hence for easy definition of new combination/augmentation schemes.

Overview of the Paper. In §2 we give some basic background and notation information. In §3 we describe several schemes from the literature for combining and augmenting decision procedures. In §4 we introduce macro inference rules that constitute a general setting (GS) for combination and augmentation of decision procedures; they are based on schemes known from the literature. In §5 we give a case study showing how six known schemes can be described via the introduced macro inference rules. In §6 we describe an extension of the general setting by an ordering on the macro inference rules. In §7 we report on a prototype implementation of the described general setting and on some results obtained by different schemes implemented within the setting. In §8 we discuss related work. In §9 we discuss future work, and in §10 we draw some conclusions.

2. Background and Notation

A *signature* (or a *language*) Λ is a pair (Σ, Π), where Σ and Π are disjoint sets of *function* and *predicate* symbols, each with an associated *arity* (or *degree*), an integer greater than or equal to 0. If a function symbol has an arity 0, then we call it a *constant*. If a predicate symbol has an arity 0, then we call that it a *logical constant* or a *truth constant*. There are two truth constants: $\top$ and $\bot$. Given a set V of variables (disjoint with Σ and with Π), we define the notion of a Λ-*term* in the usual way. We also adopt standard definitions for *free variable*, Λ-*atomic formula*, Λ-*literal*, and first-order Λ-*formula*. A Λ-formula with no free variables we call a *closed* Λ-*formula* or a Λ-*sentence*. A formula F is *ground* if it has no variables. A formula F is in *prenex normal form* if it is of the form $Q_1x_1\ Q_2x_2\ \ldots Q_kx_k\ F'$, where $Q_i \in \{\forall, \exists\}$, $x_i \in V$ and there are no quantifiers in F'. If $\{x_1, x_2, \ldots, x_k\}$ is a set of free variables in F, we say that $\forall x_1\ \forall x_2\ \ldots \forall x_k\ F$ is the *universal closure* of F (and that $\forall x_1\ \forall x_2\ \ldots \forall x_k\ F$ is a *universally closed* formula). We abbreviate the universal closure of a formula F by $\forall * F$. By analogy we define the *existential closure* of formula F, and we abbreviate it by $\exists * F$.

Given a signature Λ, a Λ-*structure* $\mathcal{A}$ is a pair (A, I^{Λ}), where A, the *universe* of $\mathcal{A}$, is a set, while I^{Λ} is a function mapping each constant $c \in \Sigma$ to an element, or *individual*, $c^{\mathcal{A}}$ of $\mathcal{A}$, each n-ary function symbol $f \in \Sigma$ $(n > 0)$ to a total function $f^{\mathcal{A}}$ from A^n to A, and each n-ary predicate symbol $p \in \Pi$ to a relation $p^{\mathcal{A}}$ over A^n. If $\mathcal{A}$ is a Λ-structure, and X is a subset of V, a *valuation* of X is a mapping α of X into A. The pair $(\mathcal{A}, \alpha)$ defines a formula *interpretation*, that is, a mapping into $\{true, false\}$ of the set of all Λ-formulae F such that the set of free variables in F is a subset of X ($\top$ is always mapped to $true$ and $\perp$ is always mapped to $false$). The interpretation $(\mathcal{A}, \alpha)$ *satisfies* a Λ-formula F (or also, α *satisfies* F in $\mathcal{A}$) if $(\mathcal{A}, \alpha)$ maps F to $true$, and we denote this by $(\mathcal{A}, \alpha) \models F$. We write $\mathcal{A} \models F$ and say that $\mathcal{A}$ *models* F if every valuation of free variables in F into A satisfies F.

A *first-order theory* (or an *axiom system*) $\mathcal{T}$ is a set of first-order sentences. A Λ-theory is a theory whose all sentences are Λ-sentences. A first-order theory with equality includes the predicate symbol $=$ and reflexivity, symmetry, transitivity, and substitutivity axioms for equality. In the rest of this paper we will assume that we work only with first-order theories with equality. If $\mathcal{T}$ is a theory over a signature Λ, by $\mathcal{T}$-term, $\mathcal{T}$-formula, and so forth, we mean Λ-term, Λ-formula, and the like. A Λ-structure $\mathcal{A}$ is a *model* of a Λ-theory $\mathcal{T}$ if $\mathcal{A}$ models every sentence in $\mathcal{T}$. We denote by $Mod(\mathcal{T})$ the set of all models of $\mathcal{T}$. For a theory $\mathcal{T}$ we say that is *consistent* if it has at least one model. For Λ-theory $\mathcal{T}$ and a Λ-formula F, we say that F is a *logical consequence* of $\mathcal{T}$, and we write $\mathcal{T} \models F$, if $\mathcal{A} \models F$ (or, equivalently, if $\mathcal{A} \models \forall * F$) for every member $\mathcal{A}$ of $Mod(\mathcal{T})$. For Λ-theory $\mathcal{T}$ and a Λ-formula F, we say that F is *satisfiable* in $\mathcal{T}$ if there is a valuation of free variables of F that satisfies F in some member $\mathcal{A}$ of $Mod(\mathcal{T})$ (or, equivalently, $\mathcal{A} \models \exists * F$). If a formula F can be derived from $\mathcal{T}$ by using the usual classical first-order logic inference system [35], 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 $\mathcal{T}$. For a given signature Λ, a Λ-theory $\mathcal{T}$ and a Λ-sentence F, $\mathcal{T} \models F$ if and only if $\mathcal{T} \vdash F$.

If for two signatures $\Lambda^e = (\Sigma^e, \Pi^e)$ and $\Lambda = (\Sigma, \Pi)$ it holds that $\Sigma \subseteq \Sigma^e$, $\Pi \subseteq \Pi^e$, we say that the signature Λ^e is an *extension* of the signature Λ. If $\mathcal{T}$ is a theory over Λ, $\mathcal{T}^e$ is a theory over Λ^e, and if it holds that $\mathcal{T} \subseteq \mathcal{T}^e$, we say that the theory $\mathcal{T}^e$ is an *extension* of the theory $\mathcal{T}$ (and that $\mathcal{T}$ is a *subtheory* of $\mathcal{T}^e$). We say that the theory $\mathcal{T}^e$ is a *conservative* extension of the theory $\mathcal{T}$ if the set of $\mathcal{T}^e$-theorems that are Λ-formulae is equal to the set of $\mathcal{T}$-theorems. Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ over signatures Λ_1 and Λ_2, we call a theory $\mathcal{T}_1 \cup \mathcal{T}_2$ (over the signature $\Lambda_1 \cup \Lambda_2$) a *combination* (or a *union*) of $\mathcal{T}_1$ and $\mathcal{T}_2$.

A theory is *quantifier-free* (or *universal*) iff all of its axioms are quantifier-free (that is, implicitly universally quantified) formulae. In the rest of this paper, we will consider only quantifier-free theories, unless stated otherwise.

A theory $\mathcal{T}$ is *decidable* if there is an algorithm (which we call a *decision procedure*) such that for an input $\mathcal{T}$-sentence F, it returns *true* if and only if $\mathcal{T} \vdash F$ (and returns *false* otherwise).

In the rest of the paper, special attention will be given to Presburger arithmetic (according to its significance in verification problems). In Presburger natural arithmetic (PNA), function symbols are 0 (with arity 0), s (with arity 1) and $+$ (with arity 2); predicate symbols are $<, >, \leq, \geq$ (all with arity 2). We write 1 instead of $s(0)$, and so on. Multiplication 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 Presburger arithmetic over integers — *Presburger integer arithmetic* (PIA) and Presburger arithmetic over rationals — and *Presburger rational arithmetic* (PRA).[1] It was Presburger who first showed that PIA is decidable [41]. The decidability of PNA can be proved in an analogous way. PRA is also decidable [30].

We denote by $\mathcal{E}$ the pure theory of equality, that is, the theory with "uninterpreted" function and predicate symbols (that is, functions and predicates with no information about them), and only with equality axioms.

A *rewrite rule* is an oriented equality of the form $l \longrightarrow r$. The rule $l \longrightarrow r$ rewrites a term t to another term s if there is a subterm t' of t and a substitution ϕ such that $l\phi$ is equal to t' and s is t with the subterm t' replaced by $r\phi$. A *rewrite system* $\mathcal{R}$ is a finite set of rewrite rules. A rewrite system $\mathcal{R}$ is *terminating* if it does not enable infinite rewrite sequences. A rewrite system is terminating if there is a reduction ordering $\prec$ such that for each rule $l \longrightarrow r$ it holds $r \prec l$. When a precedence relation on function and predicate symbols is given, Dershowitz's recursive path ordering [20] extends that relation to reduction ordering $\prec$ on terms. A *conditional rewrite rule* is a rule of the form $l \longrightarrow r$ **if** $p_1 \wedge p_2 \wedge \ldots \wedge p_k$, where the conditions $p_1, p_2, \ldots, p_k$ are literals. The rule $l \longrightarrow r$ **if** $p_1 \wedge p_2 \wedge \ldots \wedge p_k$ rewrites a term t to another term s if there is a subterm t' of t and a substitution ϕ such that $l\phi$ is equal to t', s is t with the subterm t' replaced by $r\phi$, and each of the conditions $p_i\phi$ reduces to $\top$ (also by using rewrite rules). A system of conditional rewrite rules is terminating if there is a reduction ordering $\prec$ such that for each rule $l \longrightarrow r$ **if** $p_1 \wedge p_2 \wedge \ldots \wedge p_k$, it holds $r \prec l$ and $p_i \prec l$ $(1 \leq i \leq k)$.

[1] For identical and related theories a number of different terms are used. For instance, Boyer and Moore [11] describe a universally quantified fragment of Presburger rational arithmetic as *linear arithmetic* (although in fact they work over the integers).

For a given reduction ordering $\prec$, a term t is a *maximal* in a set if there is no term t' in that set such that $t \prec t'$. A term t is called an *alien term* w.r.t. $\mathcal{T}$ if it is not a term of this theory. A literal l is a *maximal* in a set of literals if there is no literal l' in that set such that $l \prec l'$. A literal l is called an *alien literal* w.r.t. $\mathcal{T}$ if it is not a literal of this theory.

3. Schemes for Combination and Augmentation of Decision Procedures

In the late 1970s and early 1980s, interest in decision procedures for simple theories and for their use in theorem proving was revived. This interest was supported by influential schemes for combinations of decision procedures reported by Nelson and Oppen and by Shostak. One of the most influential approaches to incorporating decision procedures into heuristic theorem provers was reported in the late 1980s by Boyer and Moore and successfully applied in their NQTHM. In this section we briefly describe these and three other schemes that will be discussed in the rest of the paper.[2] The macro inference rules presented in Section 4 are motivated by these approaches.

Nelson and Oppen's Algorithm for Combination of Theories: This approach [37] gives a decision procedure for a combination of (decidable) quantifier-free stably infinite theories[3] that do not share logical symbols other than equality (given the decision procedures for these theories). Alien terms in literals are abstracted, and literals (which make the conjunction being refuted) are then divided into groups according to which theories they belong to; after that, each decision procedure tries to deduce some new equality and to propagate it to other theories. In this approach, deduction of a new equality E in some theory can be based on the decision procedure for that theory (i.e., on checking the unsatisfiability of the conjunction of literals augmented by $\neg E$), but more efficient algorithms for specific theories can also be used for that purpose. An analysis of Nelson and Oppen's procedure and its theoretical background can be found in [46].

Shostak's algorithm for combination of theories: Shostak gives a procedure for a combination of *algebraically solvable* quantifier-free

[2] This overview is by no means intended to give a detailed account of research on combining and augmenting decision procedures, but to briefly present some of the most influential schemes and schemes that served as a basis for the setting described in this paper.

[3] A consistent, quantifier-free theory $\mathcal{T}$ with signature $\mathcal{L}$ is called stably infinite if and only if any quantifier-free $\mathcal{L}$-formula is satisfiable in $\mathcal{T}$ if and only if it is satisfiable in an infinite model of $\mathcal{T}$.

equational σ-theories[4] [43]. It is based on *solvers* for specific theories that are tightly combined by an efficient underlying congruence closure algorithm for ground equalities. This algorithm deals merely with equational theories and, for instance, cannot deal with linear arithmetic inequalities. (In Shostak's original implementation linear arithmetic inequalities were treated by an external decision procedure, in the Nelson and Oppen style.) An analysis of Shostak's algorithm can be found in [19]. Recently, Rueß and Shankar [42] showed that Shostak's procedure and all its published variants [19, 7, 9] are incomplete and can even be nonterminating; they also present one new variant of Shostak's procedure and prove its termination, soundness, and completeness. There is also another recent variant of Shostak's algorithm reported by Barrett, Dill, and Stump [8].

Like the Nelson and Oppen's algorithm, Shostak's algorithm in its basic form does not address the problem of using additional rewrite rules, lemmas, and the like.

Boyer and Moore's linear arithmetic procedure: The algorithm of Boyer and Moore [11] extends the realm of one decision procedure rather than combining decision procedures for different theories. In their approach, a decision procedure for Presburger arithmetic (based on Hodes' algorithm [23]) is combined with a heuristic *augmentation* that provides information about alien functions (i.e., provides lemmas) and with a technique for using the remaining literals in the clause being proved as the context to simplify one literal. In this approach, an underlying decision procedure for Presburger arithmetic can't be changed for another one, which makes the system highly inflexible. Despite the successful use of this algorithm in NQTHM and its general influence (not only on the integration of decision procedures into heuristic provers), it hasn't gained wider popularity, and very few systems use it. One of the reasons is probably the complicated original description, which is often given in terms of special data structures rather than in terms of their logical meaning and is often given intermixed with different optimizations. All that also makes any theoretical analysis of Boyer and Moore's algorithm a very difficult task.

Reasoning About Numbers in TECTON*:* Kapur and Nie's algorithm for reasoning about numbers [27] is based on Boyer and Moore's approach but improves it in several ways. It is designed so that the underlying reasoning about Presburger formulae can be handled by different procedures. It makes the system much more flexible than Boyer and Moore's procedure. The original description is based on a form of Fourier's algorithm for Presburger arithmetic; as a decision

[4] Definitions of algebraically solvable and σ-theories are given in Example 2.

procedure it is essentially the same as one due to Hodes [23] used by Boyer and Moore, but in its extended form [31], it also provides a mechanism for deducing implicit Presburger equalities (which are efficiently used for further rewriting of the remaining parts of a formula being proved). This scheme uses literals in the formula being proved as the context to simplify one literal, similar to Boyer and Moore's procedure, but described more precisely via congruence closure and contextual rewriting [47]. This approach uses rewriting techniques and is used in the rewrite-based prover TECTON.

Constraint Contextual Rewriting: Armando and Ranise's constraint contextual rewriting [1, 3] is a formal system motivated by the ideas from Boyer and Moore's system. Constraint contextual rewriting is an extended form of contextual rewriting [47] that incorporates the functionalities provided by a decision procedure. It provides a flexible framework that can be instantiated by a specific decision procedure X for some theory, and it can formally cover approaches used by Boyer and Moore and by Kapur and Nie (or, at least, their essential parts). The soundness and termination of the constraint contextual rewriting are proved formally for the abstract scheme when certain requirements on the rewriting mechanism and the decision procedure are satisfied [3].

Extended Proof Method: The extended proof method (EPM) [25] is in spirit similar to constraint contextual rewriting — it provides a flexible framework for extending the realm of one decision procedure A for a theory $\mathcal{T}$ (say, for Presburger arithmetic) by the use of available lemmas. The framework can be used for different theories and for different decision procedures. A new procedure can be "plugged in" to the system only if it provides a limited set of functionalities such as checking satisfiability and elimination of quantifiers. The soundness and termination of the extended proof method are proved for the abstract framework, given a decision procedure that provides needed functionalities.

As with the previous three approaches, the extended proof method does not address the problem of combining decision procedures, but only the problem of extending the realm of one decision procedure by available lemmas.

4. Macro Inference Rules

There are several key steps in the different systems for combining and augmenting decision procedures, including abstraction, entailment, and congruence closure. Instead of focusing on specific instances of these steps (which is the case in most of the system descriptions), we will give a general setting for combining different instances of these steps. We will describe them in a form of macro inference rules.

The macro inference rules are applied with respect to a background theory $\mathcal{T}$ and a corresponding signature Λ. The background theory is a combination of theories $\mathcal{T}_1$, $\mathcal{T}_2$, $\ldots$, $\mathcal{T}_k$ (and possibly extended by an extension $\mathcal{L}$). In each combination/augmentation scheme based on the proposed macro inference rules, the background theory is fixed for all macro inference rules used. Some schemes can be parameterized by choosing a background theory. We deal only with quantifier-free background theories. Usually, it is sensible to consider decidable theories $\mathcal{T}_i$ ($1 \leq i \leq k$). If $\mathcal{T}$ is decidable, it is an interesting problem to build a decision procedure for it, on the basis of decision procedures for $\mathcal{T}_1$, $\mathcal{T}_2$, $\ldots$, $\mathcal{T}_k$. On the other hand, an interesting case is also when $\mathcal{T}$ is undecidable extension of a decidable theory.

