On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal*

Alessandro Armando[1], Maria Paola Bonacina[2], Silvio Ranise[3], and Stephan Schulz[2]

[1] DIST, Università degli Studi di Genova
Viale Causa 13, I-16145 Genova, Italy
armando@dist.unige.it

[2] Dipartimento di Informatica, Università degli Studi di Verona
Strada Le Grazie 15, I-37134 Verona, Italy
mariapaola.bonacina@univr.it schulz@eprover.org

[3] LORIA & INRIA-Lorraine
615 Rue du Jardin Botanique, F-54600 Villers-lès-Nancy, France
silvio.ranise@loria.fr

Abstract. The rewriting approach to $\mathcal{T}$-satisfiability is based on establishing termination of a rewrite-based inference system for first-order logic on the $\mathcal{T}$-satisfiability problem. Extending previous such results, including the *quantifier-free theory of equality* and the *theory of arrays with or without extensionality*, we prove termination for the theories of *records with or without extensionality*, *integer offsets* and *integer offsets modulo*. A general theorem for termination on *combinations of theories*, that covers any combination of the theories above, is given next. For empirical evaluation, the rewrite-based theorem prover E is compared with the validity checkers CVC and CVC Lite, on both synthetic and real-world benchmarks, including both valid and invalid instances. Parametric synthetic benchmarks test *scalability*, while real-world benchmarks test ability to handle huge sets of literals. Contrary to the folklore that a general-purpose prover cannot compete with specialized reasoners, the experiments are overall favorable to the theorem prover, showing that the rewriting approach is both elegant and practical.

1 Introduction

Many state-of-the-art verification tools (e.g., [21, 11, 9, 6]) incorporate satisfiability procedures for theories of data types. However, most verification problems involve more than one theory, so that one needs procedures for *combination of theories* (e.g., [14, 20]). Combination is complicated: for instance, understanding, formalizing and proving correct the method in [20] required much work

* Research supported in part by MIUR grant no. 2003-097383.

(e.g., [16, 7, 12]). Combining theories by combining algorithms may lead to *ad hoc* procedures, that are hard to modify, extend, integrate into, or even interface with other systems. Satisfiability procedures need to be proved correct and complete, by showing that whenever they report "satisfiable," the ouput represents a model. Model-construction arguments for concrete procedures are specialized for those, so that each new procedure requires a new proof (e.g., [16, 22]), while abstract frameworks often focus on combining the theory of equality with at most one other theory (e.g., [12]). Data structures and algorithms for each new procedure are usually implemented from scratch, with little software reuse and high risk of errors.

If one could use *first-order theorem-proving strategies*, combination would become much simpler, because in several cases it would be sufficient to give as input the union of the presentations of the theories. No *ad hoc* correctness and completeness proofs would be needed, because a sound and complete theorem-proving strategy is a *semi-decision procedure* for unsatisfiability. Existing first-order provers could represent a repository of code available for reuse.

The crux is *termination*: to have a *decision procedure*, one needs to prove that a complete theorem-proving strategy is bound to terminate on satisfiability problems in the theories of interest. It was shown in [5] that a standard, (refutationally) complete *rewrite-based inference system*, named $\mathcal{SP}$, is guaranteed to terminate on satisfiability problems in the quantifier-free theories of *equality*, *lists*, *arrays with and without extensionality*, *sets with extensionality* and the combination of lists and arrays. Thus, rewrite-based theorem provers can be used *off the shelf* as validity checkers, as done in, e.g., [10, 1].

This paper advances the rewriting approach to satisfiability in several ways. First, we prove termination of $\mathcal{SP}$ for the theories of *records with or without extensionality*, *integer offsets* and *integer offsets modulo*. Second, we give a *modularity theorem for combination of theories* stating sufficient conditions for $\mathcal{SP}$ to terminate on the combination, if it terminates on each theory separately. Any combination of the theories above, and with the *quantifier-free theory of equality* and *arrays (with or without extensionality)*, is covered. Third, we report on experiments comparing the rewrite-based E prover [17], CVC [21] and CVC Lite [6] on six sets of *parametric synthetic benchmarks*: three on arrays with extensionality, one combining arrays and integer offsets, one combining arrays, records and integer offsets to model *queues*, and one combining arrays, records and integer offsets modulo to model *circular queues*. CVC and CVC Lite seem to be the only state-of-the-art tools implementing a correct and complete procedure for arrays with extensionality.[4] Contrary to expectation, the general first-order prover with the theory presentations in input is, overall, comparable with the validity checkers with the theories built-in, and in many cases even outperforms them. To complete our appraisal, we tested E on sets of literals extracted from real-world problems of the UCLID suite [8], and found it solves them very fast.

[4] Neither Simplify nor ICS 2.0 are complete in this regard: cf. [11], Sec. 5, and a personal communication from H. Rueß to A. Armando in April 2004, respectively.

An extended abstract of this paper was presented in [3]. Very preliminary experiments with a few of the synthetic benchmarks were reported in [2]. A full version of this paper with proofs and a description of the synthetic benchmarks is available in [4].

2 A Rewrite-Based Methodology for $\mathcal{T}$-Satisfiability

$\mathcal{T}$-satisfiability is the problem of deciding satisfiability of sets $\mathcal{T} \cup S$, where $\mathcal{T}$ is a presentation of a (decidable) theory and S a set of ground equational literals on $\mathcal{T}$'s signature (or $\mathcal{T}$-literals). The rewrite-based methodology [5] applies first-order theorem-proving strategies based on the $\mathcal{SP}$ inference system (*superposition/paramodulation, reflection, equational factoring, subsumption, simplification and tautology deletion*, e.g., [15]). A theorem-proving strategy (inference system + search plan) is complete if the inference system is refutationally complete and the search plan is fair. A fundamental feature of $\mathcal{SP}$ is the usage of a *complete simplification ordering (CSO)* on terms and literals, in such a way that only maximal sides of maximal instances of literals are paramodulated into and from. Let $\mathcal{SP}_\succ$ be $\mathcal{SP}$ with CSO $\succ$ and *$\mathcal{SP}_\succ$-strategy* be any strategy with inference system $\mathcal{SP}_\succ$.

In the following, $\simeq$ is (unordered) equality, $=$ is identity, $\bowtie$ is either $\simeq$ or $\not\simeq$, l, r, u, t are terms, v, w, x, y, z are variables, all other lower case letters are constants or functions based on arity, and $Var(t)$ denotes the set of variables occurring in t. For a term t, $depth(t) = 0$, if t is a constant or a variable, and $depth(f(t_1, \ldots, t_n)) = 1 + max\{depth(t_i) : 1 \leq i \leq n\}$. A term is *flat* if its depth is 0 or 1. For a literal, $depth(l \bowtie r) = depth(l) + depth(r)$. A positive literal is *flat* if its depth is 0 or 1. A negative literal is *flat* if its depth is 0.

