New results on rewrite-based satisfiability procedures

ALESSANDRO ARMANDO
Università degli Studi di Genova
MARIA PAOLA BONACINA
Università degli Studi di Verona
SILVIO RANISE
LORIA & INRIA-Lorraine
and
STEPHAN SCHULZ
Università degli Studi di Verona

Program analysis and verification require decision procedures to reason on theories of data structures. Many problems can be reduced to the *satisfiability* of sets of *ground literals* in theory T. If a sound and complete inference system for first-order logic is guaranteed to *terminate* on T-*satisfiability problems*, any theorem-proving strategy with that system and a fair search plan is a T-*satisfiability procedure*. We prove termination of a rewrite-based first-order engine on the theories of *records*, *integer offsets*, *integer offsets modulo* and *lists*. We give a *modularity theorem* stating sufficient conditions for termination on a *combination of theories*, given termination on each. The above theories, as well as others, satisfy these conditions. We introduce several sets of benchmarks on these theories and their combinations, including both *parametric* synthetic benchmarks to test *scalability*, and real-world problems to test performances on huge sets of literals. We compare the rewrite-based theorem prover E with the validity checkers CVC and CVC Lite. Contrary to the folklore that a general-purpose prover cannot compete with reasoners with built-in theories, the experiments are overall favorable to the theorem prover, showing that not only the rewriting approach is elegant and conceptually simple, but has important practical implications.

Categories and Subject Descriptors: I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Inference engines*

General Terms: Verification, Theory, Performance

Additional Key Words and Phrases: Automated reasoning, decision procedures, satisfiability modulo a theory, combination of theories, inference, superposition, rewriting, termination, scalability

1. INTRODUCTION

Decision procedures for satisfiability in theories of data types, such as arrays, lists and records, are at the core of many state-of-the-art verification tools (e.g., PVS [Owre et al. 1992], ACL2 [Kaufmann et al. 2000], Simplify [Detlefs et al. 2005], CVC [Barrett et al. 2000], ICS [de Moura et al. 2004], CVC Lite [Barrett and Berezin 2004], Zap [Lahiri and Musuvathi 2005a], MathSAT [Bozzano et al. 2005], Yices [Dutertre and de Moura 2006] and Barcelogic [Nieuwenhuis and Oliveras 2007]). The design, proof of correctness, and implementation of satisfiability procedures[1] present several issues, that have brought them to the forefront of research in automated reasoning applied to verification.

First, most verification problems involve more than one theory so that one needs procedures for combinations of theories, such as those pioneered by [Nelson and Oppen 1979] and [Shostak 1984]. Combination is complicated: for instance, understanding, formalizing and proving correct Shostak's method required much work (e.g., [Cyrluk et al. 1996; Rueß and Shankar 2001; Barrett et al. 2002b; Ganzinger 2002; Ranise et al. 2004]). The need for combination of theories means that decision procedures ought to be easy to modify, extend, integrate into, or at least interface, with other decision procedures or more general systems. Second, satisfiability procedures need to be proved correct and complete: a key part is to show that whenever the algorithm reports satisfiable, a model of the input does exist. Model-construction arguments for concrete procedures are specialized for those so that each new procedure requires a new proof. Frameworks that offer a higher level of abstraction (e.g., [Bachmair et al. 2003; Ganzinger 2002]) often focus on combining the quantifier-free theory of equality[2] with at most one additional theory, while problems from applications and existing systems combine many. Third, although systems begin to offer some support for adding theories, developers usually have to write a large amount of new code for each procedure with little software reuse and high risk of errors.

If one could use first-order theorem-proving strategies, combination would become conceptually much simpler because combining theories would amount to giving as input to the strategy 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 semidecision procedure for unsatisfiability. Existing first-order provers, that embody the results of years of research on data structures and algorithms for deduction could be applied or at least their code could be reused, offering a higher degree of assurance about soundness and completeness of the procedure. Furthermore, theorem-proving strategies support proof generation and model generation that are two more desiderata of satisfiability procedures (e.g., [Necula and Lee 1998; Stump and Dill 2002; Lahiri and Musuvathi 2005a]) in

---

[1]In the literature on decision procedures, a "satisfiability procedure" is a decision procedure for "satisfiability problems" that are sets of ground literals.

[2]Also known as *EUF* for Equality with Un-interpreted Function symbols. In the literature on decision procedures, most authors use "interpreted" and "un-interpreted" to distinguish between those symbols whose interpretation is restricted to the models of a given theory and those whose interpretation is unrestricted. In the literature on rewriting, it is more traditional to use "definite" in place of "interpreted" and "free" in place of "un-interpreted", as done in [Ganzinger 2002].

a theory-independent way. Indeed, if the input is unsatisfiable, the strategy generates a proof with no additional effort. If it is satisfiable, the strategy generates a saturated set that if finite may form a basis for model generation [Caferra et al. 2004].

The crux is *termination*: in order 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. Results of this nature were obtained in [Armando et al. 2003]: a refutationally complete *rewrite-based inference system*, named $\mathcal{SP}$ (from superposition), was shown to generate finitely many clauses on satisfiability problems in the theories of *nonempty lists*, *arrays with or without extensionality*, *encryption*, *finite sets with extensionality*, *homomorphism*, and the combination of lists and arrays. This work was extended in [Lynch and Morawska 2002], by using a meta-saturation procedure to add complexity characterizations[3]. Since the inference system $\mathcal{SP}$ reduces to ground completion on a set of ground equalities and inequalities, it terminates and represents a decision procedure also for the quantifier-free theory of equality[4].

These termination results suggest that, at least in principle, rewrite-based theorem provers might be used "off the shelf" as validity checkers. The common expectation, however, is that validity checkers with built-in theories will be much faster than theorem provers that take theory presentations as input. In this article, we bring evidence that using rewrite-based theorem provers can be a pratical option. Our contributions include:

—New termination results, showing that $\mathcal{SP}$ generates finitely many clauses from satisfiability problems in the theories of more data structures, *records with or without extensionality* and *possibly empty lists*, and in two fragments of integer arithmetic, the theories of *integer offsets* and *integer offsets modulo*;

—A general *modularity theorem*, that states sufficient conditions for $\mathcal{SP}$ to terminate on satisfiability problems in a union of theories, if it terminates on the satisfiability problems of each theory taken separately;

—A report on experiments where six sets of parametric synthetic benchmarks were given to the rewrite-based theorem prover E [Schulz 2002; 2004], the *Cooperating Validity Checker* CVC [Stump et al. 2002] and its successor CVC Lite [Barrett and Berezin 2004]: contrary to expectation, the general first-order prover with the theory presentations in input was, overall, comparable with the validity checkers with built-in theories, and in some cases even outperformed them.

Among the termination results, the one for the theory of *integer offsets* is perhaps the most surprising, because the axiomatization is infinite. All the theories considered in this article (i.e., *records with or without extensionality*, *lists*, *integer offsets*, *integer offsets modulo*, *arrays with or without extensionality* and the quantifier-free theory of equality) satisfy the hypotheses of the modularity theorem, so that a fair $\mathcal{SP}$-strategy is a satisfiability procedure for *any* of their combinations. This shows the flexibility of the rewrite-based approach.

---

[3]Meta-saturation as in [Lynch and Morawska 2002] was later corrected in [Lynch and Tran 2007].
[4]That ground completion can be used to compute congruence closure has been known since [Lankford 1975].

For the experiments, we chose a state-of-the-art theorem prover that implements $\mathcal{SP}$, and two systems that combine decision procedures with built-in theories *à la* Nelson-Oppen. At the time of these experiments, CVC and CVC Lite were the only state-of-the-art tools implementing a correct and complete procedure for arrays with extensionality[5], namely that of [Stump et al. 2001]. We worked with *parametric* synthetic benchmarks, because they allow one to assess the *scalability* of systems by a sort of *experimental asymptotic analysis*. Three sets of benchmarks involve the theory of *arrays with extensionality*, one combines the theory of *arrays* with that of *integer offsets*, one is about *queues*, and one is about *circular queues*. In order to complete our appraisal, we tested E on sets of literals extracted from real-world problems of the UCLID suite [Lahiri and Seshia 2004], and found it solves them extremely fast. The selection of problems emphasizes *the combination of theories*, because it is relevant in practice. The synthetic benchmarks on *queues* feature the theories of *records*, *arrays* and *integer offsets*, because a queue can be modelled as a record, that unites a partially filled array with two indices that represent *head* and *tail*. Similarly, the benchmarks on *circular queues* involve the theories of *records*, *arrays* and *integer offsets modulo*, because a circular queue of length k is a queue whose indices take integer values modulo k. The UCLID problems combine the *theory of integer offsets* and the *quantifier-free theory of equality*.

1.1   Previous Work

Most termination results for theorem-proving methods are based on identifying generic syntactic constraints that the input must satisfy to induce termination (e.g., [Fermüller et al. 2001; Caferra et al. 2004] for two overviews). Our results are different, because they apply to specific theories, and in this respect they can be considered of a more semantic nature. There are a few other recent works that experiment with the application of first-order theorem provers to decidable theories of data structures. A proof of correctness of a basic Unix-style file system implementation was obtained in [Arkoudas et al. 2004], by having a proof checker invoke the SPASS [Weidenbach et al. 1999] and Vampire [Riazanov and Voronkov 2002] provers for noninductive reasoning on *lists* and *arrays*, on the basis of their first-order presentations. The haRVey system [Déharbe and Ranise 2003] is a verification tool based on the rewriting approach that we propound in this paper. It integrates the E prover with a SAT solver, based on ordered binary decision diagrams, to implement decision procedures for a few theories. Experiments with haRVey offered additional evidence of the effectiveness of the rewriting approach [Ranise and Déharbe 2003].

The collection of theories considered here is different from that treated in [Armando et al. 2003]. Lists *à la Shostak* (with cons, car, cdr and three axioms) and lists *à la Nelson-Oppen* (with cons, car, cdr, atom and four axioms) were covered in [Armando et al. 2003]. Both axiomatize *nonempty* lists, since there is no symbol such as *nil* to represent the empty list. Here we consider a different presentation, with cons, car, cdr, nil and six axioms, that allows for empty lists. In an approach where the axioms are given in input to a theorem prover, a different presenta-

---

[5]Neither Simplify nor ICS are complete in this regard: cf. Section 5 in [Detlefs et al. 2005] and [Rueß 2004], respectively.

tion represents a different problem, because termination on satisfiability problems including a presentation does not imply termination on satisfiability problems including another presentation. To wit, the finite saturated sets generated by $\mathcal{SP}$ are different (cf. Lemma 11 in this article with Lemmata 4.1 and 5.1 in [Armando et al. 2003]). Application of a rewrite-based engine to the theories of records, integer offsets and integer offsets modulo is studied here for the first time. Although the presentation of the theory of records resembles that of arrays, the treatment of extensionality is different and the generated saturated sets are very different (cf. Lemma 2 and 14 in this article). The only overlap with [Armando et al. 2003] is represented by the theory of arrays, for which we redo only the case analysis of generated clauses, because that reported in [Armando et al. 2003] is incomplete (cf. Lemma 14 in this article with Lemma 7.2 in [Armando et al. 2003]). A short version and an extended abstract of this article were presented in [Armando et al. 2005b] and [Armando et al. 2005a], respectively. Very preliminary experiments with a few of the synthetic benchmarks were reported in [Armando et al. 2002].

2. BACKGROUND

We employ the basic notions from logic usually assumed in theorem proving. For notation, the symbol $\simeq$ denotes equality;[6] $\bowtie$ stands for either $\simeq$ or $\not\simeq$; $=$ denotes identity; l, r, u, t are terms; v, w, x, y, z are variables; other lower-case Latin letters are constant or function symbols based on arity; L is a literal; C and D denote clauses, that is, multisets of literals interpreted as disjunctions; φ is a formula; and σ is used for substitutions. More notation will be introduced as needed.

A theory is presented by a set of sentences, called its *presentation* or *axiomatization*. Given a presentation $\mathcal{T}$, the *theory Th $\mathcal{T}$* is the set of all its logical consequences, or theorems: *Th $\mathcal{T} = \{\varphi \mid \mathcal{T} \models \varphi\}$*. Thus, a theory is a deductively-closed presentation. An *equational theory* is a theory presented by a set of universally quantified equations. A *Horn clause* is a clause with at most one positive literal, and a *definite Horn clause*, or non-negative Horn clause, is a clause with at most and at least one positive literal. A *Horn theory* is presented by a set of non-negative Horn clauses, and a *Horn equational theory* is a Horn theory where the only predicate is equality. From a model-theoretic point of view, the term *theory* refers to the family of *models* of $\mathcal{T}$, or $\mathcal{T}$-models. A model is called *trivial* if its domain has only one element. It is customary to ascribe to a model the cardinality of its domain, so that a model is said to be *finite* or *infinite* if its domain is.

By $\mathcal{T}$-atom, $\mathcal{T}$-literal, $\mathcal{T}$-clause, $\mathcal{T}$-sentence and $\mathcal{T}$-formula, we mean an atom, a literal, a clause, a sentence and a formula, respectively, on $\mathcal{T}$'s signature, omitting the $\mathcal{T}$ when it is clear from context. Equality is the only predicate, so that all $\mathcal{T}$-atoms are $\mathcal{T}$-equations. The problem of $\mathcal{T}$-*satisfiability*, or, equivalently, *satisfiability modulo* $\mathcal{T}$, is the problem of deciding whether a set S of ground $\mathcal{T}$-literals is satisfiable in $\mathcal{T}$, or has a $\mathcal{T}$-model. The more general $\mathcal{T}$-*decision problem* consists of deciding whether a set S of quantifier-free $\mathcal{T}$-formulæ is satisfiable in $\mathcal{T}$. In principle, the $\mathcal{T}$-decision problem can be reduced to the $\mathcal{T}$-satisfiability problem via reduction of every quantifier-free $\mathcal{T}$-formula to disjunctive normal form. However, this is not pratical in general. In this paper, we are concerned only with

---

[6]The notation $\simeq$ is standard for unordered pair, so that $l \simeq r$ stands for $l \simeq r$ or $r \simeq l$.

$\mathcal{T}$-*satisfiability*.[7] $\mathcal{T}$-satisfiability is important, because many problems reduce to $\mathcal{T}$-satisfiability: the *word problem*, or the problem of deciding whether $\mathcal{T} \models \forall \bar{x} \, l \simeq r$, where $l \simeq r$ is a $\mathcal{T}$-equation, the *uniform word problem*, or the problem of deciding whether $\mathcal{T} \models \forall \bar{x} \, C$, where C is a Horn $\mathcal{T}$-clause, and the *clausal validity problem*, or the problem of deciding whether $\mathcal{T} \models \forall \bar{x} \, C$, where C is a $\mathcal{T}$-clause, all reduce, through skolemization, to deciding the unsatisfiability of a set of ground literals, since all variables are universally quantified.

The traditional approach to $\mathcal{T}$-satisfiability is that of *"little" engines of proofs* (e.g., [Shankar 2002]), which consists of building each theory into a dedicated inference engine. Since the theory is built into the engine, the input of the procedure consists of S only. The most basic example is that of congruence closure algorithms for satisfiability of sets of ground equalities and inequalities (e.g., [Shostak 1978; Nelson and Oppen 1980; Downey et al. 1980; Bachmair et al. 2003])[8]. Theories are built into the congruence closure algorithm by generating the necessary instances of the axioms (see [Nelson and Oppen 1980] for nonempty lists) or by adding preprocessing with respect to the axioms and suitable case analyses (see [Stump et al. 2001] for arrays with extensionality). Theories are combined by using the method of [Nelson and Oppen 1979]. Two properties relevant to this method are *convexity* and *stable infiniteness*:

DEFINITION 1. *A theory Th $\mathcal{T}$ is* convex, *if for any conjunction H of $\mathcal{T}$-atoms and for $\mathcal{T}$-atoms P_i, $1 \le i \le n$, $\mathcal{T} \models H \supset \bigvee_{i=1}^{n} P_i$ implies that there exists a j, $1 \le j \le n$, such that $\mathcal{T} \models H \supset P_j$.*

In other words, if $\bigvee_{i=1}^{n} P_i$ is true in all models of $\mathcal{T} \cup H$, there exists a P_j that is true in all models of $\mathcal{T} \cup H$. This excludes the situation where all models of $\mathcal{T} \cup H$ satisfy some P_j, but no P_j is satisfied by all. Since Horn theories are those theories whose models are closed under intersection – a fact due to Alfred Horn [Horn 1951, Lemma 7] – it follows that Horn theories, hence equational theories, are convex. The method of Nelson-Oppen without case analysis (also known as "branching" or "splitting") is complete for combinations where all theories are convex (e.g., [Nelson and Oppen 1979; Barrett et al. 2002b; Ganzinger 2002]). The method of Nelson-Oppen with case analysis is complete for combinations where all involved theories are stably infinite [Tinelli and Harandi 1996]:

DEFINITION 2. *A theory Th $\mathcal{T}$ is* stably infinite, *if for any quantifier-free $\mathcal{T}$-formula φ, φ has a $\mathcal{T}$-model if and only if it has an infinite $\mathcal{T}$-model.*

When combining the quantifier-free theory of equality with only one other theory, the requirement of stable infiniteness can be dropped [Ganzinger 2002]. For first-order logic, compactness implies that if a set of formulæ has models with domains of arbitrarily large finite cardinality, then it has models with infinite domains (e.g., [van Dalen 1989] for a proof). Thus, for $\mathcal{T}$ a first-order presentation, and φ a quantifier-free $\mathcal{T}$-formula, if φ has arbitrarily large finite $\mathcal{T}$-models, it has infinite $\mathcal{T}$-models; or, equivalently, if it has no infinite $\mathcal{T}$-model, there is a finite bound

---

[7]We discuss existing approaches and future directions for the $\mathcal{T}$-decision problem in Section 7.
[8]Unknown to most, the conference version of [Downey et al. 1980] appeared in [Downey et al. 1978] with a different set of authors.

on the size of its $\mathcal{T}$-models.[9] Using this property, one proves (cf. Theorem 4 in [Barrett et al. 2002b]):

THEOREM 2.1. *(Barrett, Dill and Stump 2002) Every convex first-order theory with no trivial models is stably-infinite.*

Thus, stable infiniteness is a weaker property characterizing the theories that can be combined according to the Nelson-Oppen scheme.