We use proof by refutation, so if F is a quantifier-free formula to be proved, we need to show that $\neg F$ is unsatisfiable in $\mathcal{T}$; that is, we need to show that $\mathcal{T} \cup \neg F$ is inconsistent. More precisely, in order to prove a quantifier-free formula F in a theory $\mathcal{T}$, that is, to show $\mathcal{T} \models \forall^* F$, we need to show that $\mathcal{T}, \exists^*(\neg F) \models \perp$. For now on, we will assume that for one conjecture and for one combination/augmentation procedure, the background theory $\mathcal{T}$ is fixed, and by "$\neg F$ is unsatisfiable" we will mean "$\neg F$ is unsatisfiable in $\mathcal{T}$", and so forth.

We denote macro inference rules by $\Rightarrow_X$, where X is the name of the specific rule. If a rule X is parameterized by parameters P, then we denote it by $X(P)$. The soundness of each macro inference rule has to be ensured: for each rule $\Rightarrow_X$, if it holds $f_1 \Rightarrow_X f_2$, then it has to be ensured that if f_2 is unsatisfiable (in the background theory $\mathcal{T}$), then f_1 is unsatisfiable (in $\mathcal{T}$), too (in other words, each rule $\Rightarrow_X$ has to preserve satisfiability). If the inference rule X is both satisfiability and unsatisfiability preserving, we denote it by $\Leftrightarrow_X$. We abbreviate by $f_1 \Rightarrow^* f_n$ the sequence $f_1 \Rightarrow_{X_{i_1}} f_2 \Rightarrow_{X_{i_2}} f_3 \ldots f_{n-1} \Rightarrow_{X_{i_{n-1}}} f_n$, and by $f_1 \Leftrightarrow^* f_n$ the sequence $f_1 \Leftrightarrow_{X_{i_1}} f_2 \Leftrightarrow_{X_{i_2}} f_3 \ldots f_{n-1} \Leftrightarrow_{X_{i_{n-1}}} f_n$.

If $\neg F \Rightarrow^* \perp$, then $\neg F$ is unsatisfiable, so the original formula F is valid. If $\neg F \Leftrightarrow^* \top$ and the background theory $\mathcal{T}$ is consistent, then F is invalid. In a heuristic theorem prover, negative results can also be important and useful (for example, in controlling generalization, and other non satisfiability-preserving heuristics). Additionally, after each macro inference step is applied, if $f \Leftrightarrow^* f'$, a formula f' can be passed to any other component of a prover, and some other technique (e.g., induction) can be tried.

We assume that there is an ordering on Λ function and predicate symbols available that induces a reduction ordering $\prec$ on terms. We also assume that there may be some additional rewrite rules, definitional equalities, and lemmas available.

The rest of this section describes in details the various macro inference rules. Some of them are defined with respect to theories $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_k$ (and possibly an extension $\mathcal{L}$) that are combined into the background theory.

4.1. Disjunctive Normal Form

If f is the negation of a formula F to be proved, we can transform it into disjunctive normal form and then try to refute each of its disjuncts. We denote this transformation[5] in the following way:

$$f \Rightarrow_{dnf} (f_1 \vee f_2 \vee \ldots \vee f_n)$$

Since transformation into disjunctive normal form is (un)satisfiability preserving, we also denote it by

$$f \Leftrightarrow_{dnf} (f_1 \vee f_2 \vee \ldots \vee f_n)\ .$$

4.2. Proving Disjuncts (case split)

If a formula f is of the form $(f_1 \vee f_2 \vee \ldots \vee f_n)$ (where each f_i is a conjunction of literals), in order to refute f, we have to refute each f_i. Hence,

$$f \Leftrightarrow_{split} \bot \;\text{ if }\; f_i \Rightarrow^* \bot \text{ for all } i,\ 1 \leq i \leq n\ .$$

Additionally,

$$f \Leftrightarrow_{split} \top \;\text{ if }\; f_i \Leftrightarrow^* \top \text{ for some } i,\ 1 \leq i \leq n\ .$$

We will further deal mostly with conjunctions of literals (i.e., conjunctions of atomic formulae and negations of atomic formulae). We won't distinguish between a conjunction $l_1 \wedge l_2 \wedge \ldots \wedge l_n$ and a multiset $\{l_1, l_2, \ldots, l_n\}$.

[5] Transformation into disjunctive normal form can be performed via a terminating set of rewrite rules. However, there is no canonical rewriting system for doing this.

4.3. Simplification

Simplification is based on the following simple rules:

$$\begin{array}{rcl}
\{\,\} & \longrightarrow & \top \\
f \cup \{l\} \cup \{l\} & \longrightarrow & f \cup \{l\} \\
f \cup \{l\} \cup \{\neg l\} & \longrightarrow & \bot \\
f \cup \{\bot\} & \longrightarrow & \bot \\
f \cup \{\top\} & \longrightarrow & f \\
f \cup \{\neg(a = a)\} & \longrightarrow & \bot \\
f \cup \{a = a\} & \longrightarrow & f \\
f \cup \{x = y\} & \longrightarrow & f
\end{array}$$

where x is a variable that does not occur in y and f, or y is a variable that does not occur in x and f.

The inference rule that exhaustively applies the above rules (on a formula f and yields a formula f') we denote by

$$f \Leftrightarrow_{simpl} f' \, .$$

In addition, there are simplifications specific to each theory. For instance, a literal $x \leq (y+z) - (y-x)$ can be simplified to $0 \leq z$ by a simplifier for PRA. Also, a simplifier for a theory $\mathcal{T}_i$ should detect valid and unsatisfiable ground $\mathcal{T}_i$ literals and rewrite them to $\top$ and $\bot$, respectively. Simplification for a specific theory $\mathcal{T}_i$ is performed only on $\mathcal{T}_i$ literals. Dealing with a combination of theories $\mathcal{T}_i$ $(i = 1, 2, \ldots, k)$, we iteratively use simplifications for theories $\mathcal{T}_i$ $(i = 1, 2, \ldots, k)$, and we denote this simplification (the extended form of $\Leftrightarrow_{simpl}$) by

$$f \Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \, .$$

We assume that $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ does not introduce new variables and that can replace a $\mathcal{T}_i$-literal only by a $\mathcal{T}_i$-literal. We also assume that $f \Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f'$ and $f' \Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f''$ implies that f' and f'' are identical formulae. Moreover, in order to ensure (un)satisfiability preservation in the background theory $\mathcal{T}$, the rule $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ must fulfill some conditions. If this rule replaces a $\mathcal{T}_i$-literal l by a $\mathcal{T}_i$-literal l', then the following must hold: for every model M of $\mathcal{T}_i$, every valuation of the variables of l that satisfies l in M can be extended to a valuation that satisfy l' in M, and vice versa. With this condition met, it is not difficult to show that $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ preserves both satisfiability and unsatisfiability in $\mathcal{T}$.

4.4. Abstraction

Depending on the dominating functional and predicate symbols, we can conditionally consider that an atomic formula belongs to a theory $\mathcal{T}_i$ and abstract its alien subterms (with respect to $\mathcal{T}_i$ by new, *abstraction variables* in order to obtain an atomic formula of a theory $\mathcal{T}_i$. For instance, $x \leq y + g(x+y)$ can be abstracted into a Presburger arithmetic atomic formula $x \leq y + t$, where $t = g(x+y)$.

We can perform abstraction for several theories $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ in one step (in such a way that each abstracted literal belongs to one of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$). We can replace all original literals by their abstracted versions (which is not (un)satisfiability preserving):

$$f \Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f'$$

We can also add the set C of equalities defining newly introduced variables ($\{t = g(x+y)\}$ in the above example), which we denote by

$$f \Rightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \cup C$$

or, alternatively (as it preserves unsatisfiability), by

$$f \Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \cup C \ .$$

Abstraction can be propagated if a literal defining an abstracted variable does not belong to one of the theories $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$. For instance, after propagated abstraction (with respect to PNA and the pure theory of equality) $x \leq y + g(x+y)$ becomes $x \leq y + t_1 \wedge t_1 = g(t_2) \wedge t_2 = x + y$. This form of abstraction (propagated abstraction) we denote by *absp*, and it always adds equalities C defining newly introduced variables.[6] We denote the corresponding inference by

$$f \Rightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \cup C \ .$$

or, alternatively (as it preserves unsatisfiability), by

$$f \Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \cup C \ .$$

4.5. Replacement

If a current formula f is a conjunction of literals and if there is a literal $x = t$, where x is a variable and t is a term that does not include occurrences of x, then we can replace all occurrences of x in all formulae by t and delete the literal $x = t$. We can perform this operation for all

[6] Some authors call this rule (or equivalent rules) *purification*.

variables x fulfilling the given condition (one after another[7]) obtaining a formula f', and we denote this inference by

$$f \Rightarrow_{repl} f'$$

or, alternatively (as it preserves unsatisfiability), by

$$f \Leftrightarrow_{repl} f' \ .$$

Assuming that f does not include literals fulfilling the given conditions, we note that

$$f \Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \Leftrightarrow_{repl} f \ .$$

This illustrates the fact that one has to be cautious with the use of the replacement rule, since it can annul the effect of the abstraction rule.

Another version of the replacement is restricted only to replacement of variables by variables (i.e., in the case where there are equalities of the form $x = y$). We denote it by

$$f \Rightarrow_{repl^=} f'$$

or, alternatively (as it preserves unsatisfiability), by

$$f \Leftrightarrow_{repl^=} f' \ .$$

4.6. Unsatisfiability

Let us suppose that there is a procedure that can detect if a conjunction of some literals $\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ ($i_k < n$) from a formula f is unsatisfiable in a theory $\mathcal{T}_i$, where f is a conjunction of literals $l_1, l_2, \ldots, l_n$. If the conjunction $\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ ($i_k < n$) is unsatisfiable, then f is unsatisfiable, too. We denote this inference in the following way:

$$f \Leftrightarrow_{unsat(\mathcal{T}_i)} \bot \ .$$

It is obvious that this rule preserves both satisfiability and unsatisfiability in the background theory $\mathcal{T}$ (since $\mathcal{T}_i$ is a subtheory of $\mathcal{T}$).

4.7. Entailment

Let f be a disjoint union of two sets of literals: A and B. Let us suppose that literals $B = \{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ entail (in a theory $\mathcal{T}_i$) some conjunction or disjunction of literals C (which do not occur in $A \cup B$).

[7] Generally, the effect of the replacement might depend on the order of variables. For simplicity, we won't discuss this issue.

We assume that soundness is guaranteed by the specification (which is denoted by P for $\mathcal{T}_i$) of this entailment. If we keep the literals from B, we denote it by

$$A \cup B \Rightarrow_{entail^+(P,\mathcal{T}_i)} A \cup B \cup C$$

or, alternatively (as it preserves unsatisfiability), by

$$A \cup B \Leftrightarrow_{entail^+(P,\mathcal{T}_i)} A \cup B \cup C \ .$$

If we don't keep the literals from B, we denote it by

$$A \cup B \Rightarrow_{entail(P,\mathcal{T}_i)} A \cup C \ ,$$

while in some cases it also holds

$$A \cup B \Leftrightarrow_{entail(P,\mathcal{T}_i)} A \cup C \ .$$

In order to ensure (un)satisfiability preservation in the background theory, entailment rules must fulfill some conditions. If for every model M of $\mathcal{T}_i$, every valuation of the variables of B that satisfies B in M can be extended to a valuation that satisfy $B \cup C$ in M, then $A \cup B \Rightarrow_{entail^+(P,\mathcal{T}_i)} A \cup B \cup C$ preserves satisfiability in $\mathcal{T}$. Similarly, if for every model M of $\mathcal{T}_i$ it holds that every valuation of the variables of B that satisfies B in M can be extended to a valuation that satisfy C in M, and vice versa, then $A \cup B \Leftrightarrow_{entail(P,\mathcal{T}_i)} A \cup C$ preserves both satisfiability and unsatisfiability in $\mathcal{T}$.

If the entailment rule is applicable and changes the input formula, we say that it is successful.

The entailment rule, typically, has an essential role in combination/augmentation schemes. The rules $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$, $\Leftrightarrow_{unsat(\mathcal{T}_i)}$ and $\Leftrightarrow_{entail(P,\mathcal{T}_i)}$ are the only inference rules in our general setting that use domain specific knowledge (i.e., knowledge specific to the theories $\mathcal{T}_i$). Some other rules (such as 'abstraction') use information about these theories, but only information about their signatures.

EXAMPLE 1. *Nelson and Oppen's procedure for combinations of quantifier-free theories*[8] *is based on propagation of entailed equalities between different theories.*

A formula F is nonconvex *if F entails $x_1 = y_1 \vee x_2 = y_2 \vee \ldots x_n = y_n$, but for no i between 1 and n does F entail $x_i = y_i$. Otherwise, F is* convex. *A theory is* convex *if every conjunction of its literals is convex. For instance, Presburger rational arithmetic, the theory of lists with*

[8] The description of Nelson and Oppen's procedure and its scope is based on the original paper [37] and on [46].

car, cdr, cons, and atom and the pure theory of equality are all convex theories, while the theory of arrays with store and select, Presburger integer arithmetic and strict multiplication arithmetic are nonconvex theories.

For entailing equalities, it is sufficient to have a procedure for a given theory that can detect the unsatisfiability of a set of given literals. So, in the Nelson and Oppen's algorithm, procedures for specific theories can be treated as external functions. If there are n variables, we can try all possible disequalities between them[9] *and detect as entailed an equality* $x = y$ *if the current set of literals is unsatisfiable with* $\neg(x = y)$ *(in other words, the inference* $f \Leftrightarrow_{entail^+(NO,\mathcal{T}_i)} f \cup \{x = y\}$ *is equivalent to:* $f \cup \{\neg(x = y)\} \Leftrightarrow_{unsat(\mathcal{T}_i)} \bot$*). If a theory is nonconvex, we have to consider not only disequalities but also conjunctions of disequalities. When a conjunction of disequalities is detected as unsatisfiable, then its negation (a disjunction of equalities) is entailed, leading to a case split.*

For some theories there are efficient algorithms for detecting entailed equalities and Nelson and Oppen cited some of them in their original paper (e.g., the simplex algorithm for Presburger rational arithmetic).

We denote by NO entailment for a theory $\mathcal{T}_i$*, such that it entails an equality E (or a disjunctions of equalities if* $\mathcal{T}_i$ *is nonconvex) from a set B of (all)* $\mathcal{T}_i$*-literals. The entailed equality (or disjunction or equalities) E is added to a current set of literals, and we denote this entailment by*

$$A \cup B \Rightarrow_{entail^+(NO,\mathcal{T}_i)} A \cup B \cup E$$

or, alternatively (as it preserves unsatisfiability), by

$$A \cup B \Leftrightarrow_{entail^+(NO,\mathcal{T}_i)} A \cup B \cup E\ .$$

EXAMPLE 2. *Shostak's procedure*[10] *for a combination of quantifier-free theories deals with algebraically solvable* σ*-theories. A theory T is a* σ*-theory if there is a computable canonizer* σ *from terms to terms such that the following conditions hold (*$\equiv$ *denotes syntactical identity):*

1. *An equality* $t = u$ *in the theory is valid iff* $\sigma(t) \equiv \sigma(u)$*.*

2. $\sigma(x) \equiv x$*, for any variable* x*.*

3. $\sigma(\sigma(t)) \equiv \sigma(t)$*, for any term* t*.*

---

[9] As said in [46], variables can be considered restricted only to variables shared by some of component theories. However, in this paper we won't discuss this optimization of the original Nelson and Oppen's procedure.

[10] The description of Shostak's procedure and its scope is based on [43, 42, 19].

4. *If* $\sigma(t) \equiv f(t_1, \ldots, t_n)$ *(f is any function symbol), then* $\sigma(t_i) \equiv t_i$ *for* $1 \leq i \leq n$.

5. $\sigma(t)$ *does not contain any variables that are not already in* t.