The rewrite-based methodology for $\mathcal{T}$-satisfiability consists of:

1. *$\mathcal{T}$-reduction:* specific inferences, depending on $\mathcal{T}$, are applied to remove certain literals or symbols and obtain an equisatisfiable *$\mathcal{T}$-reduced* problem.
2. *Flattening:* all ground literals are flattened by introducing new constants, yielding an equisatisfiable $\mathcal{T}$-reduced *flat* problem.
3. *Ordering selection and termination:* any fair $\mathcal{SP}_\succ$-strategy is shown to terminate when applied to a $\mathcal{T}$-reduced flat problem, provided $\succ$ is "good" for $\mathcal{T}$, or *$\mathcal{T}$-good*. An $\mathcal{SP}_\succ$-strategy will be *$\mathcal{T}$-good*, if $\succ$ is.

This methodology is *fully automated*, except for the proof of termination: the $\mathcal{T}$-reduction inferences are mechanical, flattening is a mechanical operation, and a theorem prover can generate $\mathcal{T}$-good orderings.

Let $\mathcal{E}$ denote the empty presentation, i.e., the presentation of the *quantifier-free theory of equality*. If $\mathcal{T}$ is $\mathcal{E}$, S is a set of ground equational literals built from free symbols, and $\mathcal{SP}_\succ$ reduces to ground completion, which is guaranteed to terminate, so that any fair $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{E}$.

2.1 The Theory of Arrays with and without Extensionality

Given sorts INDEX, ELEM and ARRAY, for indices, elements and arrays, respectively, and function symbols select : ARRAY $\times$ INDEX $\rightarrow$ ELEM, and store :

ARRAY × INDEX × ELEM → ARRAY, with the usual meaning, the standard presentation $\mathcal{A}$ consists of axioms (1) and (2), while the presentation $\mathcal{A}^e$ of the theory with extensionality also includes axiom (3) in the following list:

$$\forall x, z, v. \quad \text{select}(\text{store}(x, z, v), z) \simeq v \tag{1}$$

$$\forall x, z, w, v. \quad (z \not\simeq w \supset \text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w)) \tag{2}$$

$$\forall x, y. \quad (\forall z. \, \text{select}(x, z) \simeq \text{select}(y, z) \supset x \simeq y) \tag{3}$$

with variables x, y of sort ARRAY, w, z of sort INDEX and v of sort ELEM.

Definition 1. *A set of ground $\mathcal{A}$-literals is* $\mathcal{A}$-reduced *if it contains no literal $l \not\simeq r$, where l and r are terms of sort* ARRAY.

Definition 2. *A CSO $\succ$ is* $\mathcal{A}$-good *if (1) $t \succ c$ for all ground compound terms t and all constants c, and (2) $a \succ e \succ j$, for all constants a of sort* ARRAY*, e of sort* ELEM *and j of sort* INDEX.

$\mathcal{A}$-reduction replaces every literal $l \not\simeq r \in S$, with l and r of sort ARRAY, by $select(s, sk_{l,r}) \not\simeq select(t, sk_{l,r})$, where $sk_{l,r}$ is a Skolem constant of sort INDEX. Then (cf. [5], Th. 7.1), $\mathcal{A}^e \cup S$ is satisfiable if and only if $\mathcal{A} \cup Red_{\mathcal{A}}(S)$ is, where $Red_{\mathcal{A}}(S)$ is the $\mathcal{A}$-reduced form of S, and any additional function symbol other than select and store is *array-safe* (i.e., for $f\colon \mathrm{S}_0, \ldots, \mathrm{S}_{m-1} \longrightarrow \mathrm{S}_m$, with $m \geq 1$, S_k is not ARRAY for all k, $0 \leq k \leq m$).

Theorem 1. *A fair $\mathcal{A}$-good $\mathcal{SP}_{\succ}$-strategy is guaranteed to terminate on $\mathcal{A} \cup S$, where S is an $\mathcal{A}$-reduced set of ground flat $\mathcal{A}$-literals, and therefore is a satisfiability procedure for $\mathcal{A}$ and $\mathcal{A}^e$ (cf. Theorem 7.2 in [5]).*

2.2 The Theory of Records with and without Extensionality

Records are data structures that aggregate attribute-value pairs: assuming $Id = \{id_1, \ldots, id_n\}$ is a set of field identifiers and $\mathrm{T}_1, \ldots, \mathrm{T}_n$ are n types, $\text{REC}(id_1 : \mathrm{T}_1, \ldots, id_n : \mathrm{T}_n)$, abbreviated REC, is the sort of records that associate an element of type T_i to the field identifier id_i, for $1 \leq i \leq n$. The signature of the *theory of records* features a pair of function symbols $\text{rselect}_i : \text{REC} \to \mathrm{T}_i$ and $\text{rstore}_i : \text{REC} \times \mathrm{T}_i \to \text{REC}$ for each i, $1 \leq i \leq n$. Its presentations, $\mathcal{R}$, without extensionality, and $\mathcal{R}^e$, with extensionality, are given by the following axioms (only (4) and (5) in $\mathcal{R}$ and all three in $\mathcal{R}^e$):

$$\forall x, v. \quad \text{rselect}_i(\text{rstore}_i(x, v)) \simeq v \quad \text{for all } i,\ 1 \leq i \leq n \tag{4}$$

$$\forall x, v. \quad \text{rselect}_j(\text{rstore}_i(x, v)) \simeq \text{rselect}_j(x) \quad \text{for all } i, j,\ 1 \leq i \neq j \leq n \tag{5}$$

$$\forall x, y. \quad (\textstyle\bigwedge_{i=1}^{n} \text{rselect}_i(x) \simeq \text{rselect}_i(y) \supset x \simeq y) \tag{6}$$

where x, y have sort REC and v has sort T_i.

Definition 3. *A set of ground $\mathcal{R}$-literals is* $\mathcal{R}$-reduced *if it contains no literal $l \not\simeq r$, where l and r are terms of sort* REC.