If a decision procedure with a built-in theory is a *little engine of proof*, an inference system for full first-order logic with equality can be considered a *"big" engine of proof* (e.g., [Stickel 2002]). One such engine is the *rewrite-based inference system* $\mathcal{SP}$, whose *expansion* and *contraction* inference rules are listed in Figures 1 and 2, respectively. Expansion rules *add* what is below the inference line to the clause set that contains what is above the inference line. Contraction rules *remove* what is above the double inference line and add what is below the double inference line. Combinations of these inference rules or variants thereof form the core of most theorem provers for first-order logic with equality, such as Otter [McCune 2003], SPASS [Weidenbach et al. 1999], Vampire [Riazanov and Voronkov 2002], and E [Schulz 2002], to name a few. Formulations with different terminologies (e.g., left and right superposition in place of paramodulation and superposition) appear in the vast literature on the subject (e.g., [Plaisted 1993; Bonacina 1999; Nieuwenhuis and Rubio 2001; Dershowitz and Plaisted 2001] for surveys where more references can be found).

A fundamental assumption of rewrite-based inference systems is that the universe of terms, hence those of literals and clauses, is ordered by a *well-founded ordering*. $\mathcal{SP}$ features a *complete simplification ordering (CSO)* $\succ$ on terms, extended to literals and clauses by multiset extension as usual. A simplification ordering is *stable* ($l \succ r$ implies $l\sigma \succ r\sigma$ for all substitutions σ), *monotonic* ($l \succ r$ implies $t[l] \succ t[r]$ for all t, where the notation $t[l]$ represents a term where l appears as subterm in context t), and has the *subterm property* (i.e., it contains the *subterm ordering* $\rhd$: $l \rhd r$ implies $l \succ r$). An ordering with these properties is well-founded. A CSO is also total on ground terms. The most commonly used CSO's are instances of the *recursive path ordering (RPO)* and the *Knuth-Bendix ordering (KBO)*. An RPO is based on a *precedence* (i.e., a partial ordering on the signature) and the attribution of a *status* to each symbol in the signature (either lexicographic or multiset status). If all symbols have lexicographic status, the ordering is called *lexicographic (recursive) path ordering (LPO)*. A KBO is based on a *precedence* and the attribution of a *weight* to each symbol. All instances of RPO and KBO are simplification orderings. All instances of KBO and LPO based on a total precedence are CSO's. Definitions, results and references on orderings for rewrite-based inference can be found in [Dershowitz and Plaisted 2001].

A well-founded ordering $\succ$ provides the basis for a notion of *redundancy*: a ground clause C is redundant in S if for ground instances $\{D_1, \ldots D_k\}$ of clauses in S it is $\{D_1, \ldots D_k\} \models C$ and $\{D_1, \ldots D_k\} \prec \{C\}$; a clause is redundant if all its ground

---

[9]A proof of this consequence of compactness in the context of decision procedures appears in [Ganzinger et al. 2004], where φ is assumed to have been reduced to disjunctive normal form, so that the proof is done for a set of literals.

Superposition	$\dfrac{C \vee l[u'] \simeq r \quad D \vee u \simeq t}{(C \vee D \vee l[t] \simeq r)\sigma}$ $(i),\ (ii),\ (iii),\ (iv)$
Paramodulation	$\dfrac{C \vee l[u'] \not\simeq r \quad D \vee u \simeq t}{(C \vee D \vee l[t] \not\simeq r)\sigma}$ $(i),\ (ii),\ (iii),\ (iv)$
Reflection	$\dfrac{C \vee u' \not\simeq u}{C\sigma}$ $\forall L \in C : (u' \not\simeq u)\sigma \not\prec L\sigma$
Equational Factoring	$\dfrac{C \vee u \simeq t \vee u' \simeq t'}{(C \vee t \not\simeq t' \vee u \simeq t')\sigma}$ $(i),\ \forall L \in \{u' \simeq t'\} \cup C : (u \simeq t)\sigma \not\prec L\sigma$

where σ is the most general unifier (mgu) of u and u', u' is not a variable in *Superposition* and *Paramodulation*, and the following abbreviations hold:

(i). is $u\sigma \not\preceq t\sigma$,
(ii). is $\forall L \in D : (u \simeq t)\sigma \not\preceq L\sigma$,
(iii). is $l[u']\sigma \not\preceq r\sigma$, and
(iv). is $\forall L \in C : (l[u'] \bowtie r)\sigma \not\preceq L\sigma$.

Fig. 1.   Expansion inference rules of $\mathcal{SP}$.

Strict Subsumption	$\dfrac{C \quad D}{C}$	$D \succ\!\!\!\!\cdot\ C$
Simplification	$\dfrac{C[u] \quad l \simeq r}{C[r\sigma], \quad l \simeq r}$	$u = l\sigma, \quad l\sigma \succ r\sigma, \quad C[u] \succ (l \simeq r)\sigma$
Deletion	$\dfrac{C \vee t \simeq t}{}$	

where $D \succ\!\!\!\!\cdot\ C$ if $D \succeq\!\!\!\!\cdot\ C$ and $C \not\succeq\!\!\!\!\cdot\ D$; and $D \succeq\!\!\!\!\cdot\ C$ if $C\sigma \subseteq D$ (as multisets) for some substitution σ. In practice, theorem provers such as E also apply subsumption of variants (if $D \succeq\!\!\!\!\cdot\ C$ and $C \succeq\!\!\!\!\cdot\ D$, the oldest clause is retained) and tautology deletion (that removes clauses such as $C \vee t \simeq u \vee t \not\simeq u$).

Fig. 2.   Contraction inference rules of $\mathcal{SP}$.

instances are. An inference is redundant if it uses or generates a redundant clause and a set of clauses is *saturated* if all expansion inferences in the set are redundant. In $\mathcal{SP}$, clauses deleted by contraction are redundant and expansion inferences that do not respect the ordering constraints are redundant.

Let $\mathcal{SP}_{\succ}$ be $\mathcal{SP}$ with CSO $\succ$. An $\mathcal{SP}_{\succ}$-*derivation* is a sequence of sets of clauses

$$S_0 \underset{\mathcal{SP}_{\succ}}{\vdash} S_1 \underset{\mathcal{SP}_{\succ}}{\vdash} \dots S_i \underset{\mathcal{SP}_{\succ}}{\vdash} \dots$$

where at each step an $\mathcal{SP}_{\succ}$-inference is applied. A derivation is characterized by its *limit*, defined as the set of *persistent clauses*

$$S_\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} S_i.$$

A derivation is *fair* if all expansion inferences become redundant eventually, and a fair derivation generates a *saturated* limit.

Since inference systems are non-deterministic, a *theorem-proving strategy* is obtained by adding a *search plan*, that drives rule application. A search plan is fair if it only generates fair derivations. An $\mathcal{SP}_{\succ}$-*strategy* is a theorem-proving strategy

with inference system $\mathcal{SP}_{\succ}$. If the inference system is *refutationally complete* and the search plan is *fair*, the theorem-proving strategy is *complete*: S_{∞} is saturated and the empty clause $\square$ is in S_{∞} if and only if S_0 is unsatisfiable. A proof of the refutational completeness of $\mathcal{SP}$ can be found in [Nieuwenhuis and Rubio 2001] and definitions and references for redundancy, saturation and fairness in, for example, [Bonacina and Hsiang 1995; Nieuwenhuis and Rubio 2001; Bonacina and Dershowitz 2007].

For additional notations and conventions used in this article, $Var(t)$ denotes the set of variables occurring in t; the *depth* of a term t is written $depth(t)$, and $depth(t) = 0$, if t is either a constant or a variable, $depth(t) = 1 + max\{depth(t_i): 1 \leq i \leq n\}$, if t is a compound term $f(t_1, \ldots, t_n)$. 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. Let $\Gamma = \langle D, J \rangle$ be the interpretation with domain D and interpretation function J. Since our usage of interpretations is fairly limited, we use Γ without specifying D or J whenever possible. Lower case letters surmounted by a hat, such as $\hat{d}$ and $\hat{e}$, denote elements of the domain D. As usual, $[t]_{\Gamma}$ denotes the interpretation of term t in Γ. Generalizing this notation, if c is a constant symbol and f a function symbol, we use $[c]_{\Gamma}$ in place of $J(c)$ for the interpretation of c in Γ and $[f]_{\Gamma}$ in place of $J(f)$ for the interpretation of f in Γ. Small capital letters, such as s, denote sorts. If there are many sorts, D is replaced by a tuple of sets, one per sort, and $[s]_{\Gamma}$ denotes the one corresponding to s in Γ.

3. REWRITE-BASED SATISFIABILITY PROCEDURES

The rewriting approach to $\mathcal{T}$-satisfiability aims at applying an inference system such as $\mathcal{SP}$ to clause sets $S_0 = \mathcal{T} \cup S$, where $\mathcal{T}$ is a presentation of a theory and S a set of ground $\mathcal{T}$-literals. This is achieved through the following phases, that, together, define a *rewrite-based methodology for satisfiability procedures*:

(1) $\mathcal{T}$*-reduction.* Specific inferences, depending on $\mathcal{T}$, are applied to the problem to remove certain literals or symbols and obtain an equisatisfiable $\mathcal{T}$*-reduced* problem.

(2) *Flattening.* All ground literals are transformed into flat literals, or flattened, by introducing new constants and new equations, yielding an equisatisfiable $\mathcal{T}$-reduced *flat* problem. For example, a literal $store(a_1, i_1, v_1) \simeq store(a_2, i_2, v_2)$ is replaced by the literals $store(a_1, i_1, v_1) \simeq c_1$, $store(a_2, i_2, v_2) \simeq c_2$ and $c_1 \simeq c_2$. Depending on $\mathcal{T}$, flattening may precede or follow $\mathcal{T}$-reduction.

(3) *Ordering selection and termination.* $\mathcal{SP}_{\succ}$ is shown to generate finitely many clauses when applied fairly to a $\mathcal{T}$-reduced flat problem. Such a result may depend on simple properties of the ordering $\succ$: an ordering that satisfies them is termed $\mathcal{T}$*-good*, and an $\mathcal{SP}_{\succ}$-strategy is $\mathcal{T}$*-good* if $\succ$ is. It follows that a fair $\mathcal{T}$-good $\mathcal{SP}_{\succ}$-strategy is guaranteed to terminate on a $\mathcal{T}$-reduced flat problem. The $\mathcal{T}$-goodness requirement may be vacuous, meaning that any CSO is $\mathcal{T}$-good.

This methodology can be *fully automated*, except for the proof of termination and the definition of $\mathcal{T}$-goodness: indeed, $\mathcal{T}$-reduction is made of mechanical inferences, flattening is a mechanical operation, and contemporary theorem provers feature

mechanisms to generate automatically orderings for given signatures and with given properties.

Let $\mathcal{E}$ denote the empty presentation, that is, 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 function and constant symbols, and $\mathcal{SP}_\succ$ reduces to ground completion, which is guaranteed to terminate, with no need of flattening, $\mathcal{T}$-reduction or $\mathcal{T}$-goodness. Therefore, any fair $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for the quantifier-free theory of equality. In the rest of this section we apply the rewrite-based methodology to several theories. For each theory, the signature contains the function symbols indicated and a finite set of constant symbols. A supply of countably many new constant symbols is assumed to be available for flattening.

3.1 The Theory of Records

Records aggregate attribute-value pairs. Let $Id = \{id_1, \ldots, id_n\}$ be a set of attribute identifiers and $\mathrm{T}_1, \ldots, \mathrm{T}_n$ be n sorts. Then, $\mathrm{REC}(id_1 : \mathrm{T}_1, \ldots, id_n : \mathrm{T}_n)$, abbreviated REC, is the sort of records that associate a value of sort T_i to the attribute identifier id_i, for $1 \leq i \leq n$. The signature of the *theory of records* has a pair of function symbols $\mathrm{rselect}_i : \mathrm{REC} \to \mathrm{T}_i$ and $\mathrm{rstore}_i : \mathrm{REC} \times \mathrm{T}_i \to \mathrm{REC}$ for each i, $1 \leq i \leq n$. The presentation, named $\mathcal{R}(id_1 : \mathrm{T}_1, \ldots, id_n : \mathrm{T}_n)$, or $\mathcal{R}$ for short, is given by the following axioms, where x is a variable of sort REC and v is a variable of sort T_i:

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

$$\forall x, v. \ \mathrm{rselect}_j(\mathrm{rstore}_i(x, v)) \simeq \mathrm{rselect}_j(x) \ \text{ for all } i, j, \ 1 \leq i \neq j \leq n \qquad (2)$$

For the *theory of records with extensionality*, the presentation, named $\mathcal{R}^e$, includes also the following axiom, that states that two records are equal if all their fields are:

$$\forall x, y. \ (\bigwedge_{i=1}^{n} \mathrm{rselect}_i(x) \simeq \mathrm{rselect}_i(y) \supset x \simeq y), \qquad (3)$$

where x and y are variables of sort REC. $\mathcal{R}$ and $\mathcal{R}^e$ are Horn theories, and therefore they are convex. We begin with $\mathcal{R}$-reduction, that allows us to reduce $\mathcal{R}^e$-satisfiability to $\mathcal{R}$-satisfiability:

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 and a literal $L = l \not\simeq r \in S$, such that l and r are terms of sort REC, $\mathcal{R}$-*reduction* first replaces L by the clause

$$C_L = \bigvee_{i=1}^{n} \mathrm{rselect}_i(l) \not\simeq \mathrm{rselect}_i(r)$$

that is the resolvent of L and the clausal form of (3). Thus, if $S = S_1 \uplus S_2$, where S_2 contains the literals $l \not\simeq r$ with l and r of sort REC and S_1 all the other literals, S is replaced by $S_1 \cup \{C_L : L \in S_2\}$. Then, this set of clauses is reduced into disjunctive normal form, yielding a disjunction of $\mathcal{R}$-reduced sets of literals. Let $Red_{\mathcal{R}}(S)$ denote the class of $\mathcal{R}$-reduced sets thus obtained.

LEMMA 1. *Given a set of ground $\mathcal{R}$-literals S, $\mathcal{R}^e \cup S$ is satisfiable if and only if $\mathcal{R} \cup Q$ is, for some $Q \in Red_{\mathcal{R}}(S)$.*

PROOF.
($\Leftarrow$) Let Γ be a many-sorted model of $\mathcal{R} \cup Q$. The claim is that there exists an interpretation Γ' that satisfies $\mathcal{R}^e \cup S$. The only non-trivial part is to show that Γ' satisfies the extensionality axiom of $\mathcal{R}^e$, because in order to satisfy extensionality Γ' needs to interpret the equality predicate $\simeq$ also on records, whereas Γ does not. To simplify notation, let $\sim_\Gamma$ stand for $[\simeq]_\Gamma$ and $\sim_{\Gamma'}$ stand for $[\simeq]_{\Gamma'}$. Then let Γ' be the interpretation that is identical to Γ, except that $\sim_{\Gamma'}$ is defined as follows:

—for all $\hat{a}, \hat{b} \in [\text{REC}]_\Gamma$, $\hat{a} \sim_{\Gamma'} \hat{b}$ if and only if $[\text{rselect}_i]_\Gamma(\hat{a}) \sim_\Gamma [\text{rselect}_i]_\Gamma(\hat{b})$ for all i, $1 \leq i \leq n$, and
—for all $\hat{a}, \hat{b} \in [\text{T}_i]_\Gamma$, $\hat{a} \sim_{\Gamma'} \hat{b}$ if and only if $\hat{a} \sim_\Gamma \hat{b}$, for all i, $1 \leq i \leq n$.