For instance, each term in real linear arithmetic can be reduced to a canonized form $a_1x_1 + a_2x_2 + \ldots + a_nx_n + c$ *($n \geq 0$), where there is imposed some ordering on symbols* x_i *($0 \leq i \leq n$).*

A model M *is a* σ*-model if* $M \models t = \sigma(t)$ *for any term* t*, and* $M \not\models a = b$ *for distinct canonical, variable free terms* a *and* b*. A* σ*-theory* $\mathcal{T}$ *is algebraically solvable if there is a computable function, solve, that takes an equality* e *and returns* $\top$, $\bot$, *or a conjunction of equalities and has the following properties (let* $solve(e) \equiv E$*):*

1. E *and* e *are* σ*-equivalent; that is, for all* σ*-models* M *and valuations* α *over the variables in* e, $(M, \alpha) \models e$ *iff there is a valuation* α' *extending* α*, over the variables in* e *and* E *(recall that* E *can contain new variables not appearing in* e*), such that* $(M, \alpha') \models E$;

2. $E \in \{\top, \bot\}$ *or* $E \equiv \bigwedge_{i=1}^{k}(x_i = t_i)$;

3. *if* e *is unsatisfiable in any* σ*-model of* $\mathcal{T}$*, then* $E = \bot$;

4. *if* e *contains no variables, then* $E \in \{\top, \bot\}$;

5. *if* $E \equiv \bigwedge_{i=1}^{k}(x_i = t_i)$*, then the following hold:*
   - (a) *for all* i $(1 \leq i \leq k)$, x_i *occurs in* e;
   - (b) *for all* i *and* j $(1 \leq i, j \leq k)$, x_i *does not occur in* t_j;
   - (c) *for all* i *and* j $(1 \leq i, j \leq k)$, $i \neq j$ *implies* $x_i \not\equiv x_j$;
   - (d) $\sigma(t_i) \equiv t_i$.

Note that solve can detect unsatisfiability of an equality and, hence, unsatisfiability of a formula being proved. Moreover, whenever an equality is equivalent to $\bot$*, solve can detect it.*

Among algebraically solvable σ *theories are equational real linear arithmetic, equational integer linear arithmetic, the convex theory of lists, monadic set theory, and the like. For instance, a solver for real linear arithmetic takes an equality of the form* $a_1x_1 + \ldots + a_nx_n + c = b_1x_1 + \ldots + b_nx_n + d$ *and returns* $x_1 = ((b_2 - a_2)/(a_1 - b_1))x_2 + \ldots + ((b_n - a_n)/(a_1 - b_1))x_n + (d - c)$*; a solver for integer linear arithmetic takes an equality* $17x - 49y = 30$ *and returns* $x = 49z - 4$, $y = 17z - 2$*, where* z *is a new variable; a solver for the theory of lists takes* $cons(car(x)), cdr(car(y))) = cdr(cons(y, x))$ *and returns* $y = cons(cons(a, cdr(x)), d)$*, where* a *and* d *are new variables.*

If solve(e) is E, then we denote this kind of entailment by

$$A \cup \{e\} \Rightarrow_{entail(solve,\mathcal{T})} A \cup \{E\}$$

or, alternatively (as it preserves unsatisfiability), by

$$A \cup \{e\} \Leftrightarrow_{entail(solve,\mathcal{T})} A \cup \{E\}\ .$$

This rule solves just one equality in turn (i.e., it does not exhaustively solve all (unsolved) equalities from the given formula). In the above rule e can also be a disequality, that is, of the form $\neg d$, where d is equality; in that case, E is $\neg D$, where $solve(d)$ is equal to D. We say that an equality (or a disequality) e is solved *if e is equal to $solve(e)$. If all equalities and disequalities in a formula being proved are solved, the rule $\Leftrightarrow_{entail(solve,\mathcal{T})}$ is not applicable.*

EXAMPLE 3. *In [27] there is a description of a procedure (inspired by Fourier's method[11] [31]) for entailing implicit equalities E from Presburger rational arithmetic (PRA) inequalities B (see the same paper for limitations in entailing implicit equalities in PIA and PNA). We denote it by*

$$A \cup B \Rightarrow_{entail^+(impl-eqs,pra)} A \cup B \cup E$$

or, alternatively (as it preserves unsatisfiability), by

$$A \cup B \Leftrightarrow_{entail^+(impl-eqs,pra)} A \cup B \cup E\ .$$

EXAMPLE 4. *Variable elimination is a special case of entailment. Let us assume that there is a quantifier elimination procedure P for a theory $\mathcal{T}$ available (which can serve as a basis for a decision procedure for $\mathcal{T}$), that is, a procedure that transforms a formula f (of a theory $\mathcal{T}$) into an equivalent formula f' with fewer variables.*

If there is a subset B of given literals L ($L = A \cup B$) such that

- *all literals from B belong to $\mathcal{T}$,*
- *there is a variable x such that it does not occur in A*

then we can perform a variable elimination procedure P on the literals B and on the variable x, yielding a formula C (note that C may contain disjunctions).

Variable elimination procedures are usually aimed at the elimination of existentially quantified variables, and the elimination of universally quantified variables is easily reduced to it. However, typically, more

[11] More precisely, it is the Fourier and Motzkin's algorithm for linear inequalities. As in [27] we will refer to this algorithm briefly as Fourier's algorithm.

efficient versions of procedures of this kind can be obtained by making optimized eliminations for both existentially and universally quantified variables. Note that, in our setting, all variables are (implicitly) existentially quantified and, thus, it is sufficient to have available only elimination of existentially quantified variables.

We denote elimination of all variables fulfilling given conditions by

$$A \cup B \Rightarrow_{entail(P,\mathcal{T})} A \cup C$$

or, for some procedures and theories, by

$$A \cup B \Leftrightarrow_{entail(P,\mathcal{T})} A \cup C\ .$$

Some instances of variable elimination entailment are $\Leftrightarrow_{entail(Cooper,pia)}$, $\Leftrightarrow_{entail(Hodes,pra)}$, $\Leftrightarrow_{entail(Fourier,pra)}$, *and* $\Rightarrow_{entail(Hodes,pia)}$. *Fourier's method for variable elimination is used in system* TECTON *[27] and is essentially the same as Hodes' procedure used in* NQTHM *(for quantifier-free formulae). Note that Hodes' [23] procedure is an (un)satisfiability preserving procedure for Presburger rational arithmetic (*PRA*), but not for Presburger integer arithmetic (*PIA*). Cooper's [17] procedure is (un)-satisfiability-preserving for Presburger integer arithmetic and can be adapted for Presburger natural arithmetic (*PNA*), too.*

We also consider one relaxed form of entailment based on variable elimination: in this variant of entailment, a variable x *might occur (at most once) in* B *within an equality of the form* $x = f(t)$, *where* f *is an uninterpreted function symbol. This rule we denote by* $\Rightarrow_{entail^-(P,\mathcal{T})}$. *For instance,* $v_0 = f(x) \wedge b < v_0 \wedge v_0 < a \Rightarrow_{entail^-(Hodes,pra)} b < a$. *This rule is primarily intended for the elimination of abstraction variables (variables introduced by the abstraction rules).*

For a number of decidable theories there are decision procedures that work by using the idea of successive elimination of quantifiers from formula being proved [30].

Some of the variable elimination procedures can be flexibly implemented via rewrite rules [14, 16].

4.8. Constant Congruence Closure/Ground Completion

A constant congruence closure rule *ccc* is based on a relation $\simeq_C$ (which we define below) and on an algorithm for computing it. It can also serve as a decision procedure for the pure theory of equality. Equivalence classes given by constant congruence closure are then interreduced (giving a ground canonical system) and all literals in a formula being proved are normalized.

DEFINITION 1. *Given a signature* $\Lambda = (\Sigma, \emptyset)$ *and a set of variables* V*, the* constant congruence closure $\simeq_C$ *generated by a set* C *of equalities over* Λ*-terms is the smallest equivalence relation satisfying the following properties:*

- *for any equality* $t_1 = t_2$ *of* C*, it holds* $t_1 \simeq_C t_2$*;*
- *if* $t_1 \simeq_C t_2$ *and* f *is a function symbol from* Σ *of arity* n*, then* $f(u_1, \ldots, u_{i-1}, t_1, u_{i+1}, \ldots, u_n) \simeq_C f(u_1, \ldots, u_{i-1}, t_2, u_{i+1}, \ldots, u_n)$.

As said in [47], this relation is different from the equational congruence relation because in the constant congruence closure, a variable is treated as constant. That is, no instantiation for variables is allowed in the constant congruence closure.[12]

Constant congruence closure can be computed by using a congruence closure algorithm such as described in [43, 38], using a variant of the congruence closure algorithm interpreted as completion [26] or using Knuth-Bendix's completion algorithm for ground terms (each ground equality system admits a canonical rewrite system; for connections between congruence closure and ground completion and for deriving ground canonical system from congruence closure graphs, see also [9, 6]). As discussed in [47], we can also use only equality axioms, but the above methods are much more efficient.

In our general setting, we perform a constant congruence closure on a set of all equalities in a set of literals being refuted. This returns a set of equivalence classes; from each class we choose a minimal element (according to an ordering $\prec$) and introduce equalities for all remaining terms in that class. The equalities are then oriented as rewrite rules (according to the ordering $\prec$) and interreduced (while trivial equalities are eliminated). These oriented equalities make a ground canonical system (note that we treat variables as constants). They replace the initial set of equalities and are further used to normalize other literals in a formula being proved (literals other than equalities).[13] In addition, if in an obtained formula there is a literal $\neg(t = t)$ or literals l and $\neg l$, the current formula is replaced by $\perp$. This transformation we call *constant congruence closure* and denote it by $\Rightarrow_{ccc}$. It is unsatisfiability preserving, so we can also write $\Leftrightarrow_{ccc}$.

[12] In this paper, we deal only with quantifier-free theories, and we use the proof by refutation — thus, we can treat all variables in a negated conjecture as constants.

[13] We can also, within this rule only, treat all literals in a uniform way as equalities (and apply the ground completion procedures on all literals): If a literal ρ is not equality, we can represent it as $\rho = \top$, and if a literal ρ is not disequality, we can represent $\neg\rho$ as $\rho = \perp$.

EXAMPLE 5. *Let us suppose that the background theory includes the pure theory of equality with a function symbol f of arity 1. Let us suppose that a formula being proved is*

$$\{f(f(f(x))) = x, f(f(f(f(f(x))))) = x\} .$$

All subterms appearing in the given formula are partitioned into equivalence classes by a constant congruence closure algorithm, and, in this case, we get only one equivalence class:

$$\{x, f(x), f(f(x)), f(f(f(x))), f(f(f(f(x)))), f(f(f(f(f(x)))))\} .$$

This class gives the following set of equalities (oriented left to right):

$$\{f(x) = x, f(f(x)) = x, f(f(f(x))) = x, f(f(f(f(x)))) = x,$$
$$f(f(f(f(f(x))))) = x\} ,$$

which, after interreducing and eliminating trivial equalities, gives

$$\{f(x) = x\} .$$

Thus it holds that:

$$\{f(f(f(x))) = x, f(f(f(f(f(x))))) = x\} \Leftrightarrow_{ccc} \{f(x) = x\}$$

EXAMPLE 6. *Let us suppose that the background theory includes the pure theory of equality with a function symbol f of arity 1. The set of literals $\{\neg(f(a) = f(b)), a = b\}$ yields only one equivalence class $\{a, b\}$, and gives the following set of equalities (oriented left to right, assuming $a \prec b$): $\{b = a\}$. After normalizing, $\neg(f(a) = f(b))$ becomes $\neg(f(a) = f(a))$ and, hence, it holds that:*

$$\{\neg(f(a) = f(b)), a = b\} \Leftrightarrow_{ccc} \perp .$$

4.9. Superposing

If $\mathcal{R}$ is a canonical rewrite system such that for any finite set of ground equalities E (expressed by using symbols in $\mathcal{R}$ and constants) completion terminates producing a finite canonical rewrite system $\mathcal{R}'$, then $\mathcal{R}$ can be used to decide whether the conclusion of a conditional equality, whose conditions constitute E, follows from $\mathcal{R}$ or not. We call a canonical system $\mathcal{R}$ with this property an *admissible* rewrite system.[14] For an

[14] The definition of the admissible rewrite system and the examples given in this subsection are from [27]. This macro inference rule is inspired by and based on this work.

admissible $\mathcal{R}$, its quantifier-free theory is decidable using completion. If a rule $l \longrightarrow r$ from $\mathcal{R}$ can be superposed with some equality e from the formula f being proved (and we treat variables in f as constants), the obtained equality e' can replace the equality e. In addition, all literals have to be normalized by the rules from $\mathcal{R}$. We denote this inference by $\Rightarrow_{superpose(\mathcal{R})}$ or, alternatively (as it preserves unsatisfiability), by $\Leftrightarrow_{superpose(\mathcal{R})}$.

EXAMPLE 7. *It can be shown that the rewrite system* $\{f(f(x)) \longrightarrow x\}$ *is admissible. Let* $f(a) = f(b) \wedge a \neq b$ *is the formula being refuted. Then* $f(a)$ *superposes with* $\{f(f(x)) \longrightarrow x\}$ *yielding the new equality* $a = f(f(b))$ *with a normalized form* $a = b$*. Then it is trivially shown that the formula* $a = b \wedge a \neq b$ *is unsatisfiable.*

The above example illustrates the fact that normalization with rules from the admissible system is not enough by itself but that superposing with equalities in a formula being refuted is needed.

EXAMPLE 8. *The theory of lists with car, cdr, cons, and atom is given by the following axioms.*

1. $cons(car(x), cdr(x)) = x$,
2. $car(cons(x, y)) = x$,
3. $cdr(cons(x, y)) = y$.

The following rewrite system

1. $cons(car(x), cdr(x)) \longrightarrow x$,
2. $car(cons(x, y)) \longrightarrow x$,
3. $cdr(cons(x, y)) \longrightarrow y$

is an admissible rewrite system.

4.10. Lemma Invoking

We assume that there may be some additional rewrite rules, definitional equalities, additional theorems, and lemmas[15] available (we do

[15] For instance, let us assume that we deal with a theory $\mathcal{T}$ that includes Peano arithmetic. Peano arithmetic is undecidable, so we can't have a decision procedure for it, but there are some decidable subtheories of Peano arithmetic (such as Presburger arithmetic) that are decidable. So, for example, we can have a decision procedure for Presburger arithmetic and some of the axioms concerning multiplications as additional rewrite rules.

not address the problem of discovering or speculating lemmas — see, for instance, [29, 28, 2]). We will consider only (quantifier-free) definitional equalities, theorems, and lemmas of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow l = r$ and $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho$. We will also consider conditional rewrite rules (i.e., rules of the form $l \longrightarrow r$ **if** $p_1, p_2, \ldots, p_k$ and $\rho \longrightarrow \top$ **if** $p_1, p_2, \ldots, p_k$). Thus, we will treat all of them in a similar manner as lemmas of corresponding forms.[16] We assume (in order to ensure termination) that there is a reduction ordering $\prec$ available, such that for each lemma $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow l = r$ it holds $r \prec l$ and $p_i \prec l$ (for $i = 1, 2, \ldots, k$) and for each lemma $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho$ it holds $p_i \prec \rho$ (for $i = 1, 2, \ldots, k$).

If our proving strategy is based on decision procedures for some theories $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_j$ we assume that available lemmas (i.e., available additional rules) make a conservative extension to each of them. Then it is only sensible to search for lemmas that are not formulae of this theories; indeed, no lemma that belongs to some of these theories can contain information that could not be derived by decision procedures for $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_j$. Thus, we formulate the lemma invoking inference with respect to theories $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_j$.

Let t be a maximal alien term (with respect to the combination of $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_j$) in the formula f being refuted (and there is no alien literal l such that $t \prec l$). Let c be a new abstraction variable for the term t (if there is no equality $t = c$ in the formula already). Let us assume that there is a (quantifier-free) lemma $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho$ available in the set $\mathcal{L}$ such that a maximal alien term in ρ (w.r.t. $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_j$) is a term t' and there is a most general substitution ϕ such that $t'\phi$ is equal to t. Also, let us assume that all variables occurring in the lemma are instantiated by ϕ. The formula f is transformed into $f \wedge (p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho)\phi$, and in this formula all occurrences of t are replaced by c, yielding a resulting formula f' (and leading to a case split if $k > 0$). If there is no such lemma,[17] then all occurrences of t are replaced by c

[16] Note that these additional rewrite rules do not necessarily need to be lemmas (i.e., theorems of the background theory); within our system we don't examine the status of these rules, and therefore it would be more appropriate to consider these rules as additional hypotheses. Hence, if this macro inference rule (with the set $\mathcal{L}$ of additional rules) is used in proving a formula F, then we have to write $\mathcal{T}, \mathcal{L} \vdash F$ (rather than $\mathcal{T} \vdash F$). However, because additional rules are sometimes indeed lemmas (in a strict sense) of the background theory and for tradition reasons, we use the name "lemma-invoking" for this rule and for dealing with all additional rewrite rules.