Given a set of ground $\mathcal{R}$-literals S, *$\mathcal{R}$-reduction* consists of two phases. First, every literal $l \not\simeq r \in S$, with l and r of sort REC, is replaced by the disjunction $\bigvee_{i=1}^{n} \mathrm{rselect}_i(l) \not\simeq \mathrm{rselect}_i(r)$. Second, every such disjunction is split into n literals $\mathrm{rselect}_i(l) \not\simeq \mathrm{rselect}_i(r)$ for $1 \leq i \leq n$, replacing S by n sets $S_i = S \setminus \{l \not\simeq r\} \cup \{\mathrm{rselect}_i(l) \not\simeq \mathrm{rselect}_i(r)\}$ for $1 \leq i \leq n$. Each S_i is $\mathcal{R}$-reduced, and $Red_{\mathcal{R}}(S)$ denotes the class $\{S_i : 1 \leq i \leq n\}$.

Lemma 1. *$\mathcal{R}^e \cup S$ is satisfiable if and only if $\mathcal{R} \cup S_i$ is, for some $S_i \in Red_{\mathcal{R}}(S)$.*

Definition 4. *A CSO $\succ$ is* $\mathcal{R}$-good *if $t \succ c$ for all ground compound terms t and all constants c.*

Termination depends on a case analysis showing that only certain clauses can be generated, and the consideration that only finitely many such clauses can be built from a finite signature:

Theorem 2. *A fair $\mathcal{R}$-good $\mathcal{SP}_{\succ}$-strategy is guaranteed to terminate on $\mathcal{R} \cup S$, where S is an $\mathcal{R}$-reduced set of ground flat $\mathcal{R}$-literals, and therefore is a satisfiability procedure for $\mathcal{R}$ and $\mathcal{R}^e$.*

2.3 The Theory of Integer Offsets

The *theory of integer offsets* is a fragment of the theory of the integers, often applied in verification (e.g., [8]). Its signature has two unary function symbols s and p (the successor and predecessor functions, respectively) and its presentation $\mathcal{I}$ is given by the following infinite set of formulæ(e.g., [8, 13]):

$$\forall x.\ \mathrm{s}(\mathrm{p}(x)) \simeq x \tag{7}$$

$$\forall x.\ \mathrm{p}(\mathrm{s}(x)) \simeq x \tag{8}$$

$$\forall x.\quad \mathrm{s}^i(x) \not\simeq x \quad \text{for } i > 0 \tag{9}$$

where $\mathrm{s}^1(x) = \mathrm{s}(x)$, $\mathrm{s}^{i+1}(x) = \mathrm{s}(\mathrm{s}^i(x))$ for $i \geq 1$, and the formulæ in (9) are called *acyclicity axioms*. For convenience, let $Ac = \{\forall x.\, \mathrm{s}^i(x) \not\simeq x : i > 0\}$.

Definition 5. *A set of ground flat $\mathcal{I}$-literals is* $\mathcal{I}$-reduced *if it does not contain occurrences of* p.

Given a set S of ground flat $\mathcal{I}$-literals, the symbol p may appear only in literals of the form $\mathrm{p}(c) \simeq d$, because ground flat literals have the form $c \not\simeq d$ and do not contain p. *$\mathcal{I}$-reduction* consists of replacing every equation $\mathrm{p}(c) \simeq d$ in S by $c \simeq \mathrm{s}(d)$. The resulting $\mathcal{I}$-reduced form of S is denoted $Red_{\mathcal{I}}(S)$. $\mathcal{I}$-reduction reduces satisfiability with respect to $\mathcal{I}$ to satisfiability with respect to Ac, so that axioms (7) and (8) can be removed:

Lemma 2. *Let S be a set of ground flat $\mathcal{I}$-literals. $\mathcal{I} \cup S$ is satisfiable if and only if $Ac \cup Red_{\mathcal{I}}(S)$ is.*

The next step is to bound the number of axioms in Ac needed to solve the problem. Let $Ac(n) = \{\forall x.\, \mathrm{s}^i(x) \not\simeq x : 0 < i \leq n\}$. The intuition is that the bound will be given by the number of occurrences of s in S: since S is flat, having n occurrences of s means that there are n literals $\mathrm{s}(c_i) \simeq c_{i+1}$ for $0 \leq i \leq n$, which means a model must have $n+1$ distinct elements. Thus, it is sufficient to consider $Ac(n+1)$. From such a model it is possible to build a model for any larger instance of acyclicity by adding elements to the domain:

Lemma 3. *Let S be an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with n occurrences of* s*. Then $Ac \cup S$ is satisfiable if and only if $Ac(n+1) \cup S$ is.*

Termination puts no requirement on the ordering; in other words, any CSO is $\mathcal{I}$-good:

Theorem 3. *A fair $\mathcal{SP}_\succ$-strategy is guaranteed to terminate on $Ac(n+1) \cup S$, where S is an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals, and n is the number of occurrences of* s *in S, and therefore is a satisfiability procedure for $\mathcal{I}$.*

2.4 The Theory of Integer Offsets Modulo

The above treatment can be extended to the *theory of integer offsets modulo*, useful to describe data structures whose indices range over the integers modulo k, where k is a positive integer. The presentation $\mathcal{I}_k$ is obtained from $\mathcal{I}$ by replacing Ac with the following k axioms

$$\forall x.\quad \mathrm{s}^i(x) \not\simeq x \ \text{ for } 1 \leq i \leq k-1 \tag{10}$$

$$\forall x.\ \ \mathrm{s}^k(x) \simeq x \tag{11}$$

Let $C(k) = \{\forall x.\, \mathrm{s}^k(x) \simeq x\}$. Definition 5 and Lemma 2 apply also to $\mathcal{I}_k$, whereas Lemma 3 is no longer necessary, because $\mathcal{I}_k$ is finite to begin with. A case analysis of applicable inferences proves the following two results:

Lemma 4. *A fair $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $Ac(k-1) \cup C(k) \cup S$, where S is an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals.*

Alternatively, since $\mathcal{I}_k$ is finite, it is possible to omit $\mathcal{I}$-reduction and show termination of $\mathcal{SP}_\succ$ on the original problem format:

Lemma 5. *A fair $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $\mathcal{I}_k \cup S$, where S is a set of ground flat $\mathcal{I}$-literals.*

Theorem 4. *A fair $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{I}_k$.*

3 Combination of Theories

The rewrite-based approach is especially suited for combination of theories, because it combines presentations, rather than combining algorithms. Knowing that an $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{T}_1, \ldots, \mathcal{T}_n$, the combination problem is to show that it also decides satisfiability problems in the union $\mathcal{T} = \bigcup_{i=1}^n \mathcal{T}_i$. Of the rewrite-based methodology, $\mathcal{T}_i$-reduction applies separately for each theory, and flattening is harmless. Thus, one only has to prove termination. The main theorem in this section establishes sufficient conditions for $\mathcal{SP}$ to terminate on $\mathcal{T}$-satisfiability problems if it terminates on $\mathcal{T}_i$-satisfiability problems for all i, $1 \leq i \leq n$. A first condition is that the ordering be $\mathcal{T}$-*good*:

Definition 6. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be presentations of theories. A CSO $\succ$ is* $\mathcal{T}$-good, *where $\mathcal{T} = \bigcup_{i=1}^n \mathcal{T}_i$, if it is* $\mathcal{T}_i$-good *for all i, $1 \leq i \leq n$.*

A second condition will serve the purpose of excluding paramodulations from variables, when considering inferences across theories. This is key, since a variable may paramodulate into any proper non-variable subterm. A maximal literal $t \simeq x$ such that $x \in Var(t)$ cannot be used to paramodulate from x, since $t \succ x$, because a CSO includes the subterm ordering. Thus, it is sufficient to ensure that literals $t \simeq x$ such that $x \notin Var(t)$ are inactive:

Definition 7. *A clause C is* variable-inactive *for $\succ$, if for all its ground instances $C\sigma$ no maximal literal in $C\sigma$ is instance of an equation $t \simeq x$ where $x \notin Var(t)$. A set of clauses is* variable-inactive *for $\succ$, if all its clauses are.*

Definition 8. *A presentation $\mathcal{T}$ is* variable-inactive *for $\mathcal{SP}_\succ$, if S_∞ is variable-inactive for $\succ$ whenever $S_0 = \mathcal{T} \cup S$ is, where $S_\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} S_i$ is the limit of any fair $\mathcal{SP}_\succ$ derivation $S_0 \vdash_{\mathcal{SP}_\succ} S_1 \vdash_{\mathcal{SP}_\succ} \ldots S_i \vdash_{\mathcal{SP}_\succ} \ldots$ from S_0.*

For satisfiability problems, $S_0 \setminus \mathcal{T}$ is ground, so that S_0 is variable-inactive if the clauses in $\mathcal{T}$ are variable-inactive.

Last, the signatures of the $\mathcal{T}_i$'s may share constants, including constants introduced by flattening, but not function symbols. It follows that paramodulations from compound terms are excluded, and the only inferences across theories are paramodulations from constants into constants, that are finitely many, since there are finitely many constants:

Theorem 5. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be presentations of theories, with no shared function symbol, and let $\mathcal{T} = \bigcup_{i=1}^n \mathcal{T}_i$. Assume that for all i, $1 \leq i \leq n$, S_i is a $\mathcal{T}_i$-reduced set of ground flat $\mathcal{T}_i$-literals. If for all i, $1 \leq i \leq n$, a fair $\mathcal{T}_i$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate on $\mathcal{T}_i \cup S_i$, and $\mathcal{T}_i$ is variable-inactive for $\mathcal{SP}_\succ$, then, a fair $\mathcal{T}$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate on $\mathcal{T} \cup S_1 \cup \ldots \cup S_n$, and therefore is a satisfiability procedure for $\mathcal{T}$.*

All presentations considered in this paper satisfy the requirements of Theorem 5: $\mathcal{E}$ is variable-inactive vacuously, while $\mathcal{A}$, $\mathcal{R}$, $\mathcal{I}$ and $\mathcal{I}_k$ are variable-inactive by inspection of generated clauses in the proofs of their respective termination theorems. Furthermore, an $\mathcal{A}$-good ordering is also $\mathcal{R}$-good.

Corollary 1. *A fair $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for any combination of the theories of arrays, with or without extensionality, records, with or without extensionality, integer offsets or integer offsets modulo, and the quantifier-free theory of equality, provided (1) $\succ$ is $\mathcal{R}$-good whenever records are included, (2) $\succ$ is $\mathcal{A}$-good whenever arrays are included, and (3) all free function symbols are array-safe whenever arrays with extensionality and equality are included.*

In general, the requirement of being variable-inactive is rather natural for purely equational theories, where $\mathcal{T}$ is a set of equations:

Theorem 6. *If $\mathcal{T}$ is a presentation of a purely equational theory with no trivial models, then $\mathcal{T}$ is variable-inactive for $\mathcal{SP}_\succ$ for any CSO $\succ$.*

For first-order theories, variable-inactivity excludes, for instance, the generation of clauses in the form $a_1 \simeq x \vee \ldots \vee a_n \simeq x$, where the a_i's are constants, for $1 \leq i \leq n$. Such a disjunction may be generated only within a clause that contains at least one greater literal (e.g., involving function symbols). It is well-known (e.g., [7, 12]) that the Nelson-Oppen combination method [14] is complete without branching for *convex* theories, i.e., such that $\mathcal{T} \models S \supset \bigvee_{i=1}^{n} l_i \simeq r_i$ implies $\mathcal{T} \models S \supset l_j \simeq r_j$ for some j, $1 \leq j \leq n$, and S a set of $\mathcal{T}$-equations. It was shown in [7] that for a first-order theory with no trivial models convex implies *stably infinite*, i.e., such that any quantifier-free formula A has a $\mathcal{T}$-model only if it has an infinite $\mathcal{T}$-model. By applying this result, we have:

Theorem 7. *Let $\mathcal{T}$ be a presentation of a first-order theory with no trivial models. If $a_1 \simeq x \vee \ldots \vee a_n \simeq x \in S_\infty$, where S_∞ is the limit of any fair $\mathcal{SP}_\succ$-derivation from $S_0 = \mathcal{T} \cup S$, for any CSO $\succ$, then $\mathcal{T}$ is not convex.*

Thus, if $\mathcal{T}$ is not variable-inactive, because it generates some $a_1 \simeq x \vee \ldots \vee a_n \simeq x$, so that Theorem 5 does not apply, then $\mathcal{T}$ is not convex, and the Nelson-Oppen method does not apply either.

4 Experiments

We defined six sets of synthetic benchmarks: three in $\mathcal{A}^e$ (`STORECOMM`(n), `SWAP`(n) and `STOREINV`(n)), one combining $\mathcal{A}$ and $\mathcal{I}$ (`IOS`(n)), one combining $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{I}$ (`QUEUE`(n)) and one combining $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{I}_k$ (`CIRCULAR_QUEUE`(n, k)), with the common property that the set of formulæ grows monotonically with n. They were submitted to E 0.82, CVC 1.0a and CVC Lite 1.1.0. The prover E implements (a variant of) $\mathcal{SP}$ with a choice of search plans [17]. CVC [21] and CVC Lite [6] combine several $\mathcal{T}$-satisfiability procedures in the style of [14], including that of [22] for $\mathcal{A}^e$, and integrate them with a SAT engine. While CVC was superseded by CVC Lite, which is more modular and programmable, CVC is reportedly still faster on many problems.