The relation $\sim_{\Gamma'}$ is clearly an equivalence. To prove that it is a congruence, we only need to show that if $\hat{a} \sim_{\Gamma'} \hat{b}$, then $[\text{rstore}_i]_\Gamma(\hat{a}, \hat{e}) \sim_{\Gamma'} [\text{rstore}_i]_\Gamma(\hat{b}, \hat{e})$ for all i, $1 \leq i \leq n$ and $\hat{e} \in [\text{T}_i]_\Gamma$. By way of contradiction, assume that $\hat{a} \sim_{\Gamma'} \hat{b}$, but $[\text{rstore}_i]_\Gamma(\hat{a}, \hat{e}) \not\sim_{\Gamma'} [\text{rstore}_i]_\Gamma(\hat{b}, \hat{e})$ for some i, $1 \leq i \leq n$, and $\hat{e} \in [\text{T}_i]_\Gamma$. In other words, by definition of $\sim_{\Gamma'}$, it is $[\text{rselect}_k]_\Gamma([\text{rstore}_i]_\Gamma(\hat{a}, \hat{e})) \not\sim_\Gamma [\text{rselect}_k]_\Gamma([\text{rstore}_i]_\Gamma(\hat{b}, \hat{e}))$ for some k, $1 \leq k \leq n$. There are two cases: either $k = i$ or $k \neq i$. If $k = i$, then, since Γ, whence Γ', is a model of axiom (1), it follows that $\hat{e} \not\sim_\Gamma \hat{e}$, a contradiction. If $k \neq i$, since Γ, whence Γ', is a model of axiom (2), it follows that $[\text{rselect}_k]_\Gamma(\hat{a}) \not\sim_\Gamma [\text{rselect}_k]_\Gamma(\hat{b})$, which contradicts the assumption $a \sim_{\Gamma'} b$. Thus, $\sim_{\Gamma'}$ is well defined, and Γ' is a model of $\mathcal{R}^e \cup S$.
($\Rightarrow$) This direction is simple and is omitted for brevity.   □

Termination depends on a case analysis showing that only certain clauses can be generated, and resting on a simple assumption on the CSO:

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

Most orderings can meet this requirement easily: for instance, for RPO's, it is sufficient to assume a precedence where all constant symbols are smaller than all function symbols.

LEMMA 2. *All clauses in the limit S_∞ of the derivation $S_0 \vdash_{\mathcal{SP}} S_1 \ldots S_i \vdash_{\mathcal{SP}} \ldots$ generated by a fair $\mathcal{R}$-good $\mathcal{SP}_\succ$-strategy from $S_0 = \mathcal{R} \cup S$, where S is an $\mathcal{R}$-reduced set of ground flat $\mathcal{R}$-literals, belong to one of the following classes, where r, r' are constants of sort REC, and e, e' are constants of sort T_i for some i, $1 \leq i \leq n$:*

i)   *the empty clause;*
ii)   *the clauses in $\mathcal{R}$:*
        ii.a) $\text{rselect}_i(\text{rstore}_i(x, v)) \simeq v$, for all i, $1 \leq i \leq n$
        ii.b) $\text{rselect}_j(\text{rstore}_i(x, v)) \simeq \text{rselect}_j(x)$, for all i, j, $1 \leq i \neq j \leq n$;
iii)   *ground flat unit clauses of the form:*
        iii.a) $r \simeq r'$,
        iii.b) $e \simeq e'$,
        iii.c) $e \not\simeq e'$,

iii.d) rstore$_i(r, e) \simeq r'$, *for some i, $1 \leq i \leq n$,*
iii.e) rselect$_i(r) \simeq e$, *for some i, $1 \leq i \leq n$;*

iv) rselect$_i(r) \simeq$ rselect$_i(r')$, *for some i, $1 \leq i \leq n$.*

PROOF. We recall that inequalities $r \not\simeq r'$ are not listed in (iii), because S is $\mathcal{R}$-reduced. All clauses in the classes above are unit clauses, and therefore have a unique maximal literal. Since $\succ$ is an $\mathcal{R}$-good CSO, the left side of each literal is maximal (for (iii.a), (iii.b), (iii.c) and (iv) this can be assumed without loss of generality). The proof is by induction on the index i of the sequence $\{S_i\}_i$. For $i = 0$, all clauses in S_0 are in (ii) or (iii). For the inductive case, we assume the claim is true for i and we prove it for $i + 1$. Equational factoring applies to a clause with at least two positive literals, and therefore does not apply to unit clauses. Reflection may apply only to a clause in (iii.c) to yield the empty clause. For binary inferences, we consider each class in turn:

—*Inferences within (ii).* None applies.
—*Inferences within (iii).* The only possible inferences produce ground flat unit clauses in (iii) or the empty clause.
—*Inferences between a clause in (iii) and a clause in (ii).* A superposition of (iii.d), rstore$_i(r, e) \simeq r'$ into (ii.a), rselect$_i($rstore$_i(x, v)) \simeq v$, yields rselect$_i(r') \simeq e$ which is in (iii.e). A superposition of (iii.d), rstore$_i(r, e) \simeq r'$, into (ii.b), rselect$_j($rstore$_i(x, v)) \simeq$ rselect$_j(x)$, yields rselect$_j(r') \simeq$ rselect$_j(r)$ which is in (iv). No other inferences apply.
—*Inferences between a clause in (iv) and a clause in (ii)-(iv).* The only applicable inferences are simplifications between clauses in (iii.a), (iii.e) and (iv) which yield clauses in (iii.e) or (iv). □

Since only finitely many clauses of the kinds enumerated in Lemma 2 can be built from a finite signature, the saturated limit S_∞ is finite, and a fair derivation is bound to terminate:

LEMMA 3. *A fair $\mathcal{R}$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $\mathcal{R} \cup S$, where S is an $\mathcal{R}$-reduced set of ground flat $\mathcal{R}$-literals.*

THEOREM 3.1. *A fair $\mathcal{R}$-good $\mathcal{SP}_\succ$-strategy is a polynomial satisfiability procedure for $\mathcal{R}$ and an exponential satisfiability procedure for $\mathcal{R}^e$.*

PROOF. It follows from Lemmas 1 and 3 that a fair $\mathcal{R}$-good $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{R}$ and $\mathcal{R}^e$. For the complexity, let m be the number of subterms occurring in the input set of literals S. Let h be the number of literals of S in the form $l \not\simeq r$, with l and r terms of sort REC. $Red_\mathcal{R}(S)$ contains n^h sets, where n is a constant, the number of field identifiers in the presentation. Since h is $O(m)$, $Red_\mathcal{R}(S)$ contains $O(n^m)$ sets. Since flattening is a $O(m)$ operation, after flattening there are $O(n^m)$ sets, each with $O(m)$ subterms. For each set, inspection of the types of clauses and inferences allowed by Lemma 3 shows that the number of generated clauses is $O(m^2)$. In other words, the size of the set of clauses during the derivation is bound by a constant k which is $O(m^2)$. Since each inference step takes polynomial time in k, the procedure is polynomial for each set, and therefore for $\mathcal{R}$, and exponential for $\mathcal{R}^e$. □

3.2 The Theory of Integer Offsets

The *theory of integer offsets* is a fragment of the theory of the integers, which is applied in verification (e.g., [Bryant et al. 2002]). Its signature does not assume sorts, or assumes a single sort for the integers, and has two unary function symbols s and p, that represent the successor and predecessor functions, respectively. Its presentation, named $\mathcal{I}$, is given by the following infinite set of sentences, as in, e.g., [Ganzinger et al. 2004]:

$$\forall x.\, \mathrm{s}(\mathrm{p}(x)) \simeq x \tag{4}$$

$$\forall x.\, \mathrm{p}(\mathrm{s}(x)) \simeq x \tag{5}$$

$$\forall x.\, \mathrm{s}^i(x) \not\simeq x \qquad \text{for } i > 0 \tag{6}$$

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 sentences in (6) are called *acyclicity axioms*. For convenience, let $Ac = \{\forall x.\, \mathrm{s}^i(x) \not\simeq x : i > 0\}$ and $Ac(n) = \{\forall x.\, \mathrm{s}^i(x) \not\simeq x : 0 < i \leq n\}$. Like the theory of records, $\mathcal{I}$ is also Horn, and therefore convex.

Remark 1. Axiom (5) implies that s is *injective*:

$$\forall x, y.\ \mathrm{s}(x) \simeq \mathrm{s}(y) \supset x \simeq y \tag{7}$$

Indeed, consider the set $\{\mathrm{p}(\mathrm{s}(x)) \simeq x, \mathrm{s}(a) \simeq \mathrm{s}(b), a \not\simeq b\}$, where $\{\mathrm{s}(a) \simeq \mathrm{s}(b), a \not\simeq b\}$ is the clausal form of the negation of (7). Superposition of $\mathrm{s}(a) \simeq \mathrm{s}(b)$ into $\mathrm{p}(\mathrm{s}(x)) \simeq x$ generates $\mathrm{p}(\mathrm{s}(b)) \simeq a$. Superposition of $\mathrm{p}(\mathrm{s}(b)) \simeq a$ into $\mathrm{p}(\mathrm{s}(x)) \simeq x$ generates $a \simeq b$, that contradicts $a \not\simeq b$.

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 $\mathrm{p}(c) \simeq b$. Negative ground flat literals have the form $c \not\simeq b$, and therefore do not contain p. *$\mathcal{I}$-reduction* consists of replacing every equation $\mathrm{p}(c) \simeq b$ in S by $c \simeq \mathrm{s}(b)$. 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 (4) and (5) can be removed, provided lemma (7) is added:

LEMMA 4. *Let S be a set of ground flat $\mathcal{I}$-literals. $\mathcal{I} \cup S$ is satisfiable if and only if $Ac \cup \{(7)\} \cup Red_{\mathcal{I}}(S)$ is.*

PROOF.
($\Rightarrow$) It follows from Remark 1 and the observation that $\mathcal{I} \cup S \models Red_{\mathcal{I}}(S)$, since $c \simeq \mathrm{s}(b)$ is a logical consequence of $\mathcal{I}$ and $\mathrm{p}(c) \simeq b$, as it can be generated by a superposition of $\mathrm{p}(c) \simeq b$ into axiom $\mathrm{s}(\mathrm{p}(x)) \simeq x$.
($\Leftarrow$) Let Γ be a model of $Ac \cup \{(7)\} \cup Red_{\mathcal{I}}(S)$ and let D be its domain. We build a model Γ' of $\mathcal{I} \cup S$. Γ' interprets all constants in S in the same way as Γ does. The crucial point is that p does not occur in $Ac \cup \{(7)\} \cup Red_{\mathcal{I}}(S)$, so that Γ does not interpret it, whereas Γ' should. Since not all elements of D may have predecessor in D itself, the domain D' of Γ' will be a superset of D, containing as many additional elements as are needed to interpret p. We construct recursively families of sets $\{D_i\}_i$ and functions $\{s_i\}_i$ and $\{p_i\}_i$ in such a way that, at the limit, all elements have predecessor:

—*Base Case.* $i = 0$. Let $D_0 = D$, $s_0 = [s]_\Gamma$ and $p_0 = \emptyset$. By establishing $s_0 = [s]_\Gamma$, the interpretation of s on D is preserved: for all $\hat{d} \in D$, $[s]_\Gamma(\hat{d}) = [s]_{\Gamma'}(\hat{d})$. We start by partitioning D_0 into the subset E_0 of elements that are successors of some other element, and the subset F_0 of those that are not: $E_0 = \{\hat{e} \colon s_0(\hat{d}) = \hat{e} \text{ for some } \hat{d} \in D_0\}$ and $F_0 = D_0 \setminus E_0$. For all $\hat{e} \in E_0$, we define $p_1(\hat{e}) = \hat{d}$ such that $s_0(\hat{d}) = \hat{e}$: such a $\hat{d}$ exists by definition of E_0 and it is unique because $\Gamma \models (7)$. Thus, p_1 is well-defined on E_0. Next, we define p_1 on F_0. Let D'_0 be a set disjoint from D_0 and let $\eta_0 \colon F_0 \to D'_0$ be a bijection: intuitively, η_0 maps each element of F_0 to its predecessor that is missing in D_0. Indeed, for all $\hat{d} \in F_0$, we define $p_1(\hat{d}) = \eta_0(\hat{d})$. Then we define s_1: for all $\hat{e} \in D_0$, $s_1(\hat{e}) = s_0(\hat{e})$ and for all $\hat{e} \in D'_0$, $s_1(\hat{e}) = \hat{d}$, where $\hat{d}$ is the element such that $\eta_0(\hat{d}) = \hat{e}$. Establishing that $D_1 = D_0 \uplus D'_0$ closes the base case.

—*Recursive Case.* Suppose that for $i \geq 1$, we have a $D_{i-1} \subseteq D_i$, where we have defined s_i and p_i in such way that for all $\hat{d} \in D_{i-1}$, there exists an $\hat{e} \in D_i$ such that $s_i(\hat{d}) = \hat{e}$ and $p_i(\hat{e}) = \hat{d}$. On the other hand, there may be elements in D_i that are not successors of any other element, so that their predecessor is not defined. Thus, let $E_i = \{\hat{e} \colon s_i(\hat{d}) = \hat{e} \text{ for some } \hat{d} \in D_i\}$ and $F_i = D_i \setminus E_i$. Let D'_i be a set of new elements and $\eta_i \colon F_i \to D'_i$ a bijection. Then, let $D_{i+1} = D_i \uplus D'_i$. For p_{i+1}, for all $\hat{e} \in E_i$, $p_{i+1}(\hat{e}) = \hat{d}$ such that $s_i(\hat{d}) = \hat{e}$ and for all $\hat{d} \in F_i$, $p_{i+1}(\hat{d}) = \eta_i(\hat{d})$. For s_{i+1}, for all $\hat{e} \in D_i$, $s_{i+1}(\hat{e}) = s_i(\hat{e})$ and for all $\hat{e} \in D'_i$, $s_{i+1}(\hat{e}) = \hat{d}$, where $\hat{d}$ is the element such that $\eta_i(\hat{d}) = \hat{e}$.

Then we define $D' = \bigcup_i D_i$, $[s]_{\Gamma'} = \bigcup_i s_i$ and $[p]_{\Gamma'} = \bigcup_i p_i$. We show that $\Gamma' \models \mathcal{I} \cup S$. Axioms (4) and (5) are satisfied, since, by construction, for all $\hat{e} \in D'$, $[p]_{\Gamma'}(\hat{e}) = \hat{d}$ and $[s]_{\Gamma'}(\hat{d}) = \hat{e}$. The only equations in $S \setminus Red_\mathcal{I}(S)$ are the $p(c) \simeq b$ for which $s(b) \simeq c \in Red_\mathcal{I}(S)$. Let $[b]_{\Gamma'} = \hat{d}$ and $[c]_{\Gamma'} = \hat{e}$: since $\Gamma \models s(b) \simeq c$, it follows that $\hat{e} \in E_0$, and $\Gamma' \models p(c) \simeq b$.   □

EXAMPLE 1. *Let* $S = \{s(c) \simeq c'\}$ *and let* Γ *be the model with domain* $\mathbb{N}$, *such that* c *is interpreted as* 0 *and* c' *as* 1. *Then* $D_0 = \mathbb{N}$, $E_0 = \mathbb{N} \setminus \{0\}$ *and* $F_0 = \{0\}$. *At the first step of the construction, we can take* $D'_0 = \{-1\}$, *and have* $p_1(0) = -1$ *and* $s_1(-1) = 0$. *Then* $F_1 = \{-1\}$, *and we can take* $D'_1 = \{-2\}$, *and so on. At the limit,* D' *is the set* $\mathbb{Z}$ *of the integers.*

The next step is to bound the number of axioms in Ac needed to solve the problem. The intuition is that the bound will be given by the number of elements whose successor is determined by a constraint $s(c) \simeq c'$ in S. Such a constraint establishes that, in any model Γ of S, the successor of $[c]_\Gamma$ must be $[c']_\Gamma$. We call s-*free* an element that is not thus constrained:

DEFINITION 6. *Let* S *be a satisfiable* $\mathcal{I}$-*reduced set of ground flat* $\mathcal{I}$-*literals,* Γ *be a model of* S *with domain* D *and* C_S *be the set of constants* $C_S = \{c \colon s(c) \simeq c' \in S\}$. *An element* $\hat{d} \in D$ *is* s-free *in* S *for* Γ, *if for no* $c \in C_S$, *it is the case that* $[c]_\Gamma = \hat{d}$.

We shall see that it is sufficient to consider $Ac(n)$, where n is the cardinality of C_S.