[17] For some conjectures with alien terms 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. It makes this relaxed condition of *lemma* rule sensible. In interactive theorem provers, in these situations the user should be informed that the refutation attempt is being

yielding a resulting formula f'. This inference we denote by

$$f \Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_j)} f' \ .$$

EXAMPLE 9. *Let the background theory be* PRA *extended by* $\mathcal{L} = \{max(x,y) = x \rightarrow min(x,y) = y\}$. *Let us suppose that it holds* $max \prec min$. *Let the formula being refuted be*

$$\{1 \leq v,\ b \leq 0,\ max(a,b) = a,\ min(a,b) = v\}.$$

The maximal alien term (w.r.t. PRA*) in the given formula is* $min(a,b)$, *the maximal alien term in* $min(x,y) = y$ *is* $min(x,y)$. *The substitution* $\phi = \{x \mapsto a, y \mapsto b\}$ *fulfills the required conditions, and the given formula is rewritten to*

$$\{1 \leq v,\ b \leq 0,\ max(a,b) = a,\ min(a,b) = v, \neg(max(a,b) = a) \vee min(a,b) = b\}.$$

All occurrences of $min(a,b)$ *are replaced by* v, *yielding*

$$\{1 \leq v,\ b \leq 0,\ max(a,b) = a,\ v = v, \neg(max(a,b) = a) \vee v = b\}.$$

Now, there is a case split giving two conjunctions of literals:

$$\{1 \leq v,\ b \leq 0,\ max(a,b) = a,\ v = v, \neg(max(a,b) = a)\}$$

and

$$\{1 \leq v,\ b \leq 0,\ max(a,b) = a,\ v = v, v = b\},$$

both leading to contradiction.

Let $\neg\varrho$ be a maximal alien literal (ϱ is an atomic formula, alien with respect to the combination of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_j$) in the formula f being refuted (and there is no alien term t in f such that $\varrho \prec t$). Let us assume that there is a (quantifier-free) lemma $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho$ available in the set $\mathcal{L}$ such that there is a most general substitution ϕ such that $\rho\phi$ is equal to ϱ. Also, let us assume that all variables occurring in the lemma are instantiated by ϕ. The formula f is transformed into $f \wedge (p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \rho)\phi$ and in this formula all occurrences of ϱ are replaced by $\perp$, yielding a formula f' (and leading to a case split if $k > 0$). If there is no such lemma, then all occurrences of ϱ are replaced by $\perp$ yielding a resulting formula f'. This inference we (also) denote by

$$f \Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_j)} f' \ .$$

The rule is dual if ϱ is a maximal alien literal in the formula f being refuted such that there is no an alien term t in f such that $\varrho \prec t$: we look for a lemma $p_1 \wedge p_2 \wedge \ldots \wedge p_k \rightarrow \neg\rho$, and all occurrences of ϱ are replaced by $\top$.

tried without any lemma and that he/she should consider adding some lemma in the case of the final failure of the procedure.

EXAMPLE 10. *Let the background theory be* PRA*, extended by* $\mathcal{L} = \{g(x) < h(x) \rightarrow p(x)\}$*, and let us suppose that* $g \prec p$ *and* $h \prec p$*. Let the formula being refuted be*

$$\{\neg p(a), g(a) < h(a)\}.$$

The maximal alien literal (w.r.t. PRA*) in the formula* f *is* $\neg p(a)$*. The substitution* $\phi = \{x \mapsto a\}$ *fulfills the required conditions, and the formula* f *is rewritten to*

$$\{\neg p(a),\ g(a) < h(a),\ \neg(g(a) < h(a)) \vee p(a)\}.$$

All occurrences of $p(a)$ *are replaced by* $\perp$*, yielding*

$$\{\neg\perp,\ g(a) < h(a),\ \neg(g(a) < h(a)) \vee \perp\}.$$

Now, there is a case split giving two conjunctions of literals:

$$\{\neg\perp,\ g(a) < h(a),\ \neg(g(a) < h(a))\}$$

and

$$\{\neg\perp,\ g(a) < h(a),\ \perp\}$$

both leading to contradiction.

The above lemma-invoking mechanism is a relaxed form of the one used in [27]. Using the above approach, the refutation process continues even if there is no lemma available for the current maximal alien term/literal. Thus, the rule $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_j)}$ is applicable even if $\mathcal{L}$ is the empty set. Note that $\Rightarrow_{entail^-(P,\mathcal{T})}$ has similar effects as the combination of the rules $\Rightarrow_{lemma^+(\emptyset,\mathcal{T})}$ and $\Leftrightarrow_{entail(P,\mathcal{T})}$ (where P is a variable elimination procedure for the theory $\mathcal{T}$).

If there is more than one lemma fulfilling the required conditions, then in the case of failing to prove the unsatisfiability, one can backtrack and try then another lemma. As previously said, even if all lemmas are unsuccessfully tried, one can try to continue the refuting process with no help from any lemma.

In [27, 29, 1], when using conditional rules, for each condition there is one subproof, and this leads to tree-structured proofs. It is analogous to the lemma-invoking mechanism proposed here, since for the conclusion and for each condition from the used conditional rule, there is one disjunction (after $\Leftrightarrow_{dnf}$ and $\Leftrightarrow_{split}$) that has to be refuted.

In cases when the background theory is the combination of theories $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ and $k > 1$, it is sensible to apply the described mechanism not only to alien terms and literals, but to all terms and literals. There are also some other variants of the above rule for lemma invoking [47, 27, 29, 1, 25] (e.g., the use of some lemmas can be restricted according to a dominating function symbol), but we won't discuss them in this paper.[18]

Note that the combination of the rules $\Leftrightarrow_{ccc}$ and $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T})}$ (together with $\Leftrightarrow_{dnf}$ and $\Leftrightarrow_{split}$) is equivalent to contextual rewriting [47], up to duality (contextual rewriting is used for simplification of clauses) and to restrictions in the lemma mechanism.

Note that we usually leave the realm of completeness when we start to rely on lemmas [27, 29, 25]. Moreover, even when $\mathcal{T}, \mathcal{L}, f \vdash \bot$, it might be impossible to prove the unsatisfiability of f using the above lemma mechanisms (or using the lemma mechanism discussed in [11, 27, 1]). This is illustrated by the following example (taken from [47]):

EXAMPLE 11. *Let $\mathcal{L}$ be the set of the following two rules/lemmas*

$$q \rightarrow \neg p(a, x)\ ,$$

$$\neg q \rightarrow \neg p(x, b)\ .$$

Then $\neg p(a, b)$ is a logical consequence of $\mathcal{L}$, but the unsatisfiability of $p(a, b)$ cannot be proved by using either the first or the second rule.

For handling this problem, a relaxed variant of contextual rewriting might be used [49], which does not require the conditions of a rewrite rule to be proved in order to apply a rule. For keeping completeness when applying additional rewrite rules, a notion of *well-coveredness* is proposed in [49] and some variants of it in [10].

The lemma-invoking mechanism presented here can be adjusted in order to handle this problem in some situations. Specifically, instead of using just one lemma that meets given conditions, we can use all such available lemmas. Then, in the above example, the conjecture $p(a, b)$ by the lemma rule would be transformed into $p(a, b) \wedge (q \rightarrow \neg p(a, b)) \wedge (\neg q \rightarrow \neg p(a, b))$, and by the rules $\Leftrightarrow_{dnf}$ and $\Leftrightarrow_{split}$ it is further transformed into four subgoals: $p(a, b) \wedge q \wedge \neg q$, $p(a, b) \wedge q \wedge \neg p(a, b)$, $p(a, b) \wedge \neg p(a, b) \wedge \neg q$, and $p(a, b) \wedge \neg p(a, b) \wedge \neg p(a, b)$, each of which easily leads to contradiction. This version of the lemma rule extends the realm of the basic lemma-invoking mechanism and of the general setting itself.

[18] Some authors refer to lemma invoking as an *augmentation step* or an *augmentation heuristic*.

4.11. Properties of Macro Inference Rules

It is not difficult to prove that all the presented macro inference rules (in Section 4) are sound and that all $\Leftrightarrow_X$ rules preserve unsatisfiability.

The properties of $\Leftrightarrow_{entail(solve,\mathcal{T})}$ are implied by the properties of the *solve* function. Proofs of the properties of the rules $\Leftrightarrow_{entail^+(impl-eqs,pra)}$, $\Leftrightarrow_{entail(Fourier,pra)}$, $\Leftrightarrow_{entail(Cooper,pia)}$ and $\Leftrightarrow_{entail(Hodes,pra)}$ are based on the proofs from [31], [27], [23], and [17], respectively. Soundness of $\Rightarrow_{entail(Hodes,pia)}$ can be proved on the basis of soundness of the rule $\Leftrightarrow_{entail(Hodes,pra)}$ and by a simple model-theoretic argument. The proofs of the properties of $\Leftrightarrow_{ccc}$ are based on the properties of ground completion (see, for instance, [6]). The proofs of the properties of $\Leftrightarrow_{superpose(\mathcal{R})}$ are based on [27]. Satisfiability and unsatisfiability preservation of the remaining rules can be easily proved on the basis of the properties of first-order theories with equality. Thus, the following theorem holds (f is a quantifier-free formula):

THEOREM 1. *For a given background theory $\mathcal{T}$, if $f \Rightarrow^* f'$ and if f' is unsatisfiable in $\mathcal{T}$, then f is unsatisfiable in $\mathcal{T}$, too. If $f \Leftrightarrow^* f'$ and if f is unsatisfiable in $\mathcal{T}$, then f' is unsatisfiable in $\mathcal{T}$, too.*

The immediate consequence of the above theorem is the following: if it holds that $f \Rightarrow^* \bot$ (or $f \Leftrightarrow^* \bot$), then f is unsatisfiable and $\neg f$ is valid; if it holds that the background theory $\mathcal{T}$ is consistent and $f \Leftrightarrow^* \top$, then f is satisfiable and $\neg f$ is invalid. If it holds that $f \Rightarrow^* \top$ (and does not hold that $f \Leftrightarrow^* \top$), then it is *unknown* whether $\neg f$ is valid or invalid.

The termination of each inference rule can be proved in a simple manner (with the exception of $\Leftrightarrow_{ccc}$ for which the termination can be ensured by the termination argument for the constant congruence closure algorithm or by the termination argument for ground completion). However, termination of a scheme built from several rules must be proved separately.

In this paper, we won't discuss the complexity of the macro inference rules and the schemes built from them. The computational cost of the macro inference rules ranges from linear to superexponential (for Cooper's elimination of variables from Presburger formulae [39]). The overall computational cost of combination/augmentation schemes depends on the complexity of inference rules used.

5. Case Studies

In this section we describe several methods for combining and augmenting decision procedures in the framework of the macro inference rules introduced in Section 4. We point out that these descriptions do not represent fully the original methods[19] but give their key ideas. In a number of aspects the given descriptions can be optimized (as in the corresponding systems). However, the point of the given general setting is in gathering some shared ideas that can serve as a basis for a new, powerful, and flexible framework for combining and augmenting decision procedures. We believe that the potential losses in efficiency are dominated by these advantages.

Instead of a description of Boyer and Moore's procedure within GS, we give descriptions of the TECTON-style and EPM-style schemes, which are generalizations of Boyer and Moore's approach.

In the given descriptions we use the phrase "apply [an inference rule]", while it would be more precisely to say "try to apply [an inference rule]". Some inference rules are always applicable, though possibly without any effect (for instance, the *ccc* rule).

In the following schemes, when used, the rule $\Leftrightarrow_{simpl(\mathcal{T}_i)}$ can be replaced by $\Leftrightarrow_{simpl}$, but the versions with $\Leftrightarrow_{simpl(\mathcal{T}_i)}$ are typically more efficient.

5.1. Nelson and Oppen's Combination of Theories

Let a quantifier-free theory $\mathcal{T}$ be a combination of theories $\mathcal{T}_i$ ($1 \leq i \leq k$) that do not share nonlogical symbols other than equality. Let $\mathcal{T}$ be a background theory, and let F be a formula to be proved. The procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$.
4. Apply $\Leftrightarrow_{dnf}$.
5. Apply $\Leftrightarrow_{split}$.
6. Apply $\Leftrightarrow_{unsat(\mathcal{T}_i)}$, for some i, $1 \leq i \leq k$.
7. Apply $\Leftrightarrow_{entail^+(NO,\mathcal{T}_i)}$, for some $1 \leq i \leq k$; if not successful, return $\top$.

[19] This is especially the case with the implementation of Shostak's scheme, whose original version is very compact and optimized.

8. Apply $\Leftrightarrow_{repl^=}$.

9. Go to Step 4.

Note that if the theories $\mathcal{T}_i$ are convex, then in Step 9 we can jump to Step 6, instead of to Step 4. The replacement carried by the rule $\Leftrightarrow_{repl^=}$ was not used in the original procedure. This rule is useful as it reduces the number of considered variables in the search for implicit equalities.

EXAMPLE 12. *Let*

$$x \leq y \wedge y \leq x + car(cons(0,x)) \wedge p(h(x) - h(y)) \rightarrow p(0)$$

be the formula we need to prove [37]. It belongs to the combination of PRA*, the theory of lists (with elements rational numbers), and the pure theory of equality over the signature* $(\{h\},\{p\})$ *(denoted* $\mathcal{E}$*). Its negation is* $x \leq y \wedge y \leq x + car(cons(0,x)) \wedge p(h(x) + (-1) \cdot h(y)) \wedge \neg p(0)$*, and we need to show that it is unsatisfiable. To save space, we omit applications of rules that are unsuccessful or don't have any effect:*

$\{x \leq y, y \leq x + car(cons(0,x)), p(h(x) + (-1) \cdot h(y)), \neg p(0)\}$

$\Leftrightarrow_{absp^+(pra,lists,\mathcal{E})}$

$\{x \leq y, y \leq x+v_0, car(cons(v_1,x)) = v_0, 0 = v_1, p(v_2), v_4+(-1)\cdot v_3 = v_2, h(y) = v_3, h(x) = v_4, \neg p(v_1)\}$

$\Leftrightarrow_{entail^+(NO,lists)}$

$\{x \leq y, y \leq x+v_0, car(cons(v_1,x)) = v_0, 0 = v_1, p(v_2), v_4+(-1)\cdot v_3 = v_2, h(y) = v_3, h(x) = v_4, \neg p(v_1), v_1 = v_0\}$

$\Leftrightarrow_{repl^=}$

$\{x \leq y, y \leq x+v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4+(-1)\cdot v_3 = v_2, h(y) = v_3, h(x) = v_4, \neg p(v_0)\}$

$\Leftrightarrow_{entail^+(NO,pra)}$

$\{x \leq y, y \leq x+v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4+(-1)\cdot v_3 = v_2, h(y) = v_3, h(x) = v_4, \neg p(v_0), x = y\}$

$\Leftrightarrow_{repl^=}$

$\{x \leq x, x \leq x + v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4 + (-1) \cdot v_3 = v_2, h(x) = v_3, h(x) = v_4, \neg p(v_0)\}$

$\Leftrightarrow_{entail^+(NO,\mathcal{E})}$

$\{x \leq x, x \leq x + v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4 + (-1) \cdot v_3 = v_2, h(x) = v_3, h(x) = v_4, \neg p(v_0), v_3 = v_4\}$

$\Leftrightarrow_{repl^=}$

$\{x \leq x, x \leq x + v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4 + (-1) \cdot v_4 = v_2, h(x) = v_4, h(x) = v_4, \neg p(v_0)\}$

$\Leftrightarrow_{entail^+(NO,pra)}$

$\{x \leq x, x \leq x + v_0, car(cons(v_0,x)) = v_0, 0 = v_0, p(v_2), v_4 + (-1) \cdot v_4 = v_2, h(x) = v_4, h(x) = v_4, \neg p(v_0), v_0 = v_2\}$

$\Leftrightarrow_{repl^=}$

$\{x \leq x, x \leq x + v_2, car(cons(v_2, x)) = v_2, 0 = v_2, p(v_2), v_4 + (-1) \cdot v_4 = v_2, h(x) = v_4, h(x) = v_4, \neg p(v_2)\}$

$\Leftrightarrow_{unsat(\mathcal{E})}$

$\bot$

In order to prove that the above procedure terminates and is correct, we need the following theorem [46] (the theorem is used in [46] for proving the correctness of the nondeterministic version of Nelson and Oppen's procedure).

THEOREM 2. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite, signature-disjoint, quantifier-free theories, and let ϕ_1 and ϕ_2 be conjunctions of literals from $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Let V be the set of variables shared by ϕ_1 and ϕ_2, and let ψ be the conjunctions of all disequalities $x_i \neq x_j$ such that $x_i, x_j \in V$ and $i \neq j$. If $\phi_1 \wedge \psi$ is satisfiable in $\mathcal{T}_1$ and $\phi_2 \wedge \psi$ is satisfiable in $\mathcal{T}_2$, then $\phi_1 \wedge \phi_2$ is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$.*

We will prove the correctness of the given, the Nelson-Oppen-style procedure for the case of two component theories; the proof in the general case is a simple generalization.