A generator of pseudo-random instances of the benchmarks was written, producing either $\mathcal{T}$-reduced, flattened input files or plain input files. In the following,

native input means $\mathcal{T}$-reduced, flattened files for E, and plain files for CVC and CVC Lite. Run times (on a 3.00GHz 512MB RAM Pentium 4 PC, with time and memory limited to 150 sec and 256 MB per run) do not include flattening time, because flattening is a one-time linear time operation, and flattening time is insignificant. For those problems (`STORECOMM`(n) and `SWAP`(n)) such that a value of n determines a set of instances of the problem, rather than a single instance, the reported run time is the *median* over all tested instances,[5] because the median is well-defined even when a system fails on some, but not all instances of size n, a situation that occurs for all systems. (A failure is considered to be larger than all successful run times.)

The results for E refer to two strategies: *E(good-lpo)* features a lexicographic path ordering (LPO), while *E(std-kbo)* has a Knuth-Bendix ordering (KBO). *E(good-lpo)* is $\mathcal{A}$*-good*, whereas *E(std-kbo)* is $\mathcal{R}$*-good* but not $\mathcal{A}$*-good*. *E(good-lpo)* selects clauses by weight (symbol count where functions, constants and $\simeq$ weigh 2 and variables weigh 1), except for ensuring that all input clauses are selected before generated ones. *E(std-kbo)* gives ground clauses higher priority than non-ground clauses and ranks clauses of same priority by weight as above.

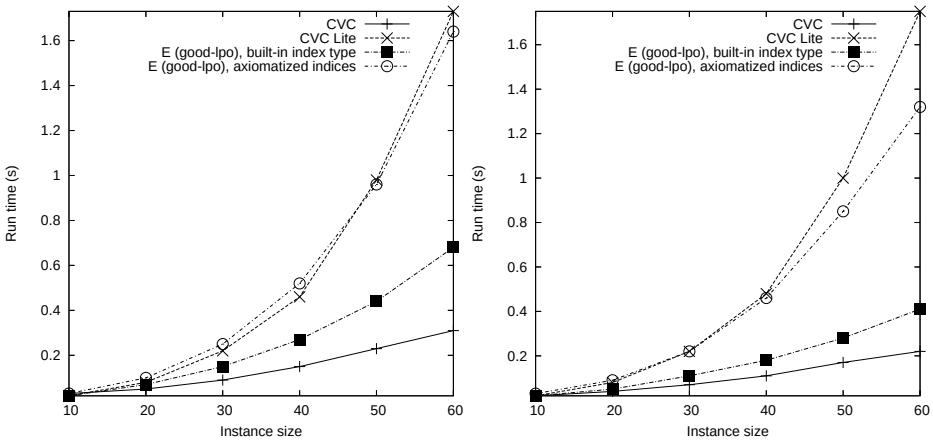

Fig. 1. Performances on valid (*left*) and invalid (*right*) `STORECOMM` instances, native input

We begin with the `STORECOMM` problems (Fig. 1 and 2). These sets include disequalities stating that all indices are distinct. Since many problems involve *distinct objects*, that is, constants that name elements known to be distinct in all models, E has a feature to build knowledge of distinct objects into the inference rules [18]. In Fig. 1 and 2, *built-in index type* refers to runs using this feature, while *axiomatized indices* refers to runs with the disequalities included in the

[5] The figures refer to runs with 9 instances for each value of n. Different numbers of instances (e.g., 5, 20) were tried, but the impact on performance was negligible.

input. On valid instances, *E(good-lpo)* with axiomatized indices and CVC Lite show nearly the same performance, with E apparently slightly ahead in the limit. *E(good-lpo)* with built-in indices outperforms CVC Lite by a factor of about 2.5. CVC performs best improving by another factor of 2. Perhaps surprisingly, since theorem provers are optimized for showing *unsatisfiability*, E performs even better on invalid (i.e., satisfiable) instances (Fig. 1, right), where it is faster than CVC Lite, and *E(good-lpo)* with built-in indices comes closer to CVC.

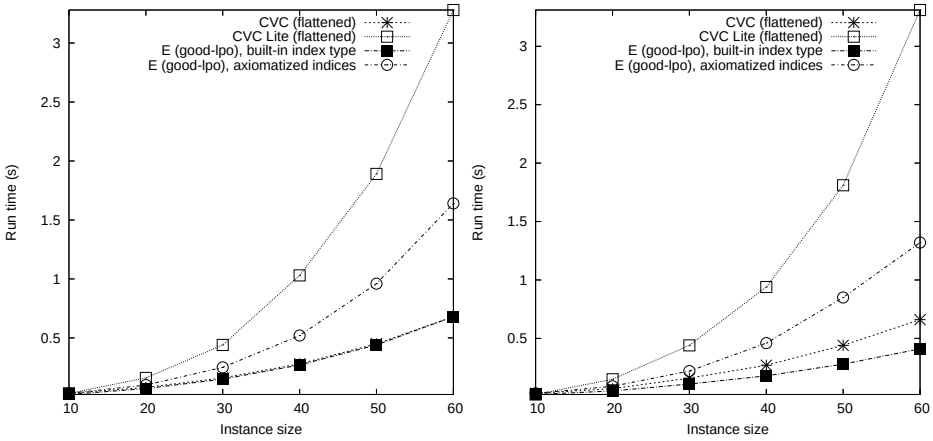

Fig. 2. Performances on valid (*left*) and invalid (*right*) `STORECOMM` instances, flat input for all

When *all* systems run on flattened input (Fig. 2), both CVC and CVC Lite exhibit run times approximately two times higher than with native format, and CVC Lite turns out to be the slowest system. CVC and E with built-in indices are the fastest: on valid instances, their performances are so close, that the plots are barely separable, but E is faster on invalid instances. Incidentally, it is not universally true that flattening hurt CVC and CVC Lite: on the `SWAP` problems CVC Lite performed better with flattened input. Although CVC is overall the fastest system on `STORECOMM`, E is faster than CVC Lite, and can do better than CVC on invalid instances when they are given the same input.