EXAMPLE 2. *If* $S = \{s(c_1) \simeq c_2, s(c_1) \simeq c_3, c_2 \simeq c_3\}$, *then* $C_S = \{c_1\}$ *and* $|C_S| = 1$. *In the worst case, however, all occurrences of* s *apply to different constants, so*

that $|C_S|$ is the number of occurrences of s *in S.*

We begin with a notion of s-*path* that mirrors the paths in the graph $(D, [s]_\Gamma)$ defined by an interpretation Γ:

DEFINITION 7. *Let S be a satisfiable $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals and let Γ be a model of S with domain D. For all $m \geq 2$, a tuple $\langle \hat{d}_1, s, \hat{d}_2, s, \dots \hat{d}_m, s \rangle$ is an* s-path *of length m if*

(1) $\forall i, j, 1 \leq i \neq j \leq m, \hat{d}_i \neq \hat{d}_j$ and

(2) $\forall i, 1 \leq i < m, \hat{d}_{i+1} = [s]_\Gamma(\hat{d}_i)$.

It is an s-cycle *if, additionally, $[s]_\Gamma(\hat{d}_m) = \hat{d}_1$.*

It is clear to see that $\Gamma \models Ac(n)$ if and only if Γ has no s-*cycles* of length smaller or equal to n.

LEMMA 5. *Let S be an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with $|C_S| = n$. If there is an* s-*path p of length $m > n$ in a model Γ of S, then p includes an element that is* s-*free in S.*

PROOF. By way of contradiction, assume that no element in $p = \langle \hat{d}_1, s, \hat{d}_2, s, \dots \hat{d}_m, s \rangle$ is s-free in S. By Definition 6, this means that for all j, $1 \leq j \leq m$, there is a constant $c_j \in C_S$ such that $[c_j]_\Gamma = \hat{d}_j$. Since by Definition 7 all elements in an s-path are distinct, it follows that C_S should contain at least m elements, or $n \geq m$, which contradicts the hypothesis that $m > n$.   □

LEMMA 6. *Let S be an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with $|C_S| = l$. For $n \geq l$, if $Ac(n) \cup \{(7)\} \cup S$ is satisfiable then $Ac(n+1) \cup \{(7)\} \cup S$ is.*

PROOF. Let $\Gamma = \langle D, J \rangle$ be a model of $Ac(n) \cup \{(7)\} \cup S$. Γ has no s-cycles of length smaller or equal to n. We build a model Γ' with no s-cycles of length smaller or equal to $n + 1$. Let $P = \{p : p \text{ is an s--cycle of length } n+1\}$. If $P = \emptyset$, $\Gamma \models Ac(n+1)$ and Γ' is Γ itself. If $P \neq \emptyset$, there is some $p \in P$. Since $n + 1 > l$, by Lemma 5, there is some $\hat{d}$ in p that is s-free in S for Γ. Let $E_p = \{\hat{e}_j : j \geq 0\}$ be a set disjoint from D, and let J_p be the interpretation function that is identical to J, except that $J_p(s)(\hat{d}) = \hat{e}_0$ and $J_p(s)(\hat{e}_j) = \hat{e}_{j+1}$ for all $j \geq 0$. By extending D into $D \cup E_p$ and extending J into J_p, we obtain a model where the s-cycle p has been broken. By repeating this transformation for all $p \in P$, we obtain the Γ' sought for. Indeed, $\Gamma' \models Ac(n+1)$, because it has no s-cycles of length smaller or equal to $n + 1$. $\Gamma' \models S$, because it interprets constants in the same way as Γ, and for each $s(c) \simeq e \in S$, $[c]_\Gamma$ is not s-free in S, which means $[s(c)]_{\Gamma'} = [s(c)]_\Gamma$. To see that $\Gamma' \models (7)$, let $\hat{d}$ and $\hat{d}'$ be two elements such that $[s]_{\Gamma'}(\hat{d}) = [s]_{\Gamma'}(\hat{d}')$. If $[s]_{\Gamma'}(\hat{d}) \in D$, then $[s]_{\Gamma'}(\hat{d}) = [s]_\Gamma(\hat{d})$, so that $\hat{d} = \hat{d}'$, because $\Gamma \models (7)$. If $[s]_{\Gamma'}(\hat{d}) \notin D$, then $[s]_{\Gamma'}(\hat{d}) = \hat{e}$ for some $\hat{e}$ introduced by the above construction, so that $\hat{d}$ is the unique element whose successor is $\hat{e}$, and $\hat{d} = \hat{d}'$.   □

By compactness, we have the following:

COROLLARY 1. *Let S be an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with $|C_S| = n$. $Ac \cup \{(7)\} \cup S$ is satisfiable if and only if $Ac(n) \cup \{(7)\} \cup S$ is.*

PROOF. The "only if" direction is trivial and for the "if" direction induction using Lemma 6 shows that for all $k \geq 0$, if $Ac(n) \cup \{(7)\} \cup S$ is satisfiable, then so is $Ac(n+k) \cup \{(7)\} \cup S$.   □

DEFINITION 8. *A CSO $\succ$ is $\mathcal{I}$-good if $t \succ c$ for all constants c and all terms t whose root symbol is* s.

For instance, a precedence where all constant symbols are smaller than s will yield an $\mathcal{I}$-good RPO.

LEMMA 7. *All clauses in the limit S_∞ of the derivation $S_0 \vdash_{\mathcal{SP}} S_1 \ldots S_i \vdash_{\mathcal{SP}} \ldots$ generated by a fair $\mathcal{I}$-good $\mathcal{SP}_\succ$-strategy from $S_0 = Ac(n) \cup \{(7)\} \cup S$, where S is an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with $|C_S| = n$, belong to one of the following classes, where $b_1, \ldots b_k$, $d_1, \ldots d_k$, c, d and e are constants $(k \geq 0)$:*

i)   *the empty clause;*
ii) *the clauses in $Ac(n) \cup \{(7)\}$:*
     ii.a) $s^i(x) \not\simeq x$, for all i, $0 < i \leq n$,
     ii.b) $s(x) \not\simeq s(y) \vee x \simeq y$;
iii) *ground flat unit clauses of the form:*
     iii.a) $c \simeq d$,
     iii.b) $c \not\simeq d$,
     iii.c) $s(c) \simeq d$;
iv) *other clauses of the following form:*
     iv.a) $s(x) \not\simeq d \vee x \simeq c \vee \bigvee_{i=1}^{k} d_i \not\simeq b_i$,
     iv.b) $c \simeq e \vee \bigvee_{i=1}^{k} d_i \not\simeq b_i$,
     iv.c) $\bigvee_{i=1}^{k} d_i \not\simeq b_i$,
     iv.d) $s(c) \simeq e \vee \bigvee_{i=1}^{k} d_i \not\simeq b_i$,
     iv.e) $s^j(c) \not\simeq e \vee \bigvee_{i=1}^{k} d_i \not\simeq b_i$, $1 \leq j \leq n-1$.

PROOF. Since $\succ$ is a CSO, the first literal is the only maximal literal in (ii.b). Since it is $\mathcal{I}$-good, the first literal is the only maximal literal in (iv.a), (iv.d) and (iv.e). For the same reason, the left-hand side is maximal in the maximal literals in (iii.c), (iv.a), (iv.d) and (iv.e). The proof is by induction on the sequence $\{S_i\}_i$. For the base case, all clauses in S_0 are in (ii) or (iii). For the inductive case, we consider all possible inferences, excluding upfront equational factoring, which applies to a clause with at least two positive literals, and therefore does not apply to Horn clauses.

—*Inferences within (ii).* Reflection applies to (ii.b) to generate $x \simeq x$, that gets deleted by deletion.

—*Inferences within (iii).* The only possible inferences produce ground flat unit clauses in (iii) or the empty clause.

—*Inferences between a clause in (iii) and a clause in (ii).* A paramodulation of an equality of kind (iii.c) into an inequality of type (ii.a) yields inequalities $s^{i-1}(d) \not\simeq c$, $1 \leq i \leq n$, that are in (iv.e) with $k = 0$ (for $i > 1$) or (iii.b) (for $i = 1$). A paramodulation of a (iii.c) equality into (ii.b) yields $s(x) \not\simeq d \vee x \simeq c$ which is in (iv.a) with $k = 0$.

—*Inferences between a clause in (iv) and a clause in (ii).* A paramodulation of a clause in (iv.d) into (ii.a) produces a clause in (iv.c) or (iv.e), and a paramodulation of a clause in (iv.d) into (ii.b) produces a clause in (iv.a).

—*Inferences between a clause in (iv) and a clause in (iii).* A simplification of a clause in (iv) by an equality in (iii.a) or (iii.c) generates another clause in (iv). Paramodulating an equality of kind (iii.c) into a (iv.a) clause yields a (iv.b) clause. Similarly, superposing a (iii.c) unit with a (iv.d) clause gives a (iv.b) clause. The only possible remaining inferences between a clause in (iv) and one in (iii) are paramodulations or superpositions of a (iv.b) clause into clauses in (iii), that add clauses in (iv.b), (iv.c) and (iv.d).

—*Inferences within (iv).* Reflection applies to clauses in (iv.b) and (iv.c) to yield clauses in (iv.b) or (iii.a) and (iv.c) or (iii.b), respectively. A paramodulation or superposition of a (iv.b) clause into a (iv.b), (iv.c), (iv.d) or (iv.e) clause generates clauses also in (iv.b), (iv.c), (iv.d) or (iv.e), respectively. A superposition of a clause of kind (iv.d) into a (iv.a) clause gives a clause in (iv.b). A superposition between two (iv.d) clauses adds a (iv.b) clause. A paramodulation of a clause of type (iv.d) into a (iv.e) clause yields a clause in (iv.e) or (iv.c). □

Given a finite signature, only finitely many clauses of the types allowed by Lemma 7 can be formed. Thus, we have:

LEMMA 8. *A fair $\mathcal{I}$-good $\mathcal{SP}_{\succ}$-strategy is guaranteed to terminate when applied to $Ac(n) \cup \{(7)\} \cup S$, where S is an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}$-literals with $|C_S| = n$.*

THEOREM 3.2. *A fair $\mathcal{I}$-good $\mathcal{SP}_{\succ}$-strategy is an exponential satisfiability procedure for $\mathcal{I}$.*

PROOF. The main result follows from Lemma 4, Corollary 1 and Lemma 8. For the complexity, let m be the number of subterms occurring in the input set of literals S. $Red_{\mathcal{I}}(S)$ has the same number of subterms as S, since $\mathcal{I}$-reduction replaces literals of the form $p(c) \simeq b$ by literals of the form $c \simeq s(b)$. Flattening is $O(m)$. The number n of retained acyclicity axioms, according to Lemma 8, is also $O(m)$, since in the worst case it is given by the number of occurrences of s in S. By the proof of Lemma 7, at most $h = O(m^2)$ distinct literals and at most $O(2^h)$ clauses can be generated. Thus, the size of the database of clauses during the derivation is bound by a constant k which is $O(2^h)$. Since each inference step takes polynomial time in k, the overall procedure is $O(2^{m^2})$. □

COROLLARY 2. *A fair $\mathcal{SP}_{\succ}$-strategy is a polynomial satisfiability procedure for the theory presented by the set of* acyclicity axioms Ac.

PROOF. The proof of Lemma 7 shows that if the input includes only $Ac(n) \cup S$, the only generated clauses are finitely many ground flat unit clauses from S (inferences within (iii)), and finitely many equalities in the form $s^{i-1}(d) \not\simeq c$, for $1 \leq i \leq n$, by paramodulation of equalities $s(c) \simeq d \in S$ into axioms $s^i(x) \not\simeq x$ (inferences between a clause in (iii) and a clause in (ii)). It follows that the number of clauses generated during the derivation is $O(m^2)$, where m is the number of subterms occurring in the input set of literals. The size of the database of clauses during the derivation

is bound by a constant k which is $O(m^2)$, and since each inference step takes polynomial time in k, a polynomial procedure results.   □

3.3   The Theory of Integer Offsets Modulo

The above treatment extends to the *theory of integer offsets modulo*, which makes possible to describe data structures with indices ranging over the integers modulo k, such as circular queues. A presentation for this theory, named $\mathcal{I}_k$, is obtained from $\mathcal{I}$ by replacing Ac with the following k axioms

$$\forall x.\, \mathrm{s}^i(x) \not\simeq x \quad \text{for } 1 \leq i \leq k-1 \tag{8}$$

$$\forall x.\, \mathrm{s}^k(x) \simeq x, \tag{9}$$

where $k > 1$. $\mathcal{I}_k$ also is Horn and therefore convex.

Definition 5 and Lemma 4 apply also to $\mathcal{I}_k$, whereas Lemma 6 is no longer necessary, because $\mathcal{I}_k$ is finite to begin with. Termination is guaranteed by the following lemma, where $C(k) = \{\forall x.\, \mathrm{s}^k(x) \simeq x\}$:

LEMMA 9. *A fair $\mathcal{I}$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $Ac(k-1) \cup C(k) \cup \{(7)\} \cup S$, where S is an $\mathcal{I}$-reduced set of ground flat $\mathcal{I}_k$-literals.*

PROOF. The proof of termination rests on the proof of Lemma 7, with $n = k - 1$ and the following additional cases to account for the presence of $C(k)$. As far as inferences between axioms are concerned (i.e., within group (ii) in the proof of Lemma 7), $C(k)$ does not introduce any, because $\mathrm{s}^k(x) \simeq x$ cannot paramodulate into $\mathrm{s}^i(x) \not\simeq x$, since $i < k$, and cannot paramodulate into (7), since $k > 1$. For inferences between axioms and literals in S (i.e., groups (ii) and (iii) in the proof of Lemma 7), the presence of $C(k)$ introduces superpositions of literals $\mathrm{s}(c) \simeq d \in S$ into $\mathrm{s}^k(x) \simeq x$, generating $\mathrm{s}^{i-1}(d) \simeq c$ for $1 \leq i \leq k$. If we use j in place of $i-1$ and n in place of $k-1$, we have $\mathrm{s}^j(d) \simeq c$ for $0 \leq j \leq n$. Excluding the cases $j = 0$ and $j = 1$ that are already covered by classes (iii.a) and (iii.c) of Lemma 7, we have an additional class of clauses, with respect to those of Lemma 7:

v) $\mathrm{s}^j(d) \simeq c$ for $2 \leq j \leq n$.

Thus, we only need to check the inferences induced by clauses of type (v). There are only two possibilities. Paramodulations of equalities in (v) into inequalities in (ii.a) gives more clauses in (iv.e) with $k = 0$. Paramodulations of equalities in (v) into clauses in (iv.e) gives more clauses in (iv.c) or (iv.e). Since only finitely many clauses of types (i-v) can be formed from a finite signature, termination follows.   □

THEOREM 3.3. *A fair $\mathcal{I}$-good $\mathcal{SP}_\succ$-strategy is an exponential satisfiability procedure for $\mathcal{I}_k$.*

PROOF. It follows the same pattern of the proof of Theorem 3.2, with Lemma 9 in place of Lemma 8.   □

Alternatively, since $\mathcal{I}_k$ is finite, it is possible to omit $\mathcal{I}$-reduction and show termination on the original problem format. The advantage is that it is not necessary to include the injectivity property (7), so that the resulting procedure is polynomial. Furthermore, abandoning the framework of $\mathcal{I}$-reduction, that was conceived to handle the infinite presentation of $\mathcal{I}$, one can add axioms for p that are dual of

(8) and (9), resulting in the presentation $\mathcal{I}'_k$ made of (4), (5), (8), (9) and:

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

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

with $k > 1$. $\mathcal{I}'_k$ is also Horn and therefore convex.