THEOREM 3. *If $\mathcal{T}_i$ $(1 \leq i \leq k)$ are stably infinite, signature-disjoint, quantifier-free theories, then the above Nelson-Oppen-style procedure terminates and is correct (complete and sound); that is, it returns $\bot$ if $\neg F$ is unsatisfiable (i.e., F is valid) and returns $\top$ if $\neg F$ is satisfiable (i.e., F is invalid).*

Proof. There is only one application of the rule $\Leftrightarrow_{absp^+}$, and after that there is no introduction of new variables. On the other hand, after each successful application of $\Leftrightarrow_{entail^+}$, the rule $\Leftrightarrow_{repl^=}$ is applied that eliminates at least one variable. In the case of nonconvex theories, there are a finite number of branches, and in each of them there is at least one variable eliminated by the rule $\Leftrightarrow_{repl^=}$. Therefore, in each branch the number of variables strictly decreases (in iterations), and the rule $\Leftrightarrow_{entail^+}$ cannot be applied an infinite number of times. Hence, the procedure terminates.

The procedure can return only $\bot$ or $\top$. All macro inference rules in the setting are sound, so if the procedure returns $\bot$, the formula $\neg F$ is unsatisfiable. If the procedure returns $\top$, it means that at some stage $\Leftrightarrow_{entail^+}$ failed to entail a new equality (or a new disjunction of equalities). At that stage, the restrictions ϕ_1 and ϕ_2 of the current formula ϕ only to $\mathcal{T}_1$ or $\mathcal{T}_2$ literals are satisfiable formulae (otherwise $\Leftrightarrow_{unsat}$ would have detected unsatisfiability). Since $\Leftrightarrow_{entail^+}$ failed to entail any equality, it means that the formulae ϕ_1 and ϕ_2 are satisfiable with conjunction ψ' of all disequalities between all variables. Then, trivially, the formulae ϕ_1 and ϕ_2 are satisfiable with conjunction ψ of all disequalities between only shared variables (we can consider only shared variables anyway). By Theorem 2, it means that the formula ϕ is also satisfiable. All other used macro inference rules are known to be satisfiability preserving, and by the properties of $\Leftrightarrow_{split}$, it follows that the original formula is satisfiable. Thus, the procedure returns $\perp$ if and only if the input formula $\neg F$ is unsatisfiable.

5.2. Shostak's Combination of Theories

Let a quantifier-free theory $\mathcal{T}$ be a combination of algebraically solvable σ-theories $\mathcal{T}_i$ $(1 \leq i \leq k)$ and the pure theory of equality $(\mathcal{E})$. Let $\mathcal{T}$ be a background theory, and let F be a quantifier-free formula to be proved. The (variant of the) procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{ccc}$.
4. Apply $\Leftrightarrow_{simpl}$.
5. Apply $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k,\mathcal{E})}$.
6. Apply $\Leftrightarrow_{entail(solve,\mathcal{T}_i)}$, for some i, $1 \leq i \leq k$; if not successful, return $\top$.
7. Apply $\Leftrightarrow_{repl}$.
8. Go to Step 1.

Note that we use solvers for specific theories aided by abstraction instead of a combined solver for the theory $\mathcal{T}$. In the rule $\Leftrightarrow_{entail(solve,\mathcal{T}_i)}$, abstraction variables are solved first, and in the rule $\Leftrightarrow_{repl}$, abstraction variables are replaced first; in this way, after each iteration there are no abstraction variables in a formula being proved. If some variable is solved in one equality, then we don't try to solve it in another equality.

In the original Shostak algorithm (and in its variants) solvers for specific theories are tightly combined by specific data structures and functions such as $find$ and σ [19]. In the case of the theory of equality with uninterpreted function symbols $find$ gives the canonical representative of a term with respect to the equality already processed. In the case of the combination of algebraically solvable theories (and without uninterpreted functions), σ returns the canonical representative of a term in the terms of the theories used. As said in [19], Shostak's procedure is a method for efficiently combining these two canonical forms. In the above Shostak-style procedure, we use solvers for specific theories and combine their results by congruence closure, but this combination is not so tight (and not so efficient) as in the original Shostak procedure.

In the original Shostak algorithm, disequalities are not processed as equalities but have the following role: after all equalities are processed, for both sides of each disequality "normal" forms (w.r.t. processed equalities and the current state of the main corresponding data structures) are computed and checked for syntactical identity; if in some disequality two sides have the same "normal" form, the given formula is unsatisfiable. In our variant of Shostak's algorithm, it is sufficient to have disequalities canonized and rewritten by ground canonical systems induced by equalities. However, for simplicity, we can also treat them (i.e., solve them) in the same manner as equalities. Since a solved equality can be replaced by a conjunction of equalities, a solved disequality can be replaced by a disjunction of disequalities. Thus we have to iterate the procedure from Step 1.

EXAMPLE 13. *Let*

$$z = f(x + (-1) \cdot y) \wedge x = y + z \rightarrow y + f(f(z)) = x$$

be the formula we need to prove [43]. It belongs to the combination of equational real linear arithmetic and the pure theory of equality over the signature $(\{f\}, \emptyset)$ *(denoted* $\mathcal{E}$*). Its negation is* $z = f(x+(-1)\cdot y) \wedge x = y+z \wedge \neg(y+f(f(z)) = x)$*, and we need to show that it is unsatisfiable. For simplicity, we omit applications of rules that are unsuccessful or don't have any effect:*

$\{z = f(x + (-1) \cdot y), x = y + z, \neg(y + f(f(z)) = x)\}$
$\Leftrightarrow_{ccc}$
$\{f(x + (-1) \cdot y) = z, y + z = x, \neg(y + f(f(z))) = x)\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{E})}$
$\{f(v_1) = z, x + (-1) \cdot y = v_1, z + y = x, \neg(y + v_0 = x), f(f(z)) = v_0\}$
$\Leftrightarrow_{entail(solve,pra)}$
$\{f(v_1) = z, x+(-1)\cdot y = v_1, z+y = x, \neg(v_0 = x+(-1)\cdot y), f(f(z)) = v_0\}$
$\Leftrightarrow_{repl}$
$\{f((z + y) + (-1) \cdot y) = z, \neg(f(f(z)) = (z + y) + (-1) \cdot y)\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{E})}$
$\{f(v_1) = z, (z + y) + (-1) \cdot y = v_1, \neg(f(f(z)) = v_0), v_0 = (z + y) + (-1) \cdot y\}$

$\Leftrightarrow_{entail(solve,pra)}$
$\{f(v_1) = z, (z + y) + (-1) \cdot y = v_1, \neg(f(f(z)) = v_0), v_0 = z\}$
$\Leftrightarrow_{repl}$
$\{f((z + y) + (-1) \cdot y) = z, \neg(f(f(z)) = z)\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{E})}$
$\{f(v_0) = z, (z + y) + (-1) \cdot y = v_0, \neg(f(f(z)) = z)\}$
$\Leftrightarrow_{entail(solve,pra)}$
$\{f(v_0) = z, z = v_0, \neg(f(f(z)) = z)\}$
$\Leftrightarrow_{repl}$
$\{f(z) = z, \neg(f(f(z)) = z)\}$
$\Leftrightarrow_{ccc}$
$\perp$

In Shostak's original paper [43] and in some of the papers on it [19] there were proofs that Shostak's procedure terminates and that it returns $\perp$ if and only if the input formula is unsatisfiable. However, Rueß and Shankar recently showed [42] that all those proofs were flawed and that Shostak's procedure and all its published variants [19, 7, 9] are incomplete and can even be nonterminating.[20] The above Shostak-style procedure terminates for the conjecture for which the original procedure loops. However, termination of the above procedure is yet to be formally explored or proved. The above procedure is sound (because all macro inference rules are sound), and, like the original Shostak algorithm, it is incomplete.[21]

THEOREM 4. *If $\mathcal{T}_i$ ($1 \leq i \leq k$) are algebraically solvable quantifier-free σ-theories, the above Shostak-style procedure is sound; that is, it returns $\perp$ only if $\neg F$ is unsatisfiable (i.e., if F is valid).*

In this paper we won't try to mimic the recent variants of Shostak's procedure proposed in [42, 8].

5.3. Reasoning about Numbers in Tecton

There is a description of the reasoning about numbers in Tecton [27, 29] with three procedures based on Presburger arithmetic. Theories pna and pia are also considered, but we will discuss only the variant

[20] Rueß and Shankar in [42] give a simple example for incompleteness of Shostak's algorithm: The conjunction $f(v-1) - 1 = v + 1, f(u) + 1 = u - 1, u + 1 = v$ is unsatisfiable, but it cannot be established by Shostak's algorithm. Moreover, it loops for the following input conjunction: $f(v) = v, f(u) = u - 1, u = v$.

[21] The Shostak-style procedure presented here also cannot prove the unsatisfiability of the conjunction $f(v-1) - 1 = v + 1, f(u) + 1 = u - 1, u + 1 = v$ (and with the analogous explanation as for the original procedure).

for Presburger rational arithmetic (PRA). The pure theory of equality is denoted by $\mathcal{E}$. As in the original paper, only quantifier-free formulae are considered. Simplification of PRA terms is used, and it is similar to the simplification based on Shostak's canonizers (the corresponding simplification is not described explicitly in the original papers). We discuss all three procedures for PRA proposed in [27], but we focus on the third procedure, which uses lemmas as additional rules.

5.3.1. *Presburger Rational Arithmetic with Uninterpreted Symbols*
Let F be a formula of quantifier-free Presburger rational arithmetic with uninterpreted symbols. The procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{simpl(pra)}$.
4. Apply $\Leftrightarrow_{ccc}$.
5. Apply $\Leftrightarrow_{absp^+(pra,\mathcal{E})}$.
6. Apply $\Leftrightarrow_{unsat(pra)}$.
7. Apply $\Leftrightarrow_{entail^+(impl-eqs,pra)}$; if successful, go to Step 1.
8. Apply $\Rightarrow_{entail^-(Fourier,pra)}$.
9. Go to Step 1.

The original procedure is a decision procedure for PRA with the pure theory of equality [27]. Note that in Step 9 we have to jump to Step 1 (instead to Step 3) because variable elimination may introduce disjunctions. In the case of the theory PRA, disjunctions may be introduced by a variable elimination if there are disequalities in a formula being proved. In the original procedure [27] all disequalities are removed in the beginning (before transforming into disjunctive normal form). We made this slight modification of the original procedure because we intend to formulate the procedures from [27] in such a way that they can be easily formulated for some other theory and some other underlying decision procedure (and in some cases introduction of disjunctions by a variable elimination might not be avoidable). Similarly, the rule $\Leftrightarrow_{entail^+(impl-eqs,\mathcal{T})}$, when applied to a nonconvex theory $\mathcal{T}$, may introduce disjunctions of equalities, so after Step 7, in a general case, we have to jump to Step 1 (instead to Step 3). The theory PRA is convex, the rule $\Leftrightarrow_{entail^+(impl-eqs,\mathcal{T})}$ does not introduce disjunctions, and in this special case we can jump to step 3 after step 7. The same explanation applies to the following procedures as well.

5.3.2. *Presburger Rational Arithmetic and Admissible Rewrite Systems*

Let $\mathcal{R}$ be an admissible rewrite system. Let F be a formula of quantifier-free Presburger rational arithmetic with uninterpreted symbols and with the quantifier-free theory of $\mathcal{R}$. The procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{simpl(pra)}$.
4. Apply $\Leftrightarrow_{ccc}$.
5. Apply $\Leftrightarrow_{absp^+(pra,\mathcal{E})}$.
6. Apply $\Leftrightarrow_{unsat(pra)}$.
7. Apply $\Leftrightarrow_{superpose(\mathcal{R})}$; if successful, go to Step 3.
8. Apply $\Leftrightarrow_{entail^+(impl-eqs,pra)}$; if successful, go to Step 1.
9. Apply $\Rightarrow_{entail^-(Fourier,pra)}$.
10. Go to Step 1.

The original procedure is a decision procedure for PRA with the pure theory of equality and with interpreted functional symbols axiomatized by an admissible rewrite system [27].

5.3.3. *Presburger Rational Arithmetic with Conditional Rewrite Rules*

Let $\mathcal{T}$ be the combination of PRA and the pure theory of equality $\mathcal{E}$, extended by $\mathcal{L}$ (while the signature of $\mathcal{E}$ includes all function and predicate symbols from PRA and $\mathcal{L}$). Let F be a quantifier-free formula of the theory $\mathcal{T}$. The procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{absp^+(pra,\mathcal{L})}$.
4. Apply $\Leftrightarrow_{simpl(pra)}$.

5. Apply $\Leftrightarrow_{ccc}$.

6. Apply $\Leftrightarrow_{unsat(pra)}$.

7. Apply $\Leftrightarrow_{entail(Fourier,pra)}$; if successful, go to Step 1.

8. Apply $\Leftrightarrow_{entail^+(impl-eqs,pra)}$; if successful, go to Step 1.

9. Apply $\Rightarrow_{lemma^+(\mathcal{L},pra)}$.

10. Go to Step 1.

If the above procedure returns $\bot$, the formula $\neg F$ is unsatisfiable and F is valid [27]. However, the above procedure is not complete and it can also return $\top$ even if the formula $\neg F$ is unsatisfiable (it might be the case when $\neg F \Rightarrow^* \top$ and not $\neg F \Leftrightarrow^* \top$; recall that in these situations it is *unknown* whether $\neg F$ is valid or invalid). This is typical for augmentation schemes, and these schemes typically are not decision procedures themselves. However, it is not considered as a severe weakness, since these schemes are intended for use in undecidable theories; they try to extend the realm of a decision procedure. The above procedure slightly differs from the original one in the lemma-invoking mechanism: even if there is no lemma that involves a maximal term, the above procedure still goes on and does not declare failure (see Section 4.10).

Note that the orderings used in $\Leftrightarrow_{ccc}$ and in $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_j)}$ rules do not necessarily need to be the same.

Assuming that in the ordering $\prec$ used in $\Leftrightarrow_{ccc}$ that $t \prec x$ does not hold, where x is any variable and t is any compound term (for instance, the total ordering for ground terms can be based on size of terms and their alphabetical ordering), the rule $\Leftrightarrow_{ccc}$ (applied after $\Leftrightarrow_{absp^+(pra,\mathcal{L})}$) does not introduce alien terms (with respect to PRA).