For `SWAP` (Fig. 3, left) the systems are very close up to instance size 5. Beyond this point, E leads up to size 7, but then is overtaken by CVC and CVC Lite. E can solve instances of size 8, but is much slower than CVC and CVC Lite, which solve instances up to size 9. No system can solve instances of size 10. For invalid instances, E solves easily instances up to size 10 in less than 0.5 sec, while CVC and CVC Lite are slower, taking 2 sec and 4 sec, respectively, and showing worse asymptotic behavior. Fig. 4 displays performances on valid instances, when the input for E includes the lemma $store(store(x, z, select(x, w)), w, select(x, z)) \simeq store(store(x, w, select(x, z)), z, select(x, w))$ that expresses "commutativity" of *store*. Although this addition means that the theorem prover is no longer a

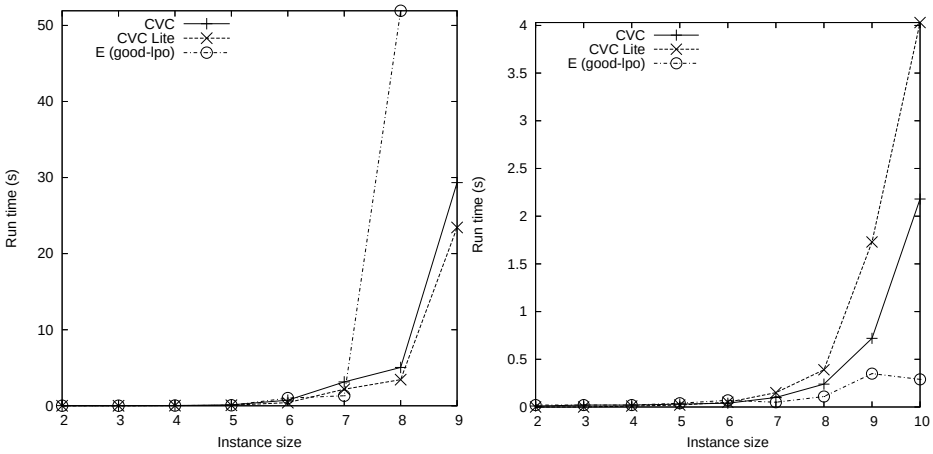

Fig. 3. Performances on valid (*left*) and invalid (*right*) `SWAP` instances, native input

decision procedure,[6] E terminates also on instances of size 9 and 10, and seems to show a better asymptotic behavior.

The comparison becomes even more favorable for the prover on `STOREINV` (Fig. 5). CVC solves valid instances up to size 8, CVC Lite goes up to size 9, but E solves instances of size 10, the largest generated. A comparison of run times at size 8 (the largest solved by all systems), gives 3.4 sec for E, 11 sec for CVC Lite, and 70 sec for CVC. Furthermore, *E(std-kbo)* (not shown in the figure) solves valid instances in *nearly constant time*, taking less than 0.3 sec for the hardest problem. For invalid instances, E does not do as well, but the run times here are minimal, as the largest, for instances of size 10, is about 0.1 sec.

The `IOS` problems (all valid) are encoded for CVC and CVC Lite by using built-in linear arithmetic (on the reals for CVC and on the integers for CVC Lite). We tried to use inductive types in CVC, but it performed badly and reported incorrect results.[7] In terms of performance (Fig. 6), CVC clearly wins, as expected from a system with built-in arithmetic. *E(good-lpo)* is no match, although it solves all tried instances (Fig. 6, left). *E(std-kbo)* does much better, because of the ordering, and because, by not preferring initial clauses, it does not consider the axioms early. *E(std-kbo)* also does better than CVC Lite: it scales smoothly, while CVC Lite displays oscillating run times, showing worse performance for even instance sizes than for odd ones (Fig. 6, right).

Fig. 7 (left) shows performances on plain queues: as expected, CVC is the fastest system, but *E(good-lpo)* competes very well with the systems with built-in linear arithmetic. Fig. 7 (right) does not include CVC, because CVC cannot

[6] Theorem 1 does not hold, if this lemma is added to the presentation.

[7] This is a known bug, that will not be fixed since CVC is no longer supported (personal communication from A. Stump to A. Armando, Feb. 2005). CVC Lite 1.1.0 does not support inductive types.

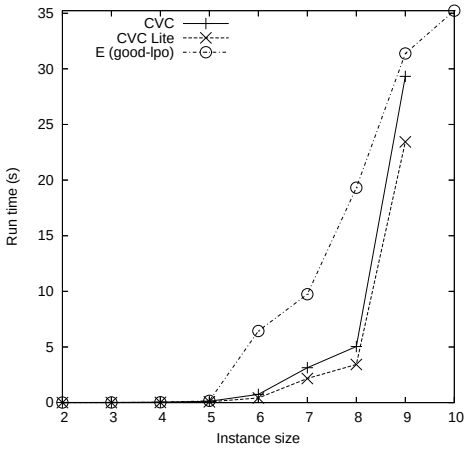

Fig. 4. Performances on valid `SWAP` instances with added lemma for E

handle arithmetic modulo. Between CVC Lite and E, the latter demonstrates a clear superiority: it shows nearly linear performance, and proves the largest instance in less than 0.5 sec, nine times faster than CVC Lite.

For "real-world" problems, we extracted the satisfiability problems generated by haRVey [10] from various UCLID inputs. This resulted in over 55,000 proof tasks in the *combination of the theories of integer offsets and equality*. These problems (all valid) were easy for E in automatic mode, where the prover chooses ordering and search plan. E could solve all problems, taking less than 4 sec on the hardest one, with average 0.372 sec and median 0.25 sec (on three 2.4GHz Pentium-4 PC's, all other parameters unchanged). Fig 8 shows a histogram of run times: the vast majority of problems is solved in less than 1 sec and very few need between 1.5 and 3 sec. An optimized search plan was found by testing on a random sample of 500 problems, or less than 1% of the full set. With this search plan, similar to *E(std-kbo)*, the performance improved by about 40% (Fig 8, right): the average is 0.249 sec, the median 0.12 sec, the longest time 2.77 sec, and the vast majority of problems is solved in less than 0.5 sec.