DEFINITION 9. *A CSO $\succ$ is $\mathcal{I}'_k$-good if $t \succ c$ for all ground compound terms t and constants c.*

LEMMA 10. *A fair $\mathcal{I}'_k$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $\mathcal{I}'_k \cup S$, where S is a set of ground flat $\mathcal{I}'_k$-literals.*

PROOF. Termination follows from the general observation that the only persistent clauses, that can be generated by $\mathcal{SP}_\succ$ from $\mathcal{I}'_k \cup S$, are unit clauses $l \bowtie r$, such that l and r are terms in the form $\mathrm{s}^j(u)$ or $\mathrm{p}^j(u)$, where $0 \leq j \leq k-1$ and u is either a constant or a variable. Indeed, if a term in this form with $j \geq k$ were generated, it would be simplified by axioms (9) $\mathrm{s}^k(x) \simeq x$ or (11) $\mathrm{p}^k(x) \simeq x$. Similarly, if a term where s is applied over p or vice-versa were generated, it would be simplified by axioms (4) $\mathrm{s}(\mathrm{p}(x)) \simeq x$ or (5) $\mathrm{p}(\mathrm{s}(x)) \simeq x$. Given a finite number of constants and with variants removed by subsumption, the bound on term depth represented by k implies that there are only finitely many such clauses.   □

THEOREM 3.4. *A fair $\mathcal{I}'_k$-good $\mathcal{SP}_\succ$-strategy is a polynomial satisfiability procedure for $\mathcal{I}'_k$.*

PROOF. Termination was established in Lemma 10. To see that the procedure is polynomial, let m be the number of subterms in the input set of ground literals. After flattening, we have $O(m)$ subterms, and since $\mathcal{I}'_k$ has $O(k)$ subterms, the input to the $\mathcal{SP}_\succ$-strategy has $O(m+k)$ subterms. By the proof of Lemma 10, only unit clauses are generated, so that their number is $O((m+k)^2)$. Since the size of the database of clauses during the derivation is bound by a constant h which is $O((m+k)^2)$, and each inference takes polynomial time in h, the overall procedure is polynomial.   □

3.4   The Theory of Possibly Empty Lists

Different presentations were proposed for a theory of *lists*. A "convex theory of cons, car and cdr," was studied by [Shostak 1984], and therefore it is named $\mathcal{L}_{Sh}$. Its signature contains cons, car and cdr, and its axioms are:

$$\forall x, y.\ \mathrm{car}(\mathrm{cons}(x, y)) \simeq x \tag{12}$$

$$\forall x, y.\ \mathrm{cdr}(\mathrm{cons}(x, y)) \simeq y \tag{13}$$

$$\forall y.\ \mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y \tag{14}$$

The presentation adopted by [Nelson and Oppen 1980], hence called $\mathcal{L}_{NO}$, adds the predicate symbol atom to the signature, and the axioms

$$\forall x, y.\ \neg\,\mathrm{atom}(\mathrm{cons}(x, y)) \tag{15}$$

$$\forall y.\ \neg\,\mathrm{atom}(y) \supset \mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y \tag{16}$$

to axioms (12) and (13).

A third presentation also appeared in [Nelson and Oppen 1980], but was not used in their congruence-closure-based algorithm. Its signature features the constant symbol nil, together with cons, car and cdr, but not atom. This presentation, that we call $\mathcal{L}$, adds to (12) and (13) the following four axioms:

$$\forall x, y. \ \mathrm{cons}(x, y) \not\simeq \mathrm{nil} \tag{17}$$

$$\forall y. \ y \not\simeq \mathrm{nil} \supset \mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y \tag{18}$$

$$\mathrm{car}(\mathrm{nil}) \simeq \mathrm{nil} \tag{19}$$

$$\mathrm{cdr}(\mathrm{nil}) \simeq \mathrm{nil} \tag{20}$$

$\mathcal{L}$ is not convex, because $y \simeq \mathrm{nil} \vee \mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y$ is in $Th\,\mathcal{L}$, but neither disjunct is.

Unlike the presentation of records given earlier, and that of arrays, that will be given in the next section, these presentations of lists are unsorted, or lists and their elements belong to the same sort. This is desirable because it allows lists of lists. Also, neither $\mathcal{L}_{Sh}$ nor $\mathcal{L}_{NO}$ nor $\mathcal{L}$ exclude cyclic lists (i.e., a model of anyone of these presentations can satisfy $\mathrm{car}(x) \simeq x$). The rewriting approach was already applied to both $\mathcal{L}_{Sh}$ and $\mathcal{L}_{NO}$ in [Armando et al. 2003]. The following analysis shows that it applies to $\mathcal{L}$ as well.

DEFINITION 10. *A CSO $\succ$ is $\mathcal{L}$-good if (1) $t \succ c$ for all ground compound terms t and constants c, (2) $t \succ$ nil for all terms t whose root symbol is* cons.

It is sufficient to impose a precedence $>$, such that function symbols are greater than constant symbols, including cons $>$ nil, to make an RPO, or a KBO with a simple weighting scheme (e.g., weight given by arity), $\mathcal{L}$-good. No $\mathcal{L}$-reduction is needed, and the key result is the following:

LEMMA 11. *All clauses in the limit S_∞ of the derivation $S_0 \vdash_{\mathcal{SP}} S_1 \ldots S_i \vdash_{\mathcal{SP}} \ldots$ generated by a fair $\mathcal{L}$-good $\mathcal{SP}_\succ$-strategy from $S_0 = \mathcal{L} \cup S$, where S is a set of ground flat $\mathcal{L}$-literals, belong to one of the following classes, where c_i and d_i for all i, $1 \leq i \leq n$, and e_1, e_2, e_3 are constants (constants include* nil*):*

i)   *the empty clause;*

ii) *the clauses in $\mathcal{L}$:*
   ii.a) $\mathrm{car}(\mathrm{cons}(x, y)) \simeq x$,
   ii.b) $\mathrm{cdr}(\mathrm{cons}(x, y)) \simeq y$,
   ii.c) $\mathrm{cons}(x, y) \not\simeq \mathrm{nil}$,
   ii.d) $\mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y \vee y \simeq \mathrm{nil}$,
   ii.e) $\mathrm{car}(\mathrm{nil}) \simeq \mathrm{nil}$,
   ii.f) $\mathrm{cdr}(\mathrm{nil}) \simeq \mathrm{nil}$;

iii) *ground flat unit clauses of the form:*
   iii.a) $c_1 \simeq c_2$,
   iii.b) $c_1 \not\simeq c_2$,
   iii.c) $\mathrm{car}(c_1) \simeq c_2$,
   iii.d) $\mathrm{cdr}(c_1) \simeq c_2$,
   iii.e) $\mathrm{cons}(c_1, c_2) \simeq c_3$;

iv) *non-unit clauses of the following form:*

$iv.a)$ $\mathrm{cons}(e_1, \mathrm{cdr}(e_2)) \simeq e_3 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.b)$ $\mathrm{cons}(\mathrm{car}(e_1), e_2) \simeq e_3 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.c)$ $\mathrm{cons}(\mathrm{car}(e_1), \mathrm{cdr}(e_2)) \simeq e_3 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.d)$ $\mathrm{cons}(e_1, e_2) \simeq e_3 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.e)$ $\mathrm{car}(e_1) \simeq \mathrm{car}(e_2) \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.f)$ $\mathrm{cdr}(e_1) \simeq \mathrm{cdr}(e_2) \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.g)$ $\mathrm{car}(e_1) \simeq e_2 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.h)$ $\mathrm{cdr}(e_1) \simeq e_2 \vee \bigvee_{i=1}^n c_i \bowtie d_i,$

$iv.i)$ $\bigvee_{i=1}^n c_i \bowtie d_i.$

PROOF. Since $\succ$ is an $\mathcal{L}$-good CSO, each clause in the above classes has a unique maximal literal, which is the first one in the above listing (up to a permutation of indices for (iv.i)). Furthermore, the left side in each maximal literal is maximal (for (iii.a), (iii.b), (iv.e), (iv.f), (iv.i) this can be assumed without loss of generality). The proof is by induction on the sequence $\{S_i\}_i$. For the base case, input clauses are in (ii) or (iii). For the inductive case, we consider all classes in order:

—*Inferences within (ii).* All inferences between axioms generate clauses that get deleted. Superposition of (ii.a) into (ii.d) generates $\mathrm{cons}(x, \mathrm{cdr}(\mathrm{cons}(x, y))) \simeq \mathrm{cons}(x, y) \vee \mathrm{cons}(x, y) \simeq$ nil, which is simplified by (ii.b) to $\mathrm{cons}(x, y) \simeq \mathrm{cons}(x, y) \vee \mathrm{cons}(x, y) \simeq$ nil, which is deleted. Superposition of (ii.d) into (ii.a) produces $\mathrm{car}(y) \simeq \mathrm{car}(y) \vee y \simeq$ nil which is deleted. Superposition of (ii.b) into (ii.d) yields $\mathrm{cons}(\mathrm{car}(\mathrm{cons}(x, y)), y) \simeq \mathrm{cons}(x, y) \vee \mathrm{cons}(x, y) \simeq$ nil, whose simplification by (ii.a) gives $\mathrm{cons}(x, y) \simeq \mathrm{cons}(x, y) \vee \mathrm{cons}(x, y) \simeq$ nil, which is deleted. Superposition of (ii.d) into (ii.b) generates $\mathrm{cdr}(y) \simeq \mathrm{cdr}(y) \vee y \simeq$ nil which gets deleted. Paramodulation of (ii.d) into (ii.c) produces the tautology $y \simeq$ nil $\vee\, y \not\simeq$ nil, which is eliminated by a step of reflection followed by one of deletion. Superposition of (ii.e) into (ii.d) yields $\mathrm{cons}(\mathrm{nil}, \mathrm{cdr}(\mathrm{nil})) \simeq \mathrm{nil} \vee \mathrm{nil} \simeq$ nil which is deleted. Similarly, superposition of (ii.f) into (ii.d) yields $\mathrm{cons}(\mathrm{car}(\mathrm{nil}), \mathrm{nil}) \simeq \mathrm{nil} \vee \mathrm{nil} \simeq$ nil which is also deleted, and no other inferences apply among axioms.

—*Inferences within (iii).* Inferences on the maximal terms in (iii) can generate only more ground flat unit clauses like those in (iii) or the empty clause.

—*Inferences between a clause in (iii) and a clause in (ii).* Inferences between an axiom and a ground flat unit clause generate either more ground flat unit clauses or non-unit clauses in the classes (iv.a) and (iv.b). Indeed, the only applicable inferences are: superposition of a unit of kind (iii.c) into (ii.d), which gives a clause in the form $\mathrm{cons}(c_2, \mathrm{cdr}(c_1)) \simeq c_1 \vee c_1 \simeq$ nil of class (iv.a); superposition of a unit of kind (iii.d) into (ii.d), which gives a clause in the form $\mathrm{cons}(\mathrm{car}(c_1), c_2) \simeq c_1 \vee c_1 \simeq$ nil of class (iv.b); superposition of a unit of kind (iii.e) into (ii.a), (ii.b), (ii.c), which generates unit clauses in (iii.c), (iii.d) and (iii.b), respectively.

—*Inferences between a clause in (iv) and a clause in (ii).* We consider the clauses in (ii) in order. For (ii.a): superposing a clause of kind (iv.a) or (iv.d) into (ii.a) generates $\mathrm{car}(e_3) \simeq e_1 \vee \bigvee_{i=1}^n c_i \bowtie d_i$, that is in (iv.g); superposing a clause of kind (iv.b) or (iv.c) into (ii.a) generates $\mathrm{car}(e_1) \simeq \mathrm{car}(e_3) \vee \bigvee_{i=1}^n c_i \bowtie d_i$ that is in (iv.e). For (ii.b): superposing a clause of kind (iv.a) or (iv.c) into (ii.b) generates $\mathrm{cdr}(e_2) \simeq \mathrm{cdr}(e_3) \vee \bigvee_{i=1}^n c_i \bowtie d_i$ that is in (iv.f); superposing a clause of kind (iv.b) or (iv.d) into (ii.b) generates $\mathrm{cdr}(e_3) \simeq e_2 \vee \bigvee_{i=1}^n c_i \bowtie d_i$, that

is in (iv.h). Paramodulating clauses of classes (iv.a), (iv.b), (iv.c) and (iv.d) into (ii.c) gives clauses in class (iv.i). For (ii.d): a superposition of (ii.d) into (iv.c) or (iv.c) into (ii.d) yields a clause in (iv.i); a superposition of (iv.e) or (iv.f) into (ii.d) produces $\mathrm{cons}(\mathrm{car}(e_2), \mathrm{cdr}(e_1)) \simeq e_1 \vee e_1 \simeq \mathrm{nil} \vee \bigvee_{i=1}^{n} c_i \bowtie d_i$ or $\mathrm{cons}(\mathrm{car}(e_1), \mathrm{cdr}(e_2)) \simeq e_1 \vee e_1 \simeq \mathrm{nil} \vee \bigvee_{i=1}^{n} c_i \bowtie d_i$ that are in (iv.c); a superposition of (iv.g) into (ii.d) produces $\mathrm{cons}(e_2, \mathrm{cdr}(e_1)) \simeq e_1 \vee e_1 \simeq \mathrm{nil} \vee \bigvee_{i=1}^{n} c_i \bowtie d_i$ that is in (iv.a); a superposition of (iv.h) into (ii.d) produces $\mathrm{cons}(\mathrm{car}(e_1), e_2) \simeq e_1 \vee e_1 \simeq \mathrm{nil} \vee \bigvee_{i=1}^{n} c_i \bowtie d_i$ that is in (iv.b). Clause (ii.e) can simplify clauses in (iv.b), (iv.c), (iv.e), (iv.g), to clauses in (iv.d), (iv.a), (iv.g), (iv.i), respectively. Clause (ii.f) can simplify clauses in (iv.a), (iv.c), (iv.f), (iv.h), to clauses in (iv.d), (iv.b), (iv.h), (iv.i), respectively. No other inferences apply.

—*Inferences between a clause in (iv) and a clause in (iii).* The only possible expansion inference here is a paramodulation of a clause in (iv.i) into a clause in (iii.b), which generates another clause in (iv.i). All other possible steps are simplifications, where an equality of class (iii) reduces a clause in (iv) to another clause in (iv).

—*Inferences within (iv).* Reflection applied to a clause in (iv) generates either the empty clause or a clause in (iv). Equational factoring applies only to a clause in (iv.i), to yield another clause of the same kind. The only other applicable inferences are superpositions that generate more clauses in (iv). Specifically, clauses of kind (iv.a) superpose with clauses in (iv.a), (iv.f), (iv.h) and (iv.i) to generate clauses in (iv.i), (iv.a) and (iv.d). Clauses of kind (iv.b) superpose with clauses in (iv.b), (iv.e), (iv.g) and (iv.i) to generate clauses in (iv.i), (iv.b) and (iv.d). Clauses of kind (iv.c) superpose with clauses in (iv.c), (iv.e), (iv.f), (iv.g), (iv.h) and (iv.i) to generate clauses in (iv.i), (iv.c), (iv.a) and (iv.b). Clauses of kind (iv.d) superpose with clauses in (iv.d) and (iv.i) to generate clauses in (iv.i) and (iv.d). Clauses of kind (iv.e) superpose with clauses in (iv.e), (iv.g) and (iv.i) to generate clauses in (iv.e) and (iv.g). Clauses of kind (iv.f) superpose with clauses in (iv.f), (iv.h) and (iv.i) to generate clauses in (iv.f) and (iv.h). Clauses of kind (iv.g) superpose with clauses in (iv.g) and (iv.i) to generate clauses in (iv.i) and (iv.g). Clauses of kind (iv.h) superpose with clauses in (iv.h) and (iv.i) to generate clauses in (iv.i) and (iv.h). Clauses of kind (iv.i) superpose with clauses in (iv.i) to generate clauses in (iv.i). □

It follows that the limit is finite and a fair derivation is bound to halt:

LEMMA 12. *A fair $\mathcal{L}$-good $\mathcal{SP}_{\succ}$-strategy is guaranteed to terminate when applied to $\mathcal{L} \cup S$, where S is a set of ground flat $\mathcal{L}$-literals.*

THEOREM 3.5. *A fair $\mathcal{L}$-good $\mathcal{SP}_{\succ}$-strategy is an exponential satisfiability procedure for $\mathcal{L}$.*

PROOF. Let m be the number of subterms occurring in the input set of literals. After flattening the number of subterms is $O(m)$. The types of clauses listed in Lemma 11 include literals of depth at most 2 (cf. (iv.a), (iv.b) and (iv.c)). Hence, at most $h = O(m^3)$ distinct literals and at most $O(2^h)$ clauses can be generated. It follows that the size of the set of clauses during the derivation is bound by a

constant k which is $O(2^h)$. Since applying an inference takes polynomial time in k, the overall complexity is $O(2^{m^3})$.   □

Exponential complexity was expected, because it was shown already in [Nelson and Oppen 1980] that the satisfiability problem for $\mathcal{L}$ is NP-complete.

3.5   The Theory of Arrays

Let INDEX, ELEM and ARRAY be the sorts of indices, elements and arrays, respectively. The signature has two function symbols, select : ARRAY × INDEX → ELEM, and store : ARRAY × INDEX × ELEM → ARRAY, with the usual meaning. The standard presentation, denoted $\mathcal{A}$, is made of two axioms, where x is a variable of sort ARRAY, w and z are variables of sort INDEX and v is a variable of sort ELEM:

$$\forall x, z, v. \ \text{select}(\text{store}(x, z, v), z) \simeq v \qquad (21)$$

$$\forall x, z, w, v. \ (z \not\simeq w \supset \text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w)). \qquad (22)$$

This theory also is not convex, because $z \simeq w \lor \text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w)$ is valid in the theory, but neither disjunct is. For the *theory of arrays with extensionality*, the presentation, named $\mathcal{A}^e$, includes also the extensionality axiom

$$\forall x, y. \ (\forall z. \ \text{select}(x, z) \simeq \text{select}(y, z) \supset x \simeq y), \qquad (23)$$

where x and y are variables of sort ARRAY, and z is a variable of sort INDEX.