EXAMPLE 14. *Let $\mathcal{L}$ is $\{p(x) \rightarrow f(x) \leq g(x),\ max(x,y) = x \rightarrow min(x,y) = y\}$, and let the signature of $\mathcal{E}$ is $(\{f,g,min,max\},\{p\})$. Let*

$$p(a)\wedge l \leq f(max(a,b))\wedge 0 < min(a,b)\wedge a \leq max(a,b)\wedge max(a,b) \leq a \rightarrow$$

$$l < g(a) + b$$

is a formula we want to prove [27]. We have to show that its negation is unsatisfiable. For simplicity, we omit applications of rules that are unsuccessful or don't have any effect:

$p(a)\wedge l \leq f(max(a,b))\wedge 0 < min(a,b)\wedge a \leq max(a,b)\wedge max(a,b) \leq a \wedge \neg(l < g(a) + b)$

$\Leftrightarrow_{absp^+(pra,\mathcal{L})}$
$\{p(a), l \leq v_0, f(max(a,b)) = v_0, 0 < v_1, min(a,b) = v_1, a \leq v_2, max(a,b) = v_2, v_2 \leq a, \neg(l < v_4 + b), g(a) = v_4\}$

$\Leftrightarrow_{simpl(pra)}$
$\{p(a), l \leq v_0, f(max(a,b)) = v_0, 1 \leq v_1, min(a,b) = v_1, a \leq v_2, max(a,b) = v_2, v_2 \leq a, \neg(((-1) \cdot v_4 + 1) + l \leq b), g(a) = v_4\}$

$\Leftrightarrow_{ccc}$
$\{p(a), l \leq v_0, f(v_2) = v_0, 1 \leq v_1, min(a,b) = v_1, a \leq v_2, max(a,b) = v_2, v_2 \leq a, \neg(((-1) \cdot v_4 + 1) + l \leq b), g(a) = v_4\}$

$\Leftrightarrow_{entail(Fourier,pra)}$ *(eliminated: l)*
$\{p(a), f(v_2) = v_0, 0 \leq v_1 + (-1), min(a,b) = v_1, 0 \leq a + (-1) \cdot v_2, max(a,b) = v_2, 0 \leq v_2 + (-1) \cdot a, g(a) = v_4, v_4 + b \leq v_0\}$

$\Leftrightarrow_{simpl(pra)}$
$\{p(a), f(v_2) = v_0, 1 \leq v_1, min(a,b) = v_1, v_2 \leq a, max(a,b) = v_2, a \leq v_2, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4\}$

$\Leftrightarrow_{entail^+(impl-eqs,pra)}$
$\{p(a), f(v_2) = v_0, 1 \leq v_1, min(a,b) = v_1, v_2 \leq a, max(a,b) = v_2, a \leq v_2, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4, v_2 = a\}$

$\Leftrightarrow_{ccc}$
$\{p(a), f(a) = v_0, 1 \leq v_1, min(a,b) = v_1, a \leq a, max(a,b) = a, a \leq a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4, v_2 = a\}$

$\Leftrightarrow_{entail(Fourier,pra)}$ *(eliminated: v_2)*
$\{p(a), f(a) = v_0, 0 \leq v1{+}(-1), min(a,b) = v_1, max(a,b) = a, g(a) = v_4, 0 \leq v_0 + (-1) \cdot v_4) + (-1) \cdot b, a = a\}$

$\Leftrightarrow_{simpl(pra)}$
$\{p(a), f(a) = v_0, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4)\}$

$\Rightarrow_{lemma^+(\mathcal{L},pra)}$ *(lemma: $p(x) \to f(x) \leq g(x)$)*
$\{p(a), v_0 = v_0, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), p(a) \to v_0 \leq g(a)\}$

case 1: $\{p(a), v_0 = v_0, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), \neg p(a)\}$

$\Leftrightarrow_{simpl(pra)}$

$\bot$

case 2: $\{p(a), v_0 = v_0, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), v_0 \leq g(a)\}$

$\Leftrightarrow_{absp^+(pra,\mathcal{L})}$

$\{p(a), v_0 = v_0, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), v_0 \leq v_5, g(a) = v_5\}$

$\Leftrightarrow_{simpl(pra)}$

$\{p(a), 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), v_0 \leq v_5, g(a) = v_5\}$

$\Leftrightarrow_{ccc}$

$\{p(a), 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq v_0 + (-1) \cdot v_4), v_0 \leq v_5, v_5 = v_4\}$

$\Leftrightarrow_{entail(Fourier,pra)}$ *(eliminated: v_0)*

$\{p(a), 0 \leq v_1 + (-1), min(a,b) = v_1, max(a,b) = a, g(a) = v_4, b \leq 0, 0 = v_4 + (-1) \cdot v_5\}$

$\Leftrightarrow_{simpl(pra)}$

$\{p(a), 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, b \leq 0\}$

$\Rightarrow_{lemma^+(\mathcal{L},pra)}$ *(eliminated: $p(a)$)*

$\{\top, 1 \leq v_1, min(a,b) = v_1, max(a,b) = a, b \leq 0\}$

$\Leftrightarrow_{simpl(pra)}$

$\{1 \leq v_1, min(a,b) = v_1, max(a,b) = a, b \leq 0\}$

$\Rightarrow_{lemma^+(\mathcal{L},pra)}$ *(lemma: $max(x,y) = x \rightarrow min(x,y) = y$)*

$\{1 \leq v_1, v_1 = v_1, max(a,b) = a, b \leq 0, max(a,b) = a \rightarrow v_1 = b\}$

case 2.1: $\{1 \leq v_1, v_1 = v_1, max(a,b) = a, b \leq 0, \neg max(a,b) = a\}$

$\Leftrightarrow_{simpl(pra)}$

$\bot$

case 2.2: $\{1 \leq v_1, v_1 = v_1, max(a,b) = a, b \leq 0, v_1 = b\}$

$\Leftrightarrow_{simpl(pra)}$

$\{1 \leq v_1, max(a,b) = a, b \leq 0, v_1 = b\}$

$\Leftrightarrow_{ccc}$

$\{1 \leq b, max(a,b) = a, b \leq 0, v_1 = b, \}$

$\Leftrightarrow_{unsat(pra)}$

$\bot$

$\Leftrightarrow_{split}$

$\bot$

$\Leftrightarrow_{split}$

$\bot$

THEOREM 5. *The above* TECTON*-style procedure is terminating and sound.*

Proof. We assume that in the ordering $\prec$ used in $\Leftrightarrow_{ccc}$ it does not hold $t \prec x$, where x is any variable and t is any compound term. In that case, the rule $\Leftrightarrow_{ccc}$ does not introduce new alien terms. Moreover, none of the rules from Steps 1 to 8 introduce new alien terms or introduce new variables (except the rule $\Leftrightarrow_{absp^+(pra,\mathcal{L})}$, when first applied within the block of steps from 1 to 8). Each of these rules always terminates. In addition, the rule $\Leftrightarrow_{entail(Fourier,pra)}$, when applicable, eliminates at least one variable, so it cannot be applied an infinite number of times. A similar argument holds for the rule $\Leftrightarrow_{entail^+(impl-eqs,pra)}$ so it cannot be applied an infinite number of times either. Therefore, the block of steps from 1 to 8 always terminates (either by returning the result of the procedure or by moving to Step 9). If the lemma rule is successfully applied, it eliminates the maximal alien term (w.r.t. PRA) according to the used reduction ordering $\prec$ (and all newly introduced alien terms are less then it w.r.t. $\prec$). Since $\prec$ is a well-founded relation, there does not exist an infinite chain of such formulae, and therefore the procedure terminates.

All macro inference rules used in the above TECTON-style procedure are sound, and so the procedure is sound. Therefore, if it returns $\perp$, then $\neg F$ is unsatisfiable, and the original formula F is valid. Additionally, if it returns $\top$ by $\Leftrightarrow^*$ (i.e., if the lemma rule is not used), then the original formula F is invalid (as PRA is consistent).

In addition, the above procedure is complete for the formulae belonging to PRA extended with the pure theory of equality.

Note that Presburger arithmetic in the above scheme can be replaced by any other decidable theory (given needed primitive procedures for detecting unsatisfiability, and the like), for instance, by strict multiplication arithmetic.

The above scheme corresponds to constraint contextual rewriting instantiated by combination of linear arithmetic and theory of ground equalities (denoted $CCR(DP_{lacc})$ in [1]).

5.4. Extended Proof Method

In [25] there is a description of the framework for the flexible augmentation of decision procedures. This framework can be used for different theories and for different decision procedures. It is based on quantifier elimination decision procedures, and new decision procedures can be simply “plugged in” to the system. Let $\mathcal{T}_1$ be a theory that admits a quantifier elimination, and let $\mathcal{T}$ be its conservative extension,[22] deter-

[22] We require that $\mathcal{T}$ be a conservative extension of $\mathcal{T}_1$. If $\mathcal{T}_1$ is complete, then this condition is trivially fulfilled.

mined by a set $\mathcal{L}$. Let A be an algorithm for quantifier elimination for $\mathcal{T}_1$. Let F be a formula of the background theory $\mathcal{T}$ to be proved. The extended proof method style scheme is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$.
2. Apply $\Leftrightarrow_{split}$.
3. Apply $\Leftrightarrow_{simpl(pra)}$.
4. Apply $\Leftrightarrow_{unsat(\mathcal{T}_1)}$; if not successful (i.e., if the rule $\Leftrightarrow_{unsat(\mathcal{T}_1)}$ fails to prove the unsatisfiability), then if a current formula belongs to the theory $\mathcal{T}_1$, return $\top$; otherwise go to next step.
5. Apply $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{L})}$, then $\Leftrightarrow_{entail(A,\mathcal{T}_1)}$, and then $\Leftrightarrow_{repl}$; if successful, go to Step 1.
6. Apply $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1)}$.
7. Go to Step 1.

In the rule $\Leftrightarrow_{repl}$, abstraction variables are replaced first. Note that the rule $\Leftrightarrow_{entail(A,\mathcal{T}_1)}$ may introduce disjunctions (for instance, for $A = Cooper$), so we (generally) need to jump to Step 1 (instead of to Step 3) after Step 5.

If the above procedure returns $\bot$, the original formula F is valid; if $\mathcal{T}_1$ is consistent, if the procedure returns $\top$ and Step 6 has not been applied, then the original formula F is invalid [25]. However, the procedure is not complete, and it can also return $\top$ if the original formula F is valid, failing to prove it. These situations arise when $\neg F \Rightarrow^* \top$, but not $\neg F \Leftrightarrow^* \top$, that is, in situations when the above procedure returns $\top$ and the lemma rule has not been applied. In these situations, the procedure can return *unknown* as a result.

The above procedure (or, rather, scheme) is different from the original version [25] in several points, including in the restrictions on lemma invoking. Additionally, in the original version of the procedure, equality reasoning is performed on the basis of using substitutivity axions as additional rewrite rules/lemmas. In the above scheme, the weak point is the reasoning about equalities (which can be easily improved within our general setting by adding the rule $\Leftrightarrow_{ccc}$). However, the weak reasoning about equalities makes the above procedure more efficient than, say, the one described in Section 5.3.3 on problems in which equality reasoning is not needed or not helpful. The above procedure also slightly differs from the original one as in the original procedure the maximal term is eliminated within the lemma invoking mechanism (which is introduced there) by a variable elimination procedure. The original procedure deals with quantifier-free formulae with arbitrary structure and does not transforms it into disjunctive normal form.

EXAMPLE 15. *Let $\mathcal{T}$ be* PRA *extended by $\mathcal{L} = \{minl(x) \leq maxl(x)\}$. Let*

$$l \leq minl(\alpha)) \wedge 0 < k \rightarrow \neg(maxl(\alpha) + k \leq l)$$

is a formula we want to prove [11, 25]. We have to show that its negation is unsatisfiable. In the instanced extended proof method for PRA *we use Hodes' algorithm as algorithm A. We omit applications of rules that are unsuccessful or don't have any effect:*

$\{l \leq minl(\alpha)), 0 < k, maxl(\alpha) + k \leq l\}$
$\Leftrightarrow_{simpl(pra)}$
$\{l \leq minl(\alpha), 1 \leq k, maxl(\alpha) + k \leq l\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{L})}$, $\Leftrightarrow_{entail(Hodes,pra)}$ *(eliminated: k)*, $\Leftrightarrow_{repl}$
$\{0 \leq minl(\alpha) + (-1) \cdot l, 1 \leq l + (-1) \cdot maxl(\alpha)\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{L})}$, $\Leftrightarrow_{entail(Hodes,pra)}$ *(eliminated: l)*, $\Leftrightarrow_{repl}$
$\{maxl(\alpha) + 1 \leq minl(\alpha)\}$
$\Rightarrow_{lemma^+(\mathcal{L},pra)}$ *(lemma: $minl(x) \leq maxl(x)$)*
$\{v_1 + 1 \leq minl(\alpha), minl(\alpha) \leq v_1, v_1 = v_1\}$
$\Leftrightarrow_{absp^+(pra,\mathcal{L})}$, $\Leftrightarrow_{entail(Hodes,pra)}$ *(eliminated: v_1)*, $\Leftrightarrow_{repl}$
$\{minl(\alpha) \leq minl(\alpha) + (-1)\}$
$\Rightarrow_{lemma^+(\mathcal{L},pra)}$ *(eliminated: $minl(\alpha)$)*
$\{v_0 \leq v_0 + (-1)\}$
$\Leftrightarrow_{unsat(pra)}$
$\perp$

THEOREM 6. *The above* EPM*-style procedure is terminating and sound.*

Proof. In the above procedure, if Step 5 is successfully applied, it leaves no abstraction variables introduced by $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{L})}$ (as abstraction variables are replaced first) and at least one other variable is eliminated; hence, the formula passed to Step 1 has fewer variables than one in the previous iteration. If Step 6 is successfully applied, it eliminates the maximal alien term (w.r.t. $\mathcal{T}_1$) according to the used reduction ordering $\prec$. Thus, in each iteration, the current formula is transformed into a formula (or, by an intermediate rules $\Leftrightarrow_{dnf}$ and $\Leftrightarrow_{split}$, into a set of formulae) either with the maximal term no greater (with respect to $\prec$) than in the original formula or with fewer variables. Since $\prec$ is a well-founded relation, there does not exist an infinite chain of such formulae, and therefore the procedure terminates.

All macro inference rules used in the above EPM-style procedure are sound, and so the procedure is sound. Additionally, if $\mathcal{T}_1$ is consistent, if the procedure returns $\top$ and if the rule $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1)}$ was not used, it means that $\neg F \Leftrightarrow^* \top$ (as all other rules are unsatisfiability preserving), that is, F is invalid.

In addition, the above procedure is complete for the formulae belonging to the theory $\mathcal{T}_1$. If the equality axioms are also used as lemmas, then the above procedure is a decision procedure for formulae belonging to theory $\mathcal{T}_1$ extended with the pure theory of equality.

6. Strict General Setting

Merged with each other, different schemes implemented within the proposed general setting have similar structures. Moreover, some slight changes can be made in order to have these structures clearly comparable in the sense that similar rules are in the same or similar positions. For this reason, we impose a fixed ordering on the macro inference rules. This ordering leads to a *strict general setting* (SGS) for building decision procedures into theorem provers.[23] A combination/augmentation scheme is, within the SGS framework, simply created by choosing and (de)activating certain rules at their fixed positions. The ordering on the rules is as given in Table I (we also include "Go to start" into the set of the rules). All schemes within the SGS framework are represented uniformly, by sequences of the macro inference rules used (i.e., by a corresponding sequence of 0s and 1s). In addition, some rules (entailment rules) have to be instantiated by specific underlying algorithms. For some schemes and some rules, if the rule fails, the scheme has to return $\top$; for some schemes and some rules, if the rule is successful, a control has to go to the start of the scheme. For instance, if we use the Nelson-Oppen-style entailment ($\Leftrightarrow_{entail^+(NO,\mathcal{T}_i)}$), if the rule has to return $\top$ when unsuccessful, and if the rule does not lead to a start of a scheme when successful, then we denote these additional parameters in the following way: $(NO;1;0)$. We omit these additional parameters if they have default values (the default values are $(/;0;0)$). For instance, the Shostak-style scheme is represented in the following way: $11101000100001(solve;1;0)0000101$. Table I gives representations of several combination/augmentation schemes: schemes made in the style of Nelson and Oppen, Shostak, TECTON (we discuss only one procedure from [27], one concerning PRA) and EPM procedures (note that these representations differs lightly from the schemes described in the preceding section). Note that it is sensible to use only one variant of

[23] This extension to the general setting has been presented as a short paper at IJCAR-2001 [24].

Table I. Descriptions of some combination/augmentation schemes within the SGS framework. Background theories are combinations of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ (extended by $\mathcal{L}$ for the TECTON style and the EPM style procedure).

#	Rule	N-O Style	Shostak Style	TECTON Style	EPM Style
1.	$\Leftrightarrow_{dnf}$	$\surd$	$\surd$	$\surd$	$\surd$
2.	$\Leftrightarrow_{split}$	$\surd$	$\surd$	$\surd$	$\surd$
3a.	$\Leftrightarrow_{simpl}$		$\surd$		
3b.	$\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$			$\surd$	$\surd$
4.	$\Leftrightarrow_{ccc}$		$\surd$	$\surd$	
5a.	$\Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$				
5b.	$\Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$				
5c.	$\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$	$\surd$			
5d.	$\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k,\mathcal{E})}$		$\surd$		
5e.	$\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k,\mathcal{L})}$				$\surd$
5f.	$\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k,\mathcal{L},\mathcal{E})}$			$\surd$	
6.	$\Leftrightarrow_{unsat(\mathcal{T}_i)}$ $(i=1,\ldots,k)$	$\surd$		$\surd$	$\surd$
7.	$\Leftrightarrow_{superpose(\mathcal{R})}$				
8a.	$\Leftrightarrow_{entail(A,\mathcal{T}_i)}$ $(i=1,\ldots,k)$		$\surd$ $(solve;1;0)$	$\surd$ $(Fourier;0;1)$	$\surd$ $(A;0;0)$
8b.	$\Rightarrow_{entail(A,\mathcal{T}_i)}$ $(i=1,\ldots,k)$				
8c.	$\Leftrightarrow_{entail^+(A,\mathcal{T}_i)}$ $(i=1,\ldots,k)$	$\surd$ $(NO;1;0)$		$\surd$ $(impl\text{–}eqs;0;1)$	
8d.	$\Rightarrow_{entail^-(A,\mathcal{T}_i)}$ $(i=1,\ldots,k)$				
9a.	$\Leftrightarrow_{repl^=}$	$\surd$			
9b.	$\Leftrightarrow_{repl}$		$\surd$		$\surd$
10.	$\Leftrightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$			$\surd$	$\surd$
11.	Go to start	$\surd$	$\surd$	$\surd$	$\surd$

the abstraction rule, only one variant of the simplification and only one variant of the replacement in one scheme. However, it may be sensible to use more variants of the entailment rule (as in TECTON).