5 Discussion

The application of automated reasoning to verification has long shown the importance of $\mathcal{T}$-satisfiability procedures. The most common approach, popularized as the *"little proof engines"* paradigm [19], works by building each theory into a dedicated inference engine. By symmetry with "little proof engines," one may use *"big proof engines"* for theorem-proving strategies for first-order logic. Although there has always been a continuum between big and little engines of proof (viz., the research on *reasoning modulo a theory*), these two paradigms have also grown apart from each other to some extent. The *rewriting approach*

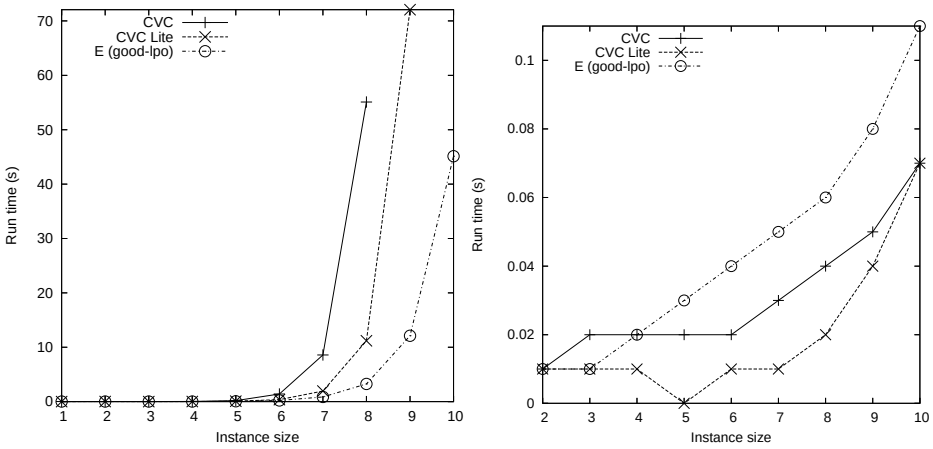

Fig. 5. Performances on valid (*left*) and invalid (*right*) `STOREINV` instances, native input

to satisfiability aims at replacing the apparent dichotomy ("little engines *versus* big engines") by a cross-fertilization ("big engines *as* little engines"). The general idea is to explore how big-engine technology (e.g., orderings, inference rules, search plans, algorithms, data structures, implementation techniques) may be applied selectively and efficiently "in the small."

The crucial point to use a first-order strategy as a $\mathcal{T}$-satisfiability procedure is to prove termination. We showed that a typical, complete, rewrite-based inference system for first-order logic, named $\mathcal{SP}$, is guaranteed to terminate on three new theories: *records, with and without extensionality*, *integer offsets*, and *integer offsets modulo*. For *combination of theories*, we gave a modularity theorem, stating sufficient conditions for $\mathcal{SP}$ to terminate on the combination, provided it terminates on each theory, and applied it to all theories under consideration.

Our experimental comparison of E with CVC and CVC Lite is the first of this kind. An analysis of E's traces showed that these $\mathcal{T}$-satisfiability problems behave very differently compared to more classical theorem-proving problems. The latter usually involve large presentations, with rich signatures and many universal variables, rewrite rules, and mixed positive/negative literal clauses. The search space is typically infinite and only a very small part of it can ever be explored. In $\mathcal{T}$-satisfiability, presentations are usually small, there is only one goal clause, and a large number of ground rewrite rules generated by flattening. The search space is finite, but nearly all of it has to be explored. Table 1 compares the behavior of E in automatic mode on medium-difficulty unsatisfiable array problems and representative TPTP problems of similar difficulty for the prover.[8] Considering these differences, the theorem prover turned out to be very competitive with the "little engines" systems, although it was optimized

[8] TPTP 3.0.0: TPTP or "Thousands of Problems for Theorem Provers" is a standard library. See `http://www.tptp.org/`.

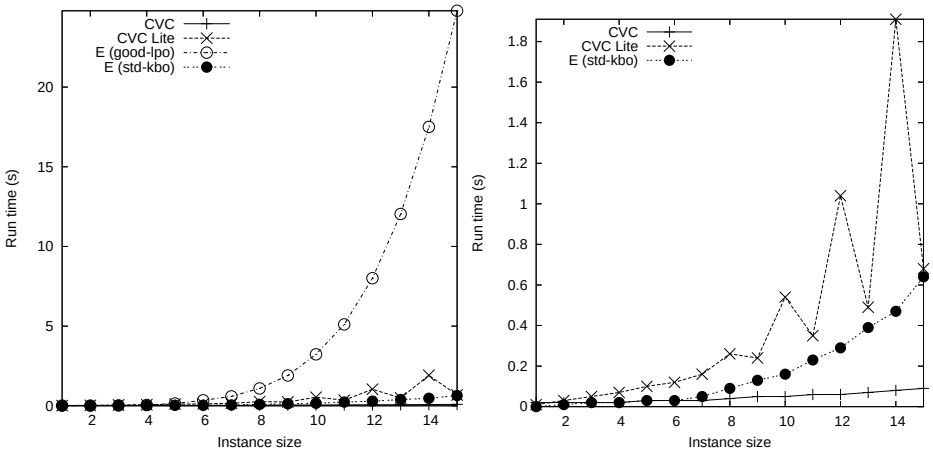

Fig. 6. Performances on `IOS` instances: on the right a rescaled version, with only the three fastest systems, of the same data on the left

for different search problems. Thus, further improvements might be obtained by studying search plans and implementation techniques of first-order inferences that target $\mathcal{T}$-satisfiability.

Table 1. Performance characteristics of array and TPTP problems

Problem Name	Initial clauses	Generated clauses	Processed clauses	Remaining clauses	Unnecessary inferences
`STORECOMM`(60)/1	1896	2840	4323	7	26.4%
`STOREINV`(5)	27	22590	7480	31	95.5%
`SWAP`(8)/3	62	73069	21743	56	98.2%
`SET015-4`	15	39847	7504	16219	99.90%
`FLD032-1`	31	44192	3964	31642	99.96%
`RNG004-1`	20	50551	4098	26451	99.90%

The prover terminated also beyond known termination results (e.g., Fig. 4, and the runs with *E(std-kbo)*), suggesting that theorem provers are not so brittle with respect to termination, and offer the flexibility of adding useful lemmata. Future directions for theoretical research include stronger termination theorems, and upper bounds on the number of generated clauses, assuming either blind saturation, or a fixed search plan, or a search plan of a given family. More general open issues are the integration with approaches to handle theories such as full linear arithmetics or bitvectors, and the application of "big engines" to more general $\mathcal{T}$-decision problems (arbitrary quantifier-free formulæ), whether by integration with a SAT solver (as explored first in [10]), or by using the prover's ability to handle first-order clauses.