DEFINITION 11. *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.*

Given a set of ground $\mathcal{A}$-literals S, *$\mathcal{A}$-reduction* consists of replacing every literal $l \not\simeq r \in S$, where l and r are terms of sort ARRAY, by $select(l, sk_{l,r}) \not\simeq select(r, sk_{l,r})$, where $sk_{l,r}$ is a Skolem constant of sort INDEX. The resulting $\mathcal{A}$-reduced form of S, denoted $Red_{\mathcal{A}}(S)$, is related to the original problem by the following (cf. Lemma 7.1 in [Armando et al. 2003]):

LEMMA 13. *(Armando, Ranise and Rusinowitch 2003) Let S be a set of ground $\mathcal{A}$-literals. $\mathcal{A}^e \cup S$ is satisfiable if and only if $\mathcal{A} \cup Red_{\mathcal{A}}(S)$ is.*

DEFINITION 12. *A CSO $\succ$ is $\mathcal{A}$-good if (1) $t \succ c$ for all ground compound terms t and 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.*

If $\succ$ is an RPO, it is sufficient to impose a precedence $>$, such that function symbols are greater than constant symbols, and $a > e > j$ for all constants a of sort ARRAY, e of sort ELEM and j of sort INDEX, for $\succ$ to be $\mathcal{A}$-good. If it is a KBO, the same precedence and a simple choice of weights will do.

LEMMA 14. *All clauses in the limit S_∞ of the derivation $S_0 \vdash_{\mathcal{SP}} S_1 \ldots S_i \vdash_{\mathcal{SP}} \ldots$ generated by a fair $\mathcal{A}$-good $\mathcal{SP}_\succ$-strategy from $S_0 = \mathcal{A} \cup S$, where S is an $\mathcal{A}$-reduced set of ground flat $\mathcal{A}$-literals, belong to one of the following classes, where a, a' are constants of sort ARRAY, $i, i_1, \ldots, i_n, i'_1, \ldots, i'_n, j_1, \ldots, j_m, j'_1, \ldots, j'_m$ are constants of sort INDEX $(n, m \geq 0)$, e, e' are constants of sort ELEM, and c_1, c_2 are constants of either sort INDEX or sort ELEM:*

i)   *the empty clause;*

ii) *the clauses in $\mathcal{A}$:*
   ii.a) $\text{select}(\text{store}(x, z, v), z) \simeq v$ *and*
   ii.b) $\text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w) \vee z \simeq w;$
iii) *ground flat unit clauses of the form:*
   iii.a) $a \simeq a',$
   iii.b) $c_1 \simeq c_2,$
   iii.c) $c_1 \not\simeq c_2,$
   iii.d) $\text{store}(a, i, e) \simeq a',$
   iii.e) $\text{select}(a, i) \simeq e;$ *and*
iv) *non-unit clauses of the following form:*
   iv.a) $\text{select}(a, x) \simeq \text{select}(a', x) \vee x \simeq i_1 \vee \ldots \vee x \simeq i_n \vee j_1 \bowtie j_1' \vee \ldots \vee j_m \bowtie j_m',$
      for $x \in$ INDEX,
   iv.b) $\text{select}(a, i) \simeq e \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n',$
   iv.c) $e \simeq e' \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n',$
   iv.d) $e \not\simeq e' \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n',$
   iv.e) $i_1 \simeq i_1' \vee i_2 \bowtie i_2' \vee \ldots \vee i_n \bowtie i_n',$
   iv.f) $i_1 \not\simeq i_1' \vee i_2 \bowtie i_2' \vee \ldots \vee i_n \bowtie i_n',$
   iv.g) $t \simeq a' \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n',$ *where t is either a or* $\text{store}(a, i, e)$.

PROOF. We recall that inequalities $a \not\simeq a'$ are not listed in (iii), because S is $\mathcal{A}$-reduced. Since $\succ$ is total on ground terms and $\mathcal{A}$-good, each clause in the above classes has a unique maximal literal, which is the first one in the above listing (up to a permutation of indices for (iv.e) and (iv.f)). Classes (iv.e) and (iv.f) are really one class separated in two classes based on the sign of the maximal literal. The proof is by induction on the sequence $\{S_i\}_i$. For the base case, input clauses are in (ii) or (iii). For the inductive case, we have:

—*Inferences within (ii).* The only inference that applies to the axioms in $\mathcal{A}$ is a superposition of (ii.a) into (ii.b) that generates the trivial clause $z \simeq z \vee \text{select}(x, z) \simeq v$, which is eliminated by deletion.

—*Inferences within (iii).* Inferences between ground flat unit clauses can produce only ground flat unit clauses in (iii) or the empty clause.

—*Inferences between a clause in (iii) and a clause in (ii).* Superposition of (iii.d) $\text{store}(a, i, e) \simeq a'$ into (ii.a) $\text{select}(\text{store}(x, z, v), z) \simeq v$ yields $\text{select}(a', i) \simeq e$ which is in (iii.e). Superposition of (iii.d) into (ii.b) $\text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w) \vee z \simeq w$ yields $\text{select}(a', w) \simeq \text{select}(a, w) \vee i \simeq w$ which is in (iv.a).

—*Inferences between a clause in (iv) and a clause in (ii).* Superposition of (iv.g) $\text{store}(a, i, e) \simeq a' \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n'$ into (ii.a) yields $\text{select}(a', i) \simeq e \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n'$, which is in (iv.b). Superposition of (iv.g) into (ii.b) yields $\text{select}(a', w) \simeq \text{select}(a, w) \vee i \simeq w \vee i_1 \bowtie i_1' \vee \ldots \vee i_n \bowtie i_n'$, which is in (iv.a). No other inferences apply.

—*Inferences between a clause in (iv) and a clause in (iii).* For an inference to apply to (iii.a) and (iv) it must be that a (or a') appears in a clause in (iv). Similarly, for an inference to apply to (iii.b) and (iv) it must be that c_1 (or c_2) appears in a clause in (iv). In either case, simplification of the clause of class (iv) by the clause of class (iii) applies. Such a step can only generate a clause in (iv). The only inference that can apply to (iii.c) and (iv) is a paramodulation

of a clause in (iv) into a clause in (iii.c). If $c_1, c_2 \in$ ELEM, paramodulation of (iv.c) into (iii.c) generates a clause in (iv.d). If $c_1, c_2 \in$ INDEX, paramodulation of (iv.e) into (iii.c) produces a clause in (iv.f) or (iv.e), depending on the sign of the maximal literal in the resulting clause. We consider next (iii.d) and (iv). The only possible application of simplification consists of applying (iii.d) to reduce a clause in (iv.g) to a clause in the same class. A superposition of (iv.c) or (iv.e) into (iii.d) generates a clause in (iv.g). No other inferences are possible. Last come (iii.e) and (iv). As a simplifier, (iii.e) may apply only to (iv.b) to yield a clause in (iv.c). All possible superpositions, namely superposition of (iii.e) and (iv.a), superposition of (iv.e) into (iii.e), and superposition of (iv.g) into (iii.e) give clauses of class (iv.b).

—*Inferences within (iv)*. Equational factoring applies only to a clause of class (iv.e) to yield a clause in (iv.e) or (iv.f). Reflection applies to a clause in (iv.d) or (iv.f), to yield a clause in one of (iii.b), (iii.c), (iv.e) or (iv.f). Then, for each kind of clause we consider all binary inferences it can have with clauses that follows in the list. We begin with (iv.a): superposition of (iv.a) and (iv.a) gives (iv.a); superposition of (iv.a) and (iv.b) gives (iv.b); superposition of (iv.g) into (iv.a) gives (iv.a). Second comes (iv.b): superposition of (iv.b) and (iv.b) gives (iv.c); superposition of (iv.e) into (iv.b) gives (iv.b); superposition of (iv.g) into (iv.b) gives (iv.b). Next there is (iv.c): superposition of (iv.c) and (iv.c) gives (iv.c); paramodulation of (iv.c) into (iv.d) gives (iv.d); superposition of (iv.c) into (iv.g) gives (iv.g). For (iv.e) and (iv.f), we have: superposition of (iv.e) and (iv.e) gives (iv.e) or (iv.f); paramodulation of (iv.e) into (iv.f) gives (iv.e) or (iv.f); superposition of (iv.e) into (iv.g) gives (iv.g). Last, all possible applications of superposition within (iv.g) give (iv.g). □

Thus, we have (cf. Lemma 7.3 and Theorem 7.2 in [Armando et al. 2003]):

LEMMA 15. *(Armando, Ranise and Rusinowitch 2003) A fair $\mathcal{A}$-good $\mathcal{SP}_\succ$-strategy is guaranteed to terminate when applied to $\mathcal{A} \cup S$, where S is an $\mathcal{A}$-reduced set of ground flat $\mathcal{A}$-literals.*

THEOREM 3.6. *(Armando, Ranise and Rusinowitch 2003) A fair $\mathcal{A}$-good $\mathcal{SP}_\succ$-strategy is an exponential satisfiability procedure for $\mathcal{A}$ and $\mathcal{A}^e$.*

4. REWRITE-BASED SATISFIABILITY: COMBINATION OF THEORIES

A big-engines approach is especially well-suited for the combination of theories, because it makes it possible to combine presentations rather than algorithms. The inference engine is the same for all theories considered, and studying a combination of theories amounts to studying the behavior of the inference engine on a problem in the combination. In a little-engines approach, on the other hand, there is in principle a different engine for each theory, and studying a combination of theories may require studying the interactions among different inference engines.

In the rewrite-based methodology, the *combination problem* is the problem of showing that an $\mathcal{SP}_\succ$-strategy decides $\mathcal{T}$-satisfiability, where $\mathcal{T} = \bigcup_{i=1}^n \mathcal{T}_i$, knowing that it decides $\mathcal{T}_i$-satisfiability for all i, $1 \leq i \leq n$. Since $\mathcal{T}_i$-reduction applies separately for each theory, and flattening is harmless, one only has to prove termination. The main theorem in this section establishes sufficient conditions for

$\mathcal{SP}_\succ$ 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 $\succ$ be $\mathcal{T}$-*good*:

DEFINITION 13. *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, $1 \leq i \leq n$.*

The 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 nonvariable subterm:

DEFINITION 14. *A clause C is* variable-inactive *for $\succ$ if no maximal literal in C is 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 15. *A theory presentation $\mathcal{T}$ is* variable-inactive *for $\mathcal{SP}_\succ$ if the limit S_∞ of any fair $\mathcal{SP}_\succ$-derivation from $S_0 = \mathcal{T} \cup S$ is variable-inactive for $\succ$.*

For satisfiability problems, S is ground, hence immaterial for variable-inactivity. If axioms persist, as generally expected, $\mathcal{T} \subseteq S_\infty$, and Definition 15 requires that they are variable-inactive. If they do not persist, they are irrelevant, because a fair strategy does not need to perform inferences from clauses that do not persist.

The third condition is that the signatures do not share function symbols, which excludes paramodulations from compound terms. Sharing of constant symbols, including those introduced by flattening, is allowed. Thus, the only inferences across theories are paramodulations from constants into constants, that are finitely many:

THEOREM 4.1. *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 $\mathcal{T} \cup S_1 \cup \ldots \cup S_n$.*

PROOF. Let S_∞^i be the set of persistent clauses generated by $\mathcal{SP}_\succ$ from $\mathcal{T}_i \cup S_i$. Since $\mathcal{SP}_\succ$ terminates on $\mathcal{T}_i \cup S_i$, for all i, $1 \leq i \leq n$, we are concerned only with binary expansion inferences between a clause in S_∞^i and a clause in S_∞^j, with $1 \leq i \neq j \leq n$. We consider first paramodulations from variables. Assume that a literal $t \simeq x$ occurs in a clause C in S_∞^i. If $x \in Var(t)$, it is $t \succ x$ by the subterm property of the CSO, and therefore, there is no paramodulation from x. If $x \notin Var(t)$, $t \simeq x$ is not maximal in C, because S_∞^i is variable-inactive by hypothesis. In other words, there is another literal L in C such that $L \succ t \simeq x$. By stability of $\succ$, $L\sigma \succ (t \simeq x)\sigma$ for all substitutions σ. Thus, no instance $(t \simeq x)\sigma$ can be maximal, so that, again, there is no paramodulation from x. Therefore, there are no paramodulations from variables. Since there are no shared function symbols, no paramodulation from a compound term applies to a clause in S_∞^i and a clause in S_∞^j. The only possible inferences are those where a clause $a \simeq t \vee C$ paramodulates into a clause $l[a] \bowtie u \vee D$, where a is a constant, t is also a constant (it cannot be a variable, because $a \simeq t \vee C$ is variable-inactive, and it cannot be a compound term, because $\succ$ is stable and good), the context l may be empty, the mgu is empty, and C and D are disjunctions of literals. Since there are only finitely many constants, only finitely many such steps may apply.   □

COROLLARY 3. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be presentations of theories, with no shared function symbol, and let $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$. If for all i, $1 \leq i \leq n$, a fair $\mathcal{T}_i$-good $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{T}_i$, and $\mathcal{T}_i$ is variable-inactive for $\mathcal{SP}_\succ$, then a fair $\mathcal{T}$-good $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{T}$.*

The requirement of being variable-inactive is rather natural for equational theories:

THEOREM 4.2. *If $\mathcal{T}$ is a presentation of an equational theory with no trivial models, then $\mathcal{T}$ is variable-inactive for $\mathcal{SP}_\succ$ for any CSO $\succ$.*

PROOF. By way of contradiction, assume that $\mathcal{T}$ is not variable-inactive, that is, for some variable inactive S_0, S_∞ is not variable inactive. Thus, there is an equation $t \simeq x \in S_\infty$ such that $x \notin Var(t)$. Since $\mathcal{SP}$ is sound, $S_0 \models t \simeq x$. An equation $t \simeq x$ such that $x \notin Var(t)$ is satisfied only by a trivial model. Thus, S_0 has only trivial models. Since $\mathcal{T} \subset S_0$, a model of S_0 is also a model of $\mathcal{T}$. It follows that $\mathcal{T}$ has trivial models, contrary to the hypothesis. □

Given an equational presentation $\mathcal{T}$, the addition of the axiom $\exists x \exists y \ x \not\simeq y$ is sufficient to exclude trivial models. Since the clausal form of this axiom is the ground flat literal $sk_1 \not\simeq sk_2$, where sk_1 and sk_2 are two Skolem constants, this addition preserves all termination results for $\mathcal{SP}_\succ$ on $\mathcal{T}$-satisfiability problems.

For Horn theories, refutational completeness is preserved if $\mathcal{SP}_\succ$ is specialized to a *maximal unit strategy*, that restricts superposition to unit clauses and paramodulates unit clauses into maximal negative literals [Dershowitz 1991]. Equational factoring is not needed for Horn theories. This strategy resembles positive unit resolution in the non-equational case and has the same character of a purely forward-reasoning strategy. At the limit, all proofs in S_∞ are *valley proofs*, that is, equational rewrite proofs in the form $u \rightarrow^* \circ \leftarrow^* t$. It follows that all nonunit clauses are redundant in S_∞ (cf. Theorem 4.9 in [Bonacina and Hsiang 1995]):

THEOREM 4.3. *If $\mathcal{T}$ is a presentation of a Horn equational theory with no trivial models, then $\mathcal{T}$ is variable-inactive for $\mathcal{SP}_\succ$ for any CSO $\succ$ and the maximal unit strategy.*

PROOF. It is the same as for Theorem 4.2, because S_∞ only contains unit clauses. □

For first-order theories, the requirement that S_∞ be variable-inactive excludes the generation of clauses in the form $a_1 \simeq x \vee \ldots \vee a_n \simeq x$, where for all i, $1 \leq i \leq n$, a_i is a constant. Such a disjunction may be generated, but only within a clause that contains at least one greater literal, such as one involving function symbols (e.g., clauses of type (iv.a) in Lemma 14).

THEOREM 4.4. *Let $\mathcal{T}$ be a presentation of a first-order theory: 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 $Th\,\mathcal{T}$ is not stably infinite. Furthermore, if $\mathcal{T}$ has no trivial models, $Th\,\mathcal{T}$ is also not convex.*

PROOF. Since $\mathcal{SP}$ is sound, $a_1 \simeq x \vee \ldots \vee a_n \simeq x \in S_\infty$ implies $S_0 \models \forall x \ a_1 \simeq x \vee \ldots \vee a_n \simeq x$. It follows that S_0 has no infinite model. On the other hand,

$a_1 \simeq x \vee \ldots \vee a_n \simeq x \in S_\infty$ implies that S_0 is satisfiable, because if S_0 were unsatisfiable, by the refutational completeness of $\mathcal{SP}$, S_∞ would contain only the empty clause. Thus, S_0 has models, but has no infinite model. Equivalently, S has $\mathcal{T}$-models, but has no infinite $\mathcal{T}$-model. This means that $Th\,\mathcal{T}$ is not stably infinite, and, if it has no trivial models, it is also not convex by Theorem 2.1.   □

In other words, if $\mathcal{T}$ is not variable-inactive for $\mathcal{SP}_\succ$, because it generates a clause in the form $a_1 \simeq x \vee \ldots \vee a_n \simeq x$, then $\mathcal{T}$ is not stably infinite either.