A combination/augmentation scheme is additionally specified by a set $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ of underlying theories. In the Nelson-Oppen-style scheme, the background theory is a combination of the underlying theories. In the Shostak-style schemes, the background theory is the combination of the underlying theories and the pure theory of equality ($\mathcal{E}$). In the TECTON style scheme, $k = 1$, and the background theory is a conservative extension (given by $\mathcal{L}$) of the combination of $\mathcal{T}_1$ (that is, PRA) and $\mathcal{E}$. In the EPM-style scheme, $k = 1$, and the background theory is a conservative extension (given by $\mathcal{L}$) of $\mathcal{T}_1$, where $\mathcal{T}_1$ is a theory that admits quantifier elimination (e.g., PRA, PIA) and A is a quantifier elimination procedure for $\mathcal{T}_1$ (e.g., Hodes' procedure for

PRA). If in the EPM-style scheme a procedure that does not preserve unsatisfiability is used (e.g., Hodes' procedure for PIA), then we choose the rule $\Rightarrow_{entail(A,\mathcal{T}_i)}$ (8b) instead of $\Leftrightarrow_{entail(A,\mathcal{T}_i)}$ (8a).

Instead of a number of implemented combination/augmentation schemes, within the SGS framework there is only one metascheme. It is parameterized by a specification for a scheme and by a set of underlying theories. There are no scheme-specific parts built-in or invoked from the metascheme. It is easy to create or modify one scheme and to combine ideas from two schemes.

The set of theories can be determined for each conjecture by a signatures library mechanism. This mechanism, for instance, could try to find a (small) set of available theories such that a conjecture belongs to their combination; if for each of them there is the Nelson-Oppen-style entailment available, then the Nelson-Oppen-style scheme can be applied. Similarly, this mechanism could try to find if a conjecture belongs to an extension of some theory that admits quantifier elimination, while there is a corresponding quantifier elimination procedure available, and the extension is given by some additional rules and lemmas; if so, the EPM-style procedure can be applied. The ordering of schemes can be adjusted according to different criteria (for certain theories, the Nelson/Oppen style scheme is complete, but, for instance, the TECTON style scheme is often more efficient).

7. Prototype Implementation and Results

We have implemented the macro inference rules described in the preceding sections. We implemented the $\Leftrightarrow_{ccc}$ rule on the basis of Nelson-Oppen congruence closure algorithm[24] [38] and in the way described in Section 4.8. We implemented the required instances of inference rules for the pure theory of equality, the theory of lists with *car*, *cdr*, *cons*, and theories PRA, PIA, PNA. Checking unsatisfiability for the pure theory of equality is based on the constant congruence algorithm. Checking unsatisfiability for the theory of lists is based on the rewrite system described in Example 8. Checking unsatisfiability for PRA, PIA, and PNA is based on Hodes' and Cooper's procedures. Nelson and Oppen's entailment is implemented on the basis of the connection: $f \Leftrightarrow_{entail^+(NO,\mathcal{T}_i)} f \cup \{x = y\}$ if and only if $f \cup \{\neg(x = y)\} \Leftrightarrow_{unsat(\mathcal{T}_i)} \perp$. There is a implementation of a *solver* for equational

[24] The first, preliminary tests show that a constant congruence algorithm does not take much of the overall CPU time taken by specific schemes, so we preliminarily use Nelson and Oppen's algorithm, which is simpler than the (more efficient) one due to Shostak.

real linear arithmetic. For the elimination of variables, we have implemented Hodes' procedure for PRA (and we use it for PIA and PNA as sound, but incomplete procedure) and Cooper's procedure for PIA (and we also adapted in for PNA); thus, we have implemented the following rules: $\Leftrightarrow_{entail(Hodes,pra)}$, $\Rightarrow_{entail(Hodes,pia)}$, $\Rightarrow_{entail(Hodes,pna)}$, $\Leftrightarrow_{entail(Cooper,pia)}$, $\Leftrightarrow_{entail(Cooper,pna)}$. For simplicity, we use $\Rightarrow_{entail^+(NO,\mathcal{T}_i)}$ instead of $\Rightarrow_{entail^+(impl-eqs,pra)}$ and we use $\Leftrightarrow_{entail(Hodes,pra)}$ instead of $\Leftrightarrow_{entail(Fourier,pra)}$.[25]

We have implemented the proposed general setting in SWI Prolog and within the *Clam* proof-planning system [15] (we used its built-in recursive path ordering with status). For the sake of efficiency, we have implemented all macro inference rules as new predicates in the method language (instead of as methods). Within this setting, it was easy to implement the schemes for combining and augmenting decision procedures discussed in Section 5. Some results (obtained on examples from the literature) are given in Table II and Table III (by "Scheme: TECTON" we mean the scheme described in Section 5.3.3).[26] Table II shows experimental results on examples worked in Section 5. We have also implemented the SGS system and the experimental results were virtually the same as results given in Table III.

Table III presents some experimental comparisons between different schemes implemented within our setting. Although the Shostak-style procedure is not implemented by the special data structures and optimized as the original procedure was, our experimental results support previous results on comparison between Nelson and Oppen's and Shostak's schemes; namely, it is observed that Shostak's scheme is generally more efficient than the one due to Nelson and Oppen (see also [19]). Also, it is typically more efficient than the TECTON-style procedure (when both of them are applicable). Example 6 illustrates the incompleteness of the original Shostak's procedure and its variant implemented within the general setting. Example 7 shows that the Shostak-style procedure terminates for the conjecture for which the original procedure loops. In Examples 8, 9, 10, and 11, most of CPU time spent by the TECTON and EPM-style procedures was spent by the reduction ordering mechanism. The weak point of the EPM-style scheme is equality reasoning (similarly to Boyer and Moore's linear arithmetic procedure), and it is shown by the obtained results. Without the substitutivity axiom used as an additional rewrite rule, it can han-

[25] This also demonstrates that the procedures implemented within the presented general setting can have the scope (although not always their efficiency) of several different schemes only on the basis of very few underlying procedures.

[26] Tests were made on a 433 MHz 64 MB, PC running under Linux. CPU time is given in seconds. Programs are available upon request to the first author.

Table II. Results of the schemes for combining/augmenting decision procedures implemented within the general setting.

1	Scheme:	Nelson/Oppen's
	Conjecture:	$x \leq y \wedge y \leq x + car(cons(0, x)) \wedge p(h(x) - h(y)) \wedge \neg p(0)$
	Theories used:	PRA, lists, equality with uninterpreted functions
	CPU time:	0.58s
2	Scheme:	Shostak's
	Conjecture:	$z = f(x - y) \wedge x = y + z \wedge \neg(y + f(f(z)) = x)$
	Theories used:	equational real linear arithmetic
	CPU time:	0.17s
3	Scheme:	TECTON
	Conjecture:	$p(a) \wedge l \leq f(max(a, b)) \wedge 0 < min(a, b) \wedge a \leq max(a, b) \wedge$ $max(a, b) \leq a \wedge \neg(l < g(a) + b)$
	Lemma used:	$p(x) \rightarrow f(x) \leq g(x)$
	Lemma used:	$max(x, y) = x \rightarrow min(x, y) = y$
	CPU time:	2.39s
4	Scheme:	Extended proof method
	Conjecture:	$l \leq minl(\alpha) \wedge 0 < k \wedge maxl(\alpha) + k \leq l$
	Theory used:	PRA
	Procedure A:	Hodes'
	Lemma used:	$minl(\xi) \leq maxl(\xi)$
	CPU time:	0.14s

dle only very simple conjectures with uninterpreted function symbols that need equality reasoning (Example 7). In addition, this procedure failed on Example 8 which needs use of lemmas but also needs stronger equality reasoning. On the other hand, this procedure was more efficient on examples in which equality reasoning is not needed or not very helpful (Examples 9, 10, 11). Table III also illustrates the fact that for each scheme there are some conjectures for which that scheme is most successful. In other words, the table illustrates the fact that it is sensible to have many combination/augmentation schemes available in one theorem prover and to choose between them on the basis of some heuristic scores that can be dynamically adjusted according to theorems already successfully proved (see, for instance, [34]).

All given results serve just as a rough illustration of efficiency of the combination/augmentation schemes implemented within our general setting, while further tests on larger corpora are needed.

Table III. Comparison among the schemes for combining/augmenting decision procedures implemented within the general setting (*n/i* means "not implemented"; *?* means that the scheme terminated on the conjecture but failed to prove or disprove it; CPU time is given in seconds).

#	Conjecture	N-O	Shostak	TECTON	EPM
1.	$x \leq y, y \leq x + car(cons(0,x)), p(h(x) - h(y)), \neg p(0)$ [37]	0.58	n/i	?	?
2.	$z = f(x-y), x = y + z, \neg(y + f(f(z)) = x)$ [43]	0.18	0.17	0.40	?
3.	$x = y, f(f(f(x))) = f(x), \neg(f(f(f(f(f(y))))) = f(x))$ [19]	0.01	0.01	0.03	?
4.	$y = f(z), x - 2 \cdot y = 0, \neg(f(x-y) = f(f(z)))$ [19]	0.16	0.10	0.14	?
5.	$f(x) = 4, f(2 \cdot y - x) = 3, x = f(2 \cdot x - y),$ $4 \cdot x = 2 \cdot x + 2 \cdot y$ [19]	0.62	0.17	0.48	?
6.	$f(a-1) - 1 = a + 1, f(b) + 1 = b - 1, b + 1 = a$ [42]	0.85	?	0.76	?
7.	$f(a) = a, f(b) = b - 1, a = b$ [42]	0.03	0.02	0.06	0.03
8.	$p(a), l \leq f(max(a,b)), 0 < min(a,b), a \leq max(a,b)$ $max(a,b) \leq a, \neg(l < g(a) + b)$ [27] lemma: $p(x) \rightarrow f(x) \leq g(x)$ lemma: $max(x,y) = x \rightarrow min(x,y) = y$	?	?	2.39	?
9.	$l \leq minl(\alpha), 0 < k, maxl(\alpha) + k \leq l$ [11] lemma: $minl(\xi) \leq maxl(\xi)$	?	?	0.26	0.14
10.	$lp + lt \leq maxint, i \leq lt, \neg(i + \delta(pat, lp, c) \leq maxint)$ [11] lemma: $\delta(x,y,z) \leq y$	?	?	0.34	0.23
11.	$(ms(c) + (ms(a) \cdot ms(a)) + ms(b) \cdot ms(b) <$ $((ms(c) + (ms(b) \cdot ms(b))) + (2 \cdot (ms(a) \cdot (ms(a) \cdot ms(b))))$ $+(ms(a) \cdot (ms(a) \cdot (ms(a) \cdot ms(a))))))$ [11] lemma: $0 < i \rightarrow j < i \cdot j$ lemma: $0 < ms(x)$	?	?	6.71	5.73

We have also made a prototype implementation of the proposed general setting within the LambdaClam proof-planning system that is being developed at the Mathematical Reasoning Group, Division of Informatics, University of Edinburgh. LambdaClam is a higher-order version of the *Clam* system, and it is being implemented in Teyjus LambdaProlog. We have implemented all macro inference rules and combination/augmentation scheme as methods but in the backward reasoning manner (and not in the proof by refutation manner, as presented in this paper). In this implementation, we use underlying theory-specific procedures implemented in Prolog as external procedures. This implementation is less efficient than the implementation within the *Clam* system and is still not fully functional (for instance, reduction ordering is still not available in LambdaClam).

8. Related Work

The long line of research concerning combining and augmenting decision procedures is related to the setting described in this paper. Several systems described in the literature and used in practice were the basis for introducing the given macro inference rules and/or for the case study [37, 43, 11, 27, 1]. All mentioned systems introduce specific, new approaches to the problem and do not give a general, uniform view to different approaches.

By the setting given in this paper we further develop the idea of the flexible integration of decision procedures described in [25]. This work is in spirit also closely related to the work of Armando and Ranise [1] in the sense that it is based on the idea of describing a general framework for using decision procedures that can be instantiated by specific theory and specific underlying procedures for that theory. For instance, constraint contextual rewriting can be instantiated to Boyer-Moore's integration procedure and to the procedure used in TECTON. Soundness and termination for constraint contextual rewriting are proved formally for the abstract scheme when certain requirements on the rewriting mechanism and the decision procedure are satisfied [3]. On the other hand, our setting is in spirit related to the work of Barrett, Dill, and Stump [8], which gives a flexible framework for cooperating decision procedures.[27] In this framework (built on few abstract primitive methods), the Nelson/Oppen and the Shostak-style procedures can be implemented, and their correctness (termination, soundness, and completeness) can be formally proved. The general setting proposed in this paper is more general than approaches proposed in [1, 25, 8] as it addresses both the problem of combining decision procedures and the problem of augmenting decision procedures. The relationship between combination/augmentation schemes discussed in this paper is illustrated in Figure 1 (contextual rewriting is not a combination/augmentation scheme but is closely related to them).

Within our general setting we proposed one new lemma-invoking mechanism and some slight variations. It enables extensions (discussed in Section 4.10) that enlarge the scope of similar methods [11, 27, 29, 1] by providing both the possibility of keeping the proving process going without lemmas and the possibility of using several lemmas at the same point.

[27] The papers [25] and [8], with analogous names (presented at two subsequent CADEs) and similar in spirit, are parallel in two research lines – in cooperating and augmenting decision procedures.

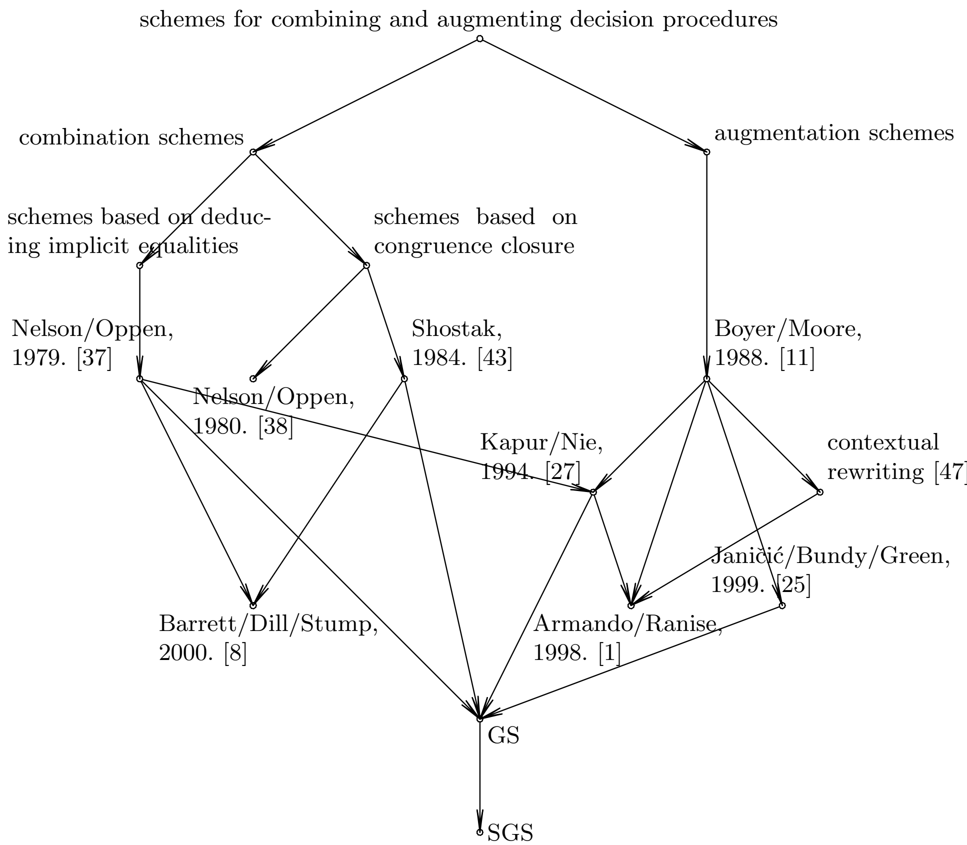

Figure 1. Schemes for combining and augmenting decision procedures and their relationship.

9. Future Work

In future work we are planning to further investigate different approaches (and their variants) for combining and augmenting decision procedures and their properties. We will investigate possibilities for generic proving of termination of different schemes implemented within the (S)GS framework. Such an extension of the framework would be based on some further properties imposed on all rules; it would serve as a basis for some well-founded ordering of formulae and, we hope, lead to a generic termination proof for a class of schemes. Proving completeness for several schemes at the same time also seems to be a very interesting and a very difficult problem. We will try to investigate whether each GS scheme can be represented as an SGS scheme and whether each GS proof can be normalized into some SGS proof (that is, whether the SGS framework has the same expressiveness as the GS framework).