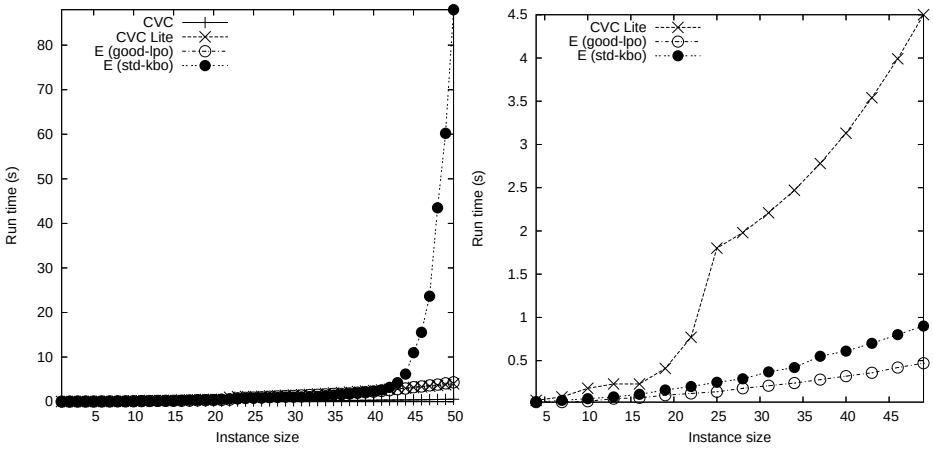

Fig. 7. Performances on `QUEUE` (*left*) and `CIRCULAR_QUEUE` with $k = 3$ (*right*)

Acknowledgements We thank Stefano Ferrari, former student of the second author, for running preliminary experiments, and Paolo Fiorini, colleague of the second author, for access to the computers of the robotics laboratory.

References

1. K. Arkoudas, K. Zee, V. Kuncak, and M. Rinard. Verifying a File System Implementation. In *Proc. ICFEM 2004*, volume 3308 of *LNCS*. Springer, 2004.
2. A. Armando, M. P. Bonacina, S. Ranise, M. Rusinowitch, and A. K. Sehgal. High-Performance Deduction for Verification: a Case Study in the Theory of Arrays. In *Notes of the 2nd VERIFY Workshop, 3rd FLoC*, number 07/2002 in Technical Reports, pages 103–112. DIKU, U. Copenhagen, 2002.
3. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. Big Proof Engines as Little Proof Engines: New Results on Rewrite-Based Satisfiability Procedures. In *Notes of the 3rd PDPAR Workshop, CAV-17*, Technical Reports. U. Edinburgh, 2005.
4. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. On a Rewriting Approach to Satisfiability Procedures: Theories of Data Structures, Combination Framework and Experimental Appraisal. Technical Report 36/2005, Dip. di Informatica, U. Verona, May 2005. `http://www.sci.univr.it/~bonacina/verify.html`.
5. A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Information and Computation*, 183(2):140–164, 2003.
6. C. W. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proc. CAV-16*, volume 3114 of *LNCS*, pages 515–518. Springer, 2004.
7. C. W. Barrett, D. L. Dill, and A. Stump. A Generalization of Shostak's Method for Combining Decision Procedures. In *Proc. FroCoS-4*, volume 2309 of *LNCS*. Springer, 2002.
8. R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and Verifying Systems Using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions. In *Proc. CAV-14*, volume 2404 of *LNCS*. Springer, 2002.

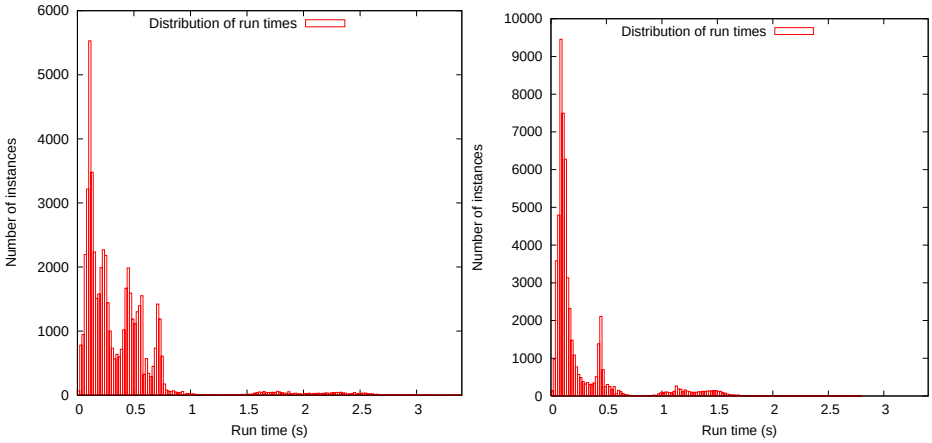

Fig. 8. Distribution of run times for E in automatic mode (*left*) and with optimized strategy (*right*) on the UCLID test set

9. L. de Moura, S. Owre, H. Rueß, J. Rushby, and N. Shankar. The ICS Decision Procedures for Embedded Deduction. In *Proc. IJCAR-2*, volume 3097 of *LNAI*, pages 218–222. Springer, 2004.
10. D. Déharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In *Proc. SEFM03*. IEEE, 2003.
11. D. L. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a Theorem Prover for Program Checking. Technical Report 148, HP Labs, 2003.
12. H. Ganzinger. Shostak Light. In *Proc. CADE-18*, volume 2392 of *LNAI*, pages 332–347. Springer, 2002.
13. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast Decision Procedures. In *Proc. CAV-16*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
14. G. Nelson and D. C. Oppen. Simplification by Cooperating Decision Procedures. *ACM TOPLAS*, 1(2):245–257, 1979.
15. R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In *Handbook of Automated Reasoning*, volume 1. Elsevier Science, 2001.
16. H. Rueß and N. Shankar. Deconstructing Shostak. In *Proc. LICS-16*. IEEE, 2001.
17. S. Schulz. E – A Brainiac Theorem Prover. *J. of AI Comm.*, 15(2–3):111–126, 2002.
18. S. Schulz and M. P. Bonacina. On Handling Distinct Objects in the Superposition Calculus. In *Notes of the 5th Int. Workshop on Implementation of Logics, LPAR-11*, pages 66–77, March 2005.
19. N. Shankar. Little Engines of Proof, 2002. Invited talk, 3rd FLoC, Copenhagen; `http://www.csl.sri.com/users/shankar/LEP.html`.
20. R. E. Shostak. Deciding Combinations of Theories. *J. ACM*, 31(1):1–12, 1984.
21. A. Stump, C. W. Barrett, and D. L. Dill. CVC: a Cooperating Validity Checker. In *Proc. CAV-14*, LNCS. Springer, 2002.
22. A. Stump, C. W. Barrett, D. L. Dill, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *Proc. LICS-16*. IEEE, 2001.