The notion of a clause in the form $a_1 \simeq x \vee \ldots \vee a_n \simeq x$ was "lifted" in [Bonacina et al. 2006] to those of *variable clause* and *cardinality constraint clause*. A *variable clause* is a clause containing only equations between variables or their negations. The *antecedent-mgu*[10] (*a-mgu*, for short) of a variable clause C is the most general unifier of the unification problem $\{x =^? y : x \not\simeq y \in C\}$. Then, a variable clause C is a *cardinality constraint clause*, if $C^+\mu$ is not empty and contains no trivial equation $x \simeq x$, where μ is the a-mgu of C and C^+ is made of the positive literals in C. This notion allows one to prove the following (cf. Lemma 5.2 in [Bonacina et al. 2006]):

LEMMA 16. *(Bonacina, Ghilardi, Nicolini, Ranise and Zucchelli 2006) If S_0 is a finite satisfiable set of clauses, then S_0 admits no infinite models if and only if the limit S_∞ of any fair $\mathcal{SP}_\succ$-derivation from S_0 contains a cardinality constraint clause.*

Next, we note that a cardinality constraint clause cannot be variable-inactive, because it must have some positive literal in the form $x \simeq y$ that is maximal. For example, in $z \not\simeq y \vee x \simeq y \vee z \simeq w$, all three literals are maximal. Thus, it follows that:

THEOREM 4.5. *If a first-order theory $\mathcal{T}$ is variable-inactive for $\mathcal{SP}_\succ$, then it is stably-infinite.*

PROOF. Assume that $\mathcal{T}$ is not stably-infinite. Then there exists a quantifier-free $\mathcal{T}$-formula φ, that has a $\mathcal{T}$-model but no infinite $\mathcal{T}$-model. Let S_0 be the clausal form of $\mathcal{T} \cup \{\varphi\}$: S_0 is finite, satisfiable and admits no infinite model. By Lemma 16, the limit S_∞ of a fair $\mathcal{SP}_\succ$-derivation from S_0 contains a cardinality constraint clause. Thus, S_∞, and hence $\mathcal{T}$, is not variable-inactive for $\mathcal{SP}_\succ$.   □

We conclude by applying Theorem 4.1 to any combination of the theories studied in Section 3. The goodness requirement (Definition 13) is easily satisfied: any CSO is good for $\mathcal{E}$, and it is simple to obtain an ordering that is simultaneously $\mathcal{R}$-good, $\mathcal{I}$-good, $\mathcal{L}$-good and $\mathcal{A}$-good. The reductions of $\mathcal{R}^e$ to $\mathcal{R}$ (Lemma 1) and $\mathcal{A}^e$ to $\mathcal{A}$ (Lemma 13) apply also when the signature contains free function symbols $f: \mathrm{s}_0, \ldots, \mathrm{s}_{m-1} \to \mathrm{s}_m$, $m \geq 1$, provided that none of the s_i, $1 \leq i \leq m$, is REC or ARRAY, respectively. A function symbol satisfying this requirement is said to be *record-safe* or *array-safe*, respectively. Thus, we have:

THEOREM 4.6. *A fair $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for any combination of the theories of records, with or without extensionality, integer offsets, integer offsets modulo, possibly empty lists, arrays, with or without extensionality, and the*

---

[10]The name derives from the sequent-style notation for clauses adopted in [Bonacina et al. 2006].

quantifier-free theory of equality, provided (1) $\succ$ is $\mathcal{R}$-good whenever records are included, (2) $\succ$ is $\mathcal{I}$-good whenever integer offsets are included, (3) $\succ$ is $\mathcal{L}$-good whenever lists are included and (4) $\succ$ is $\mathcal{A}$-good whenever arrays are included, and (5) all free function symbols are array-safe (record-safe) whenever arrays (records) with extensionality and the quantifier-free theory of equality are included.

PROOF. For the quantifier-free theory of equality, $\mathcal{E}$ is vacuously variable-inactive for $\mathcal{SP}_{\succ}$. For the other theories, the lists of clauses in Lemma 2, Lemma 7, Lemma 9, Lemma 11 and Lemma 14, show that $\mathcal{R}$, $\mathcal{I}$, $\mathcal{I}_k$, $\mathcal{L}$ and $\mathcal{A}$, respectively, are variable-inactive for $\mathcal{SP}_{\succ}$. Therefore, the result follows from Theorem 4.1. □

This theorem holds if $\mathcal{L}$ is replaced by $\mathcal{L}_{Sh}$ or $\mathcal{L}_{NO}$, since they are also variable-inactive for $\mathcal{SP}_{\succ}$ (cf. Lemmata 4.1 and 5.1 in [Armando et al. 2003]).

5. SYNTHETIC BENCHMARKS

This section presents six sets of synthetic benchmarks: three in the *theory of arrays with extensionality* (STORECOMM, SWAP and STOREINV); one in the *combination of the theories of arrays and integer offsets* (IOS); one in the *combination of the theories of arrays, records and integer offsets* to model queues (QUEUE); and one in the *combination of the theories of arrays, records and integer offsets modulo* to model circular queues (CIRCULAR_QUEUE). Each problem set is *parametric*, that is, it is formulated as a function Pb, that takes a positive integer n as parameter, and returns a set of formulæ $Pb(n)$. For all these problems, the size of $Pb(n)$ grows *monotonically* with n. This property makes them ideal to evaluate empirically how a system's performance scales with input's size, as we shall do in Section 6.

5.1 First Benchmark: STORECOMM(n) and STORECOMM_INVALID(n)

The problems of the STORECOMM family express the fact that the result of storing a set of elements in different positions within an array is not affected by the relative order of the store operations. For instance, for $n = 2$ the following valid formula belongs to STORECOMM(n):

$$i_1 \not\simeq i_2 \supset store(store(a, i_1, e_1), i_2, e_2) \simeq store(store(a, i_2, e_2), i_1, e_1).$$

Here and in the following a is a constant of sort ARRAY, $i_1, \ldots, i_n$ are constants of sort INDEX, and $e_1, \ldots, e_n$ are constants of sort ELEM.

In general, let $n > 0$ and p, q be permutations of $\{1, \ldots, n\}$. Let STORECOMM(n, p, q) be the formula:

$$\bigwedge_{(l,m) \in C_2^n} i_l \not\simeq i_m \supset (T_n(p) \simeq T_n(q))$$

where C_2^n is the set of 2-combinations over $\{1, \ldots, n\}$ and

$$T_k(p) = \begin{cases} a & \text{if } k = 0 \\ store(T_{k-1}(p), i_{p(k)}, e_{p(k)}) & \text{if } 1 \le k \le n. \end{cases} \tag{24}$$

Since only the relative position of the elements of p with respect to those of q is relevant, q can be fixed. For simplicity, let it be the identity permutation ι. Then STORECOMM(n) = {STORECOMM(n, p, ι): p is a permutation of $\{1, \ldots, n\}$}.

EXAMPLE 3. *If $n = 3$ and p is such that $p(1) = 3$, $p(2) = 1$, and $p(3) = 2$, then*

$$T_n(p) = store(store(store(a, i_3, e_3), i_1, e_1), i_2, e_2)$$

$$T_n(\iota) = store(store(store(a, i_1, e_1), i_2, e_2), i_3, e_3))$$

and STORECOMM(n, p, ι) *is*

$$((i_1 \not\simeq i_2 \wedge i_2 \not\simeq i_3 \wedge i_1 \not\simeq i_3) \supset$$

$$store(store(store(a, i_3, e_3), i_1, e_1), i_2, e_2) \simeq store(store(store(a, i_1, e_1), i_2, e_2), i_3, e_3)).$$

Each element of STORECOMM(n), once negated, reduced to clausal form, $\mathcal{A}$-reduced and flattened, leads to a problem whose number of clauses is in $O(n^2)$, because it is dominated by the $\binom{n}{2} = \frac{n(n-1)}{2}$ clauses in $\{i_l \not\simeq i_m : (l, m) \in C_2^n\}$.

A slight change in the definition of STORECOMM generates sets of formulæ that are not valid in $\mathcal{A}$:

STORECOMM_INVALID$(n) = \{$STORECOMM$(n, p, \iota') : p$ is a permutation of $\{1, \ldots, n\}\}$

where $\iota' : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ is such that $\iota'(k) = k$ for $1 \leq k \leq n - 1$ and $\iota'(n) = 1$.

EXAMPLE 4. *For n, p and $T_n(p)$ as in Example 3,*

$$T_n(\iota') = store(store(store(a, i_1, e_1), i_2, e_2), i_1, e_1))$$

and STORECOMM_INVALID(n, p, ι') *is*

$$((i_1 \not\simeq i_2 \wedge i_2 \not\simeq i_3 \wedge i_1 \not\simeq i_3) \supset$$

$$store(store(store(a, i_3, e_3), i_1, e_1), i_2, e_2) \simeq store(store(store(a, i_1, e_1), i_2, e_2), i_1, e_1)).$$

5.2 Second Benchmark: SWAP(n) and SWAP_INVALID(n)

An elementary property of arrays is that swapping an element at position i_1 with an element at position i_2 is equivalent to swapping the element at position i_2 with the element at position i_1. The problems of the SWAP family are based on generalizing this observation to any number n of swap operations. For instance, for $n = 2$ the following valid fact is in SWAP(n):

$$swap(swap(a, i_0, i_1), i_2, i_1) \simeq swap(swap(a, i_1, i_0), i_1, i_2)$$

where $swap(a, i, j)$ abbreviates the term $store(store(a, i, select(a, j)), j, select(a, i))$.

In general, let c_1, c_2 be subsets of $\{1, \ldots, n\}$, and let p, q be functions $p, q : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$. Then, we define SWAP$(n, c_1, c_2, p, q)$ to be the formula:

$$T_n(c_1, p, q) \simeq T_n(c_2, p, q)$$

with $T_k(c, p, q)$ defined by

$$T_k(c, p, q) = \begin{cases} a & \text{if } k = 0, \\ swap(T_{k-1}(c, p, q), i_{p(k)}, i_{q(k)}) & \text{if } 1 \leq k \leq n \text{ and } k \in c, \text{ and} \\ swap(T_{k-1}(c, p, q), i_{q(k)}, i_{p(k)}) & \text{if } 1 \leq k \leq n \text{ and } k \notin c. \end{cases} \quad (25)$$

$T_n(c, p, q)$ is the array obtained by swapping the elements of position $p(k)$ and $q(k)$ of the array a for $1 \leq k \leq n$. The role of the subset c is to determine whether the

element at position $p(k)$ has to be swapped with that at position $q(k)$ or vice versa, and has the effect of shuffling the indices within the formula.

EXAMPLE 5. *If $n = 3$, $c_1 = \{1\}$, $c_2 = \{2,3\}$, p and q are such that $p(k) = k$ and $q(k) = 2$, for all k, $1 \le k \le n$, then*

$$T_n(c_1, p, q) = swap(swap(swap(a, i_1, i_2), i_2, i_2), i_2, i_3)$$

$$T_n(c_2, p, q) = swap(swap(swap(a, i_2, i_1), i_2, i_2), i_3, i_2).$$

Thus, $\mathtt{SWAP}(n) = \{\mathtt{SWAP}(n, c_1, c_2, p, q) : c_1, c_2 \subseteq \{1, \dots, n\}$ and $p, q : \{1, \dots, n\} \to \{1, \dots, n\}\}$. Each formula, once negated, transformed into clausal form, $\mathcal{A}$-reduced and flattened, leads to a problem with $O(n)$ clauses.

A small change in the definition produces a class $\mathtt{SWAP\_INVALID}$. With c_1, c_2, p, q defined as for $\mathtt{SWAP}$, let $\mathtt{SWAP\_INVALID}(n, c_1, c_2, p, q)$ be the formula:

$$T_n(c_1, p, q) \simeq T_n(c_2, p, q')$$

where $T_k(c, p, q)$ is as in (25), $q' : \{1, \dots, n\} \to \{1, \dots, n\}$ is such that $q'(1) = (q(1)+1) \mod n$, and $q'(k) = q(k)$ for all k, $2 \le k \le n$. Then, $\mathtt{SWAP\_INVALID}(n) = \{\mathtt{SWAP\_INVALID}(n, c_1, c_2, p, q) : c_1, c_2 \subseteq \{1, \dots, n\}, p, q : \{1, \dots, n\} \to \{1, \dots, n\}\}$.

EXAMPLE 6. *If n, c_1, c_2, p and q are as in Example 5,*

$$T_n(c_1, p, q) = swap(swap(swap(a, i_1, i_2), i_2, i_2), i_2, i_3)$$

$$T_n(c_2, p, q') = swap(swap(swap(a, i_3, i_1), i_2, i_2), i_3, i_2).$$

5.3 Third Benchmark: $\mathtt{STOREINV}(n)$ and $\mathtt{STOREINV\_INVALID}(n)$

The problems of the $\mathtt{STOREINV}$ family capture the following property: if the arrays resulting from swapping elements of array a with the elements of array b occurring in the same positions are equal, then a and b must have been equal to begin with. For the simple case where a single position is involved, we have:

$$store(a, i, select(b, i)) \simeq store(b, i, select(a, i)) \supset a \simeq b.$$

For $n \ge 0$, let $\mathtt{STOREINV}(n) = \{\mathrm{multiswap}(a, b, n) \supset a \simeq b\}$, where

$$\mathrm{multiswap}(a, b, k) = \begin{cases} (a \simeq b) \text{ if } k = 0, \\ \textbf{let } (a' \simeq b') = \mathrm{multiswap}(a, b, k-1) \textbf{ in} \\ \quad store(a', i_k, select(b', i_k)) \simeq store(b', i_k, select(a', i_k)) \\ \quad \text{if } k \ge 1. \end{cases}$$

$$(26)$$

EXAMPLE 7. *For $n = 2$ we have*

$$store(a', i_2, select(b', i_2)) \simeq store(b', i_2, select(a', i_2)) \supset a \simeq b$$

where $a' = store(a, i_1, select(b, i_1))$ and $b' = store(b, i_1, select(a, i_1))$.

Transformation into clausal form of the negation of the formula in $\mathtt{STOREINV}(n)$, followed by $\mathcal{A}$-reduction and flattening, yields a problem with $O(n)$ clauses.

For $\mathtt{STOREINV\_INVALID}$, let $store(t_a, i_n, select(t_b, i_n)) \simeq store(t_b, i_n, select(t_a, i_n))$ be the formula returned by multiswap(a, b, n) for $n \ge 2$. Then, we define

$$\mathtt{STOREINV\_INVALID}(n) =$$

$$\{store(t_a, i_1, select(t_b, i_n)) \simeq store(t_b, i_n, select(t_a, i_n)) \supset a \simeq b\}.$$

5.4   Fourth Benchmark: $\mathtt{IOS}(n)$

The problems of the $\mathtt{IOS}(n)$ family combine the theories of arrays and integer offsets. Consider the following two program fragments:

```
for(k=1;k<=n;k++)              for(k=1;k<=n;k++)
  a[i+k]=a[i]+k;                 a[i+n-k]=a[i+n]-k;
```

If the execution of either fragment produces the same result in the array `a`, then `a[i+n]==a[i]+n` must hold initially for any value of `i`, `k`, `a`, and `n`.

EXAMPLE 8. *For $n = 2$, $\mathtt{IOS}(n)$ includes only the following valid formula:*

$$store(store(a, i + 1, select(a, i) + 1), i + 2, select(a, i) + 2) \simeq$$
$$store(store(a, i + 1, select(a, i + 2) - 1), i, select(a, i + 2) - 2)$$
$$\supset \ select(a, i + 2) \simeq select(a, i) + 2.$$

In general, for $n \geq 0$ let $\mathtt{IOS}(n) = \{L_n^n \simeq R_n^n \supset select(a, i + n) \simeq select(a, i) + n\}$ where

$$L_0^n = R_0^n = a$$
$$L_k^n = store(L_{k-1}^n, i + k, select(a, i) + k) \quad \text{for } k = 1, \dots, n$$
$$R_k^n = store(R_{k-1}^n, i + n - k, select(a, i + n) - k) \quad \text{for } k = 1, \dots, n.$$

Each formula in $\mathtt{IOS}(n)$, once negated, reduced to clausal form, flattened and $\mathcal{I}$-reduced, generates $O(n)$ clauses. $\mathcal{A}$-reduction is not needed, since the negation of the formula does not contain inequalities of sort ARRAY.

5.5   Fifth Benchmark: $\mathtt{QUEUE}(n)$

The theories of arrays, records and integer offsets can be combined to specify *queues*, defined as usual in terms of the functions enqueue : ELEM × QUEUE → QUEUE, dequeue : QUEUE → QUEUE, first : QUEUE → ELEM, last : QUEUE → ELEM and reset : QUEUE → QUEUE, where QUEUE and ELEM are the sorts of queues and their elements, respectively.

Indeed, a queue can be implemented as a record with three fields: *items* is an array storing the elements of the queue, *head* is the index of the first element of the queue in the array, and *tail* is the index where the next element will be inserted in the queue. Following Section 3.1, the signature features function symbols rstore$_{items}$, rstore$_{head}$, rstore$_{tail}$, rselect$_{items}$, rselect$_{head}$ and rselect$_{tail}$, abbreviated as rstore$_i$, rstore$_h$, rstore$_t$, rselect$_i$, rselect$_h$, rselect$_t$, respectively. Then, the above mentioned functions on QUEUE are defined as follows:

$$\begin{aligned}
\text{enqueue}(v, x) &= \text{rstore}_t(\text{rstore}_i(x, store(\text{rselect}_i(x), \text{rselect}_t(x), v)), \text{s}(\text{rselect}_t(x))) \\
\text{dequeue}(x) &= \text{rstore}_h(x, \text{s}(\text{rselect}_h(x))) \\
\text{first}(x) &= select(\text{rselect}_i(x), \text{rselect}_h(x)) \\
\text{last}(x) &= select(\text{rselect}_i(x), \text{p}(\text{rselect}_t(x))) \\
\text{reset}(x) &= \text{rstore}_h(x, \text{rselect}_t(x))
\end{aligned}$$

where x and v are variables of sort QUEUE and ELEM, respectively, store and select

are the function symbols from the signature of $\mathcal{A}$, p and s are the function symbols for predecessor and successor from the signature of $\mathcal{I}$.

A basic property of queues is the following: assume that q_0 is a properly initialized queue and q is obtained from q_0 by performing $n+1$ enqueue operations $(n > 0)$, that insert $n+1$ elements $e_0, e_1, \ldots, e_n$, and m dequeue operations $(0 \le m \le n)$, that remove m elements $e_0, e_1, \ldots, e_{m-1}$; then $\text{first}(q) = e_m$. Dequeue operations can be interleaved with enqueue operations in any order, provided the number of dequeue operations is always strictly smaller than the number of preceding enqueue operations. For instance, if $q = \text{enqueue}(e_2, \text{dequeue}(\text{enqueue}(e_1,$ $\text{enqueue}(e_0, \text{reset}(q_0)))))$, then $\text{first}(q) = e_1$. Problems in the $\texttt{QUEUE}(n)$ family express an instance of this property, where the dequeue operator is applied once every 3 applications of the enqueue operator. Thus, the number of dequeue operations will be $m = \lfloor (n+1)/3 \rfloor$.

Given a term t where the function symbols reset, enqueue, dequeue, first, last and reset may occur, let $t{\downarrow}$ denote the term obtained from t by unfolding the above function definitions. Then $\texttt{QUEUE}(n)$, for $n > 0$, is defined as follows:

$$\texttt{QUEUE}(n) = \{(q_0 \simeq \text{reset}(q){\downarrow} \wedge \bigwedge_{i=0}^{n-1} q_{i+1} \simeq f_{i+1}(e_i, q_i)) \supset \text{first}(q_n){\downarrow} \simeq e_m\},$$

where

$$f_i(e, q) = \begin{cases} \text{dequeue}(\text{enqueue}(e, q)){\downarrow} & \text{if } i \bmod 3 = 0, \\ \text{enqueue}(e, q){\downarrow} & \text{otherwise,} \end{cases}$$

and $m = \lfloor (n+1)/3 \rfloor$.

EXAMPLE 9. *If $n = 1$, then $\texttt{QUEUE}(n)$ is the formula:*

$$\begin{pmatrix} q_0 \simeq \text{rstore}_h(q, \text{rselect}_t(q)) & \wedge \\ q_1 \simeq \text{rstore}_t(\ \text{rstore}_i(q_0, \text{store}(\text{rselect}_i(q_0), \text{rselect}_t(q_0), e_0)), \text{s}(\text{rselect}_t(q_0))) \end{pmatrix} \supset$$
$$\text{select}(\text{rselect}_i(q_1), \text{rselect}_h(q_1)) \simeq e_0.$$

Each formula in $\texttt{QUEUE}(n)$, once negated, reduced to clausal form, flattened and $\mathcal{I}$-reduced, generates $O(n)$ clauses. $\mathcal{A}$-reduction and $\mathcal{R}$-reduction are not needed, since the negation of the formula does not contain inequalities of sort ARRAY or REC.

5.6 Sixth Benchmark: CIRCULAR_QUEUE(n, k)

It is sufficient to replace the theory of integer offsets $\mathcal{I}$ by $\mathcal{I}_k$, to work with indices modulo k and extend the approach of the previous section to model *circular queues of length k*. The problems of the CIRCULAR_QUEUE(n, k) family say that if q_{n+1} is obtained from a properly initialized circular queue q_0 by inserting $n+1$ elements $e_0, e_1, \ldots, e_n$, for $n > 0$, and $n \bmod k = 0$, then $\text{first}(q_{n+1}) \simeq \text{last}(q_{n+1})$ holds, because the last element inserted overwrites the one in the first position (e.g., picture inserting 4 elements in a circular queue of length 3). This is formally expressed by

$$\texttt{CIRCULAR\_QUEUE}(n, k) =$$

$$\{(q_0 \simeq \text{reset}(q){\downarrow} \wedge \bigwedge_{i=0}^{n} q_{i+1} \simeq \text{enqueue}(e_i, q_i){\downarrow}) \supset \text{first}(q_{n+1}){\downarrow} \simeq \text{last}(q_{n+1}){\downarrow}\},$$

for $n > 0$ such that $n \bmod k = 0$.

EXAMPLE 10. *If $k = n = 1$, then* CIRCULAR_QUEUE$(1, 1)$ *is the formula:*

$$
\left(
\begin{array}{c}
q_0 \simeq \text{rstore}_h(q, \text{rselect}_t(q)) \ \wedge \\
q_1 \simeq \text{rstore}_t(\text{rstore}_i(q_0, \text{store}(\text{rselect}_i(q_0), \text{rselect}_t(q_0), e_0)), \text{s}(\text{rselect}_t(q_0))) \ \wedge \\
q_2 \simeq \text{rstore}_t(\text{rstore}_i(q_1, \text{store}(\text{rselect}_i(q_1), \text{rselect}_t(q_1), e_1)), \text{s}(\text{rselect}_t(q_1))) \\
\supset \text{select}(\text{rselect}_i(q_2), \text{rselect}_h(q_2)) \simeq \text{select}(\text{rselect}_i(q_2), \text{p}(\text{rselect}_t(q_2))).
\end{array}
\right)
$$

Each formula in CIRCULAR_QUEUE(n, k), once negated, reduced to clausal form, flattened and $\mathcal{I}$-reduced, generates $O(n)$ clauses.

6. EXPERIMENTS

The synthetic benchmarks of Section 5 were submitted to three systems: E 0.82, CVC 1.0a and CVC Lite 1.1.0. The prover E implements (a variant of) $\mathcal{SP}$ with a large choice of fair search plans, based on the "given-clause" algorithm [Schulz 2002; 2004], that ensure that the empty clause will be found, if the input is unsatisfiable, and a finite satisfiable saturated set will be generated, if the input set is satisfiable and admits one. CVC [Stump et al. 2002] and CVC Lite [Barrett and Berezin 2004] combine several theory decision procedures following the Nelson-Oppen method, including that of [Stump et al. 2001] for arrays with extensionality, and integrate them with a SAT engine [Barrett et al. 2002a]. CVC is no longer supported; it was superseded by CVC Lite, a more modular and programmable system. While CVC Lite has many advantages, at the time of these experiments CVC was reported to be still faster on many problems. CVC and CVC Lite feature a choice of SAT solvers: a built-in solver or Chaff [Moskewicz et al. 2001] for CVC, a *"fast"* or a *"simple"* solver for CVC Lite. In our experiments, CVC and CVC Lite performed consistently better with their built-in and *"fast"* solver, respectively, and therefore only those results are reported.

We wrote a generator of pseudo-random instances of the synthetic benchmarks, producing either TPTP[11] or CVC syntax, and a set of scripts to run the solvers on all benchmarks. The generator creates either T-reduced, flattened input files or plain input files. Flattening times were not included in the reported run times, because flattening is a one-time linear time operation, and the time spent on flattening was insignificant. In the following, *native input* means flattened, T-reduced files for E, and plain, unflattened files for CVC and CVC Lite.

The experiments were performed on a 3.00GHz Pentium 4 PC with 512MB RAM. Time and memory were limited to 150 sec and 256 MB per instance. If a system ran out of either time or memory under these limits, the result was recorded as a "failure." When $Pb(n)$ is not a singleton (cf. STORECOMM(n), STORECOMM_INVALID(n), SWAP(n) and SWAP_INVALID(n)), the *median* run time over all tested instances is reported.[12] For the purpose of computing the median, a failure is considered to be larger than all successful run times. The median was chosen in place of the average,

---

[11]TPTP, or "Thousands of Problems for Theorem Provers" is a *de facto* standard for testing general-purpose first-order theorem provers: see `http://www.tptp.org/`.
[12]Reported figures refer to runs with 9 instances for every value of n. Different numbers of instances (e.g., 5, 20) were also tried, but the impact on the plots was negligible.

precisely because it is well-defined even in cases where a system fails on some, but not all instances of a given size, a situation that occurred for all systems.

The results for E refer to two variants of a simple strategy, termed *E(good-lpo)* and *E(std-kbo)*, for reasons that will be clear shortly. This strategy adopts a *single priority queue for clause selection*, where *E(good-lpo)* gives the same priority to all clauses, while *E(std-kbo)* privileges ground clauses. Additionally, *E(good-lpo)* ensures that all input clauses are selected before the generated ones, whereas *E(std-kbo)* does not. Both variants employ a very simple *clause evaluation heuristic* to rank clauses of equal priority: it weights clauses by counting symbols, giving weight 2 to function and predicate symbols and weight 1 to variable symbols. Since these are the default term weights that E uses for a variety of operations, they are pre-computed and cached, so that this heuristic is very fast, compared to more complex schemes.

E(std-kbo) features a Knuth-Bendix ordering (KBO), where the weight of symbols is given by their arity, and the precedence sorts symbols by arity first, and by inverse input frequency second (i.e., rarer symbols are greater), with ties broken by order of appearance in the input. This KBO is $\mathcal{R}$-good, $\mathcal{I}$-good, and it satisfies Condition (1), but not Condition (2), in Definition 12, so that it is not $\mathcal{A}$-good. It was included because it is a typical ordering for first-order theorem proving, and therefore *E(std-kbo)* can be considered representative of the behavior of a plain, standard, theorem-proving strategy. *E(good-lpo)* has a lexicographic (recursive) path ordering (LPO), whose precedence extends that of *E(std-kbo)*, in such a way that constants are ordered by sort. Thus, also Condition (2) in Definition 12 is satisfied and the resulting LPO is $\mathcal{R}$-good, $\mathcal{I}$-good and $\mathcal{A}$-good. In both precedences, constants introduced by flattening are smaller than those in the original signature. It is worth emphasizing that contemporary provers, such as E, can generate precedences and weighting schemes automatically. The only human intervention was a minor modification in the code to enable the prover to recognize the sort of constants and satisfy Condition (2) in Definition 12.

6.1 Experiments with STORECOMM and STORECOMM_INVALID

Many problems involve *distinct objects*, that is, constants which name elements that are known to be distinct in all models of the theory. E features a complete variant of $\mathcal{SP}$ that builds knowledge of the existence of distinct objects into the inference rules [Schulz and Bonacina 2005]. Under this refinement, the prover treats strings in double quotes and positive integers as distinct objects. This aspect is relevant to the STORECOMM problems, because they include the inequalities in $\{i_l \not\simeq i_m : (l, m) \in C_2^n\}$, stating that all indices are distinct. Thus, E was applied to these problems in two ways: with $\{i_l \not\simeq i_m : (l, m) \in C_2^n\}$ included in the input (*axiomatized indices*), and with array indices in double quotes (*built-in index type*).

Figure 3 shows that all systems solved the problems comfortably and scaled smoothly. 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 outperformed CVC Lite by a factor of about 2.5. CVC performed best improving by another factor of 2. It is somewhat surprising that E, a theorem prover optimized for showing *unsatisfiability*, performed comparatively even better on invalid (that is, satisfiable) instances, where it was faster
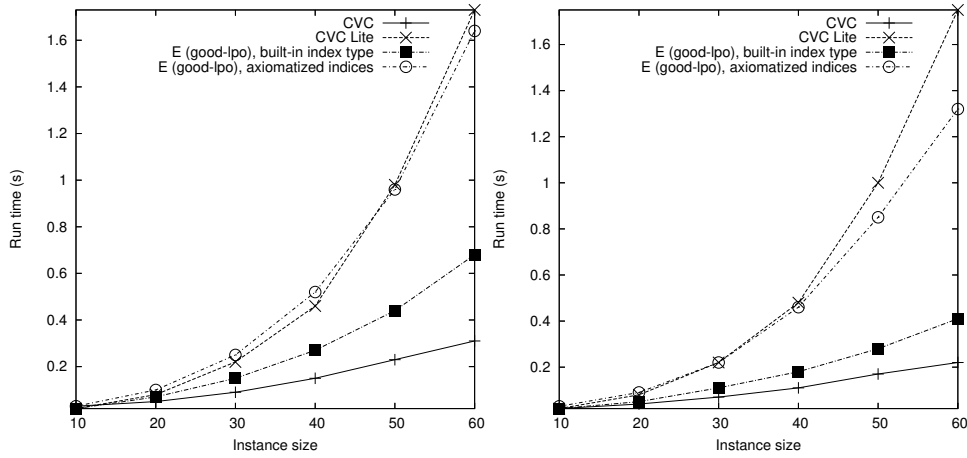
Fig. 3. Performance on valid (left) and invalid (right) STORECOMM instances with native input.

than CVC Lite, and *E(good-lpo)* with built-in indices came closer to CVC. The shared characteristics of all the plots strongly suggest that for STORECOMM the most important feature is sheer processing speed. Although there is no deep reasoning or search involved, the general-purpose prover can hold its own against the specialized solvers, and even edge out CVC Lite.
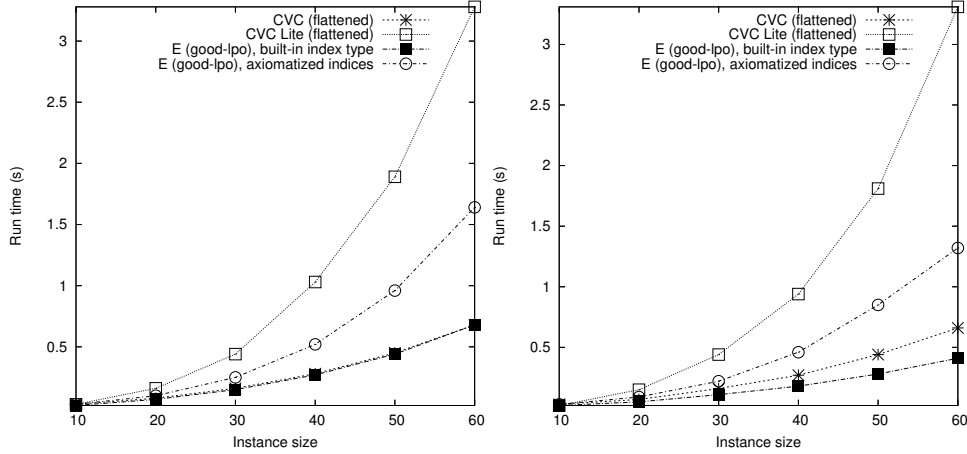

Fig. 4. Performance on valid (left) and invalid (right) STORECOMM instances with flat input for all.

When *all* systems ran on flattened input (Figure 4), both CVC and CVC Lite exhibited run times approximately two times higher than with native format, and CVC Lite turned out to be the slowest system. CVC and E with built-in indices were the fastest: on valid instances, their performances are so close, that the plots

coincide, but E is faster on invalid instances. It is not universally true that flattening hurt CVC and CVC Lite: on the SWAP problems CVC Lite performed much better on flattened input. This suggests that specialized decision procedures are not insensitive to input format.

Although CVC was overall the fastest system on STORECOMM, E was faster than CVC Lite, and did better than CVC on invalid instances when they were given the same input. As CVC may be considered a paradigmatic representative of optimized systems with built-in theories, it is remarkable that the general-purpose prover could match CVC and outperform CVC Lite.

6.2   Experiments with SWAP and SWAP_INVALID