We will look at possibilities for fully formally describing the proposed macro inference rules and for precise relationships between them and some calculi (such as the calculi discussed in [48, 4, 5]) or some other specific inference rules (such as discussed in [49, 10, 47]). We will try to modularize the entail rule, the rule that has an essential role in all combination/augmentation schemes.

We will investigate possibilities for making (S)GS schemes that combine scopes of some other schemes; that is, we will investigate possibilities for combining the scopes of the schemes described in this paper. There might be difficulties in achieving this as those schemes are defined for different classes of theories. We will also investigate possibilities for automatic generation of different instantiations of the specific macro inference rules. We will try to make a mechanism that for a new theory, recognizes relevant available rewrite rules and uses them in implementing specific macro inference rules for that theory, such as simplification and quantifier elimination. That approach follows the ideas from [14].

We will try to extend the proposed setting to many-sorted logic and over the quantifier-free fragment. Also, we will try to refine the lemma-invoking mechanism in situations in which there is no reduction ordering available or additional rules cannot be oriented.

We are planning to investigate and use more efficient versions of specific macro inference rules (such as congruence closure procedure or procedures for Presburger arithmetic). More efficient versions of macro inference rule would lead to much improved performances of schemes implemented within the proposed general setting.

We are planning to use and further improve the prototype implementation. We believe that the proof-planning paradigm [13, 12] is a convenient environment for such a flexible implementation and each macro inference rule can correspond to a method (while in the preliminary implementations in the *Clam* system the macro inference rules are represented as new predicates in the method language). The macro inference rules implemented as methods would serve as a basis for building different supermethods for combining and augmenting decision procedures. Theories would be represented by their signatures, by their properties (e.g., convex, σ-theory, admits quantifier elimination), and by available primitive procedures (e.g., checking unsatisfiability, solve, variable elimination); applicability of some supermethod (some combination/augmentation scheme) would be interpreted in terms of available theories and would be handled by a supersupermethod responsible for the use of decision procedures. Grammar library mechanisms would check the applicability of each scheme by checking the available theories and would determine the set of combined theories (as a parameter for the scheme). If more than one scheme or supermethod is applicable, they would be ordered by some heuristic scores that would be dynamically adjusted according to theorems already proved.

10. Conclusions

In this paper we presented a general setting for describing and implementing different schemes for both combining and augmenting decision procedures. This setting is based on the macro inference rules used in approaches known from the literature. Some of them are abstraction, entailment, congruence closure, and lemma invoking. Some of the rules are generic (for instance, $\Leftrightarrow_{ccc}$ and $\Leftrightarrow_{repl}$), while some are theory-specific. Some of the theory-specific rules use only information about signatures of combined theories (like the abstraction rules), while some rules use deeper theory-specific knowledge (like $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$, $\Leftrightarrow_{unsat(\mathcal{T}_i)}$, and the entailment rules). Only the rule $\Leftrightarrow_{unsat(\mathcal{T}_i)}$ uses decision procedures as black boxes, while $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ and the entailment rules use some functionalities of decision procedures, such as simplification of terms, deducing implicit equalities, and elimination of variables.

Using the proposed setting, we described several procedures for both combining and augmenting decision procedures, including Shostak's, Nelson and Oppen's, and the approach used in the TECTON system. These descriptions do not fully reflect all aspects of these approaches but have their scopes (although not always their efficiency). The general setting gives key ideas of one combination/augmentation scheme and makes different schemes comparable. The strict general setting (SGS) provides a way of implementing and using a number of schemes in a uniform way. All macro inference rules are sound, and, therefore, all schemes implemented within our general setting are sound. All macro inference rules are terminating, but termination of different combination/augmentation schemes has to be proved separately. For some of these schemes it can be proved they are complete (e.g., for the Nelson-Oppen-style scheme).

Concerning efficiency, procedures obtained in this way (in this setting) would be less efficient than some tightly integrated procedures, but we believe that gains from the flexibility would overcome the potential losses in efficiency. In some schemes, ground completion is performed in each iteration, and there is no store for rewriting rules induced by current equalities; on the one side this makes these schemes flexible, but on the other side it decreases their efficiency. However, ground completion can be performed in $O(n^2)$ or in $O(n \log n)$ time [26, 9], and this computational time is dominated by, say, variable elimination or similar steps (which may be of the exponential or of the superexponential complexity). The situation is similar for other rules that are performed less restrictively than in the original schemes, and we believe that these losses in efficiency in the schemes implemented within the proposed setting are not significant. We believe that the proposed setting provides a good balance between flexibility and efficiency.

The key features of the proposed approach are the following:

– Proofs represented in terms of applications of introduced macro inference rules are relatively simple and easy to understand.

– Precise descriptions of only a few macro inference rules leads to precise descriptions of a number of combination/augmentation schemes. This enables formal reasoning about different combination/augmentation schemes.

– Within the described setting, different combination/augmentation schemes share a lot of code (which makes them easier to implement); at the same time the whole of the setting is built fully flexible, from independent modules.

– It provides the possibility of having available different schemes both for combination and augmentation of decision procedures in a uniform way. If more than one scheme is applicable, they could be ordered by some heuristic scores that can be dynamically adjusted according to theorems already successfully proved.

– The setting yields the ability to make experimental comparisons between different combination/augmentation schemes and the results would not be contaminated by differences due to programming languages, operating systems, machines, programming skills, and the like.

– A flexible structure of the schemes implemented within the proposed setting makes possible and easy communication with other components and strategies of the prover (e.g., induction); the schemes implemented within the setting don't use or maintain any special data structures, data bases, constraint stores, and the like, and the only external device used is a reduction ordering on terms; after each step of these schemes, a current (sub)goal itself can be passed to some other proving strategy.

– Ideas from different schemes for combining and augmenting decision procedures can be easily combined (for instance, Nelson and Oppen's algorithm can be easily augmented by the lemma-invoking mechanism, and the extended proof method can be easily augmented by the procedure for congruence closure).

– Any scheme implemented within the setting can be based on different underlying procedures for a specific theory (e.g., it can choose between Hodes' and Cooper's procedure for Presburger arithmetic).

– Any module that corresponds to an inference rule can be improved or completely changed only if it meets its specification (for instance, congruence closure can be based on Shostak's instead on Nelson and Oppen's algorithm); this can be done without any consequences to other components of the setting or for some of its instances. This makes the system fully flexible and, at the same time, provides an opportunity for improving its efficiency incrementally.

We have made a prototype implementation of the (S)GS system in the *Clam* and *LambdaClam* systems and successfully proved a number of conjectures. We are planning to further improve and extend these implementations.

Acknowledgments

We are grateful to Ian Green, Cesare Tinelli, Silvio Ranise, Žarko Mijajlović, and Alessandro Armando for inspiring discussions and for a number of valuable comments on the draft version of this paper.

References

1. Armando, A. and S. Ranise: 1998, 'Constraint Contextual Rewriting'. In: *Proceedings of the International Workshop on First order Theorem Proving (FTP'98)*. Vienna, Austria, pp. 65–75.
2. Armando, A. and S. Ranise: 2000a, 'A Practical Extension Mechanism for Decision Procedures'. In: *Proceedings of Formal Methods Tools 2000 (FMT'2000)*.
3. Armando, A. and S. Ranise: 2000b, 'Termination of Constraint Contextual Rewriting'. In: *Proceedings of 3rd International Workshop on Frontiers of Combining Systems (FroCoS'2000)*. Nancy, France, pp. 47–61.
4. Bachmair, L. and H. Ganzinger: 1990, 'On Restrictions of Ordered Paramodulation with Simplification'. In: *Proceedings of the 10th Conference on Automated Deduction*. pp. 427—441.

5. Bachmair, L. and H. Ganzinger: 1998, 'Strict Basic Superposition'. In: C. Kirchner and H. Kirchner (eds.): *Proceedings of the 15th Conference on Automated Deduction*.
6. Bachmair, L. and A. Tiwari: 2000, 'Abstract Congruence Closure and Specializations'. In: D. A. MacAllester (ed.): *Proceedings of the 17th Conference on Automated Deduction (CADE-17)*.
7. Barrett, C., D. Dill, and J. Levitt: 1996, 'Validity Checking for Combinations of Theories with Equality'. In: *International Conference on Formal Methods in Computer-Aided Design*. pp. 187–201.
8. Barrett, C. W., D. L. Dill, and A. Stump: 2000, 'A Framework for Cooperating Decision Procedures'. In: D. A. MacAllester (ed.): *Proceedings of the 17th Conference on Automated Deduction (CADE–17)*.
9. Bjørner, N. S.: 1998, 'Integrating decision procedures for temporal verification'. Ph.D. thesis, Stanford University.
10. Bouhoula, A. and M. Rusinowitch: 1993, 'Automated Case Analysis in Proof by Induction'. In: R. Bajcsy (ed.): *Proc. 13th Intern. Joint Conference on Artificial Intelligence (IJCAI '93)*. San Mateo, CA.
11. Boyer, R. S. and J. S. Moore: 1988, 'Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic'. In: J. E. Hayes, J. Richards, and D. Michie (eds.): *Machine Intelligence 11*. pp. 83–124.
12. Bundy, A.: 1988, 'The Use of Explicit Plans to Guide Inductive Proofs'. In: R. Lusk and R. Overbeek (eds.): *9th Conference on Automated Deduction*. pp. 111–120. Longer version available from Edinburgh as DAI Research Paper No. 349.
13. Bundy, A.: 1991a, 'A Science of Reasoning'. In: J.-L. Lassez and G. Plotkin (eds.): *Computational Logic: Essays in Honor of Alan Robinson*. pp. 178–198. Also available from Edinburgh as DAI Research Paper 445.
14. Bundy, A.: 1991b, 'The Use of Proof Plans for Normalization'. In: R. S. Boyer (ed.): *Essays in Honor of Woody Bledsoe*. pp. 149–166. Also available from Edinburgh as DAI Research Paper No. 513.
15. Bundy, A., F. van Harmelen, C. Horn, and A. Smaill: 1990, 'The Oyster-Clam system'. In: M. E. Stickel (ed.): *Proceedings of the 10th Conference on Automated Deduction*. pp. 647–648. Also available from Edinburgh as DAI Research Paper 507.
16. Busatto, R.: 1995, 'The use of proof planning in normalisation'. Ph.D. thesis, University of Edinburgh.
17. Cooper, D. C.: 1972, 'Theorem Proving in Arithmetic Without Multiplication'. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence 7*. pp. 91–99.
18. Craigen, D., S. Kromodimoeljo, I. Meisels, B. Pase, and M. Saaltink: 1991, 'EVES: An Overview'. In: *Proceedings of Formal Software Development Methods (VDM '91)*. pp. 389–405.
19. Cyrluk, D., P. Lincoln, and N. Shankar: 1996, 'On Shostak's Decision Procedure for Combinations of Theories'. In: M. A. McRobbie and J. K. Slaney (eds.): *Proceedings of the 13th Conference on Automated Deduction*.
20. Dershowitz, N.: 1982, 'Ordering for term–rewriting systems'. *Theoretical Computer Science* **17**(3), 279–301.
21. Detlefs, D.: 1996, 'An Overview of the Extended Static Checking System'. In: *Proceedings of the First Workshop on Formal Methods in Software Practice*. pp. 1–9.

22. Ehdm: 1993, 'User Guide for the EHDM Specification Language and Verification System, Version 6.1.'. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA.
23. Hodes, L.: 1971, 'Solving Problems by Formula Manipulation in Logic and Linear Inequalities'. In: *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*. Imperial College, London, England.
24. Janičić, P. and A. Bundy: 2001, 'Strict General Setting for Building Decision Procedures into Theorem Provers'. In: R. Goré, A. Leitsch, and T. Nipkow (eds.): *The 1st International Joint Conference on Automated Reasoning (IJCAR-2001) — Short Papers*. pp. 86–95.
25. Janičić, P., A. Bundy, and I. Green: 1999, 'A Framework for the Flexible Integration of a Class of Decision Procedures into Theorem Provers'. In: H. Ganzinger (ed.): *Proceedings of the 16th Conference on Automated Deduction (CADE-16)*. pp. 127–141.
26. Kapur, D.: 1997, 'Shostak's Congruence Closure as Completion'. In: *International Conference on Rewriting Techniques and Applications, RTA '97*. Barcelona, Spain.
27. Kapur, D. and X. Nie: 1994, 'Reasoning about Numbers in Tecton'. In: *Proceedings of 8th International Symposium on Methodologies for Intelligent Systems, (ISMIS'94)*. Charlotte, NC, pp. 57–70.
28. Kapur, D. and M. Subramaniam: 1996, 'Lemma Discovery in Automating Induction'. In: M. A. McRobbie and J. K. Slaney (eds.): *13th International Conference on Automated Deduction (CADE-13)*. pp. 538–552.
29. Kapur, D. and M. Subramaniam: 2000, 'Using an induction prover for verifying arithmetic circuits'. *Software Tools for Technology Transfer* **3**(1), 32–65.
30. Kreisel, G. and J. L. Krivine: 1967, *Elements of Mathematical Logic: Model Theory*. Amsterdam: North Holland.
31. Lassez, J.-L. and M. Maher: 1992, 'On Fourier's algorithm for linear arithmetic constraints'. *Journal of Automated Reasoning* **9**, 373–379.
32. Luckham, D. C., S. M. German, F. W. Von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis: 1979, 'Stanford Pascal Verifier user manual'. Technical report. CSD Report STAN-CS-79-731, Stanford University, Stanford, CA.
33. Manna, Z.: 1994, '*STeP: The Stanford Temporal Prover*'. Technical report. STAN-CS-TR-94, Computer Science Department, Stanford University, Stanford, CA.
34. Manning, A., A. Ireland, and A. Bundy: 1993, 'Increasing the Versatility of Heuristic Based Theorem Provers'. In: A. Voronkov (ed.): *International Conference on Logic Programming and Automated Reasoning – LPAR 93, St. Petersburg*. pp. 194–204.
35. Mendelson, E.: 1964, *Introduction to Mathematical Logic*. Van Nostrand Reinhold Co.
36. Mostowski, A.: 1952, 'On direct products of theories'. *Journal of Simbolic Logic* **17**, 1–31.
37. Nelson, G. and D. C. Oppen: 1979, 'Simplification by cooperating decision procedures'. *ACM Transactions on Programming Languages and Systems* **1**(2), 245–257.
38. Nelson, G. and D. C. Oppen: 1980, 'Fast decision procedures based on congruence closure'. *Journal of the ACM* **27**(2), 356–364. Also: Stanford CS Report STAN-CS-77-646, 1977.

39. Oppen, D. C.: 1978, 'A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic'. *Journal of Computer and System Sciences* **16**(3), 323–332.
40. Owre, S., S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas: 1996, 'PVS: Combining Specification, Proof Checking, and Model Checking'. In: R. Alur and T. A. Henzinger (eds.): *Proceedings of the 1996 Conference on Computer-Aided Verification*. New Brunswick, NJ, pp. 411–414.
41. Presburger, M.: 1930, '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*. pp. 92–101, 395. Annotated English version also available [45].
42. Rueß, H. and N. Shankar: 2001, 'Deconstructing Shostak'. In: *Proceedings of the Conference on Logic in Computer Science (LICS)*.
43. Shostak, R. E.: 1984, 'Deciding combinations of theories'. *Journal of the ACM* **31**(1), 1–12. Also: *Proceedings of the 6th International Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science*, pp. 209–222. Springer-Verlag, June 1982.
44. Skolem, T.: 1970, 'Über einige Satzfunktionen in der Arithmetik'. In: J. E. Fenstad (ed.): *Selected Works in Logic (by Th. Skolem)*. Universitets–forlaget, Oslo.
45. Stansifer, R.: 1984, 'Presburger's Article on Integer Arithmetic: Remarks and Translation'. Technical Report TR 84-639, Department of Computer Science, Cornell University.
46. Tinelli, C. and M. Harandi: 1996, 'A New Correctness Proof of the Nelson–Oppen Combination Procedure'. In: F. Baader and K. U. Schultz (eds.): *Frontiers of Combining Systems: Proceeding of the 1st International Workshop*. pp. 103–120.
47. Zhang, H.: 1995, 'Contextual rewriting in automated reasoning'. *Fundamenta Informaticae 24* **24**, 107–123.
48. Zhang, H. and D. Kapur: 1985, 'First-Order Theorem Proving Using Conditional Rewrite Rules'. In: E. Lusk and R. Overbeek (eds.): *Proceedings of 9th Conference on Automated Deduction*. pp. 1–20.
49. Zhang, H. and J. L. Rémy: 1985, 'Contextual Rewriting'. In: J. P. Jouannaud (ed.): *Proceedings of 1st International Conference on Rewriting Techniques and Applications*. pp. 1–20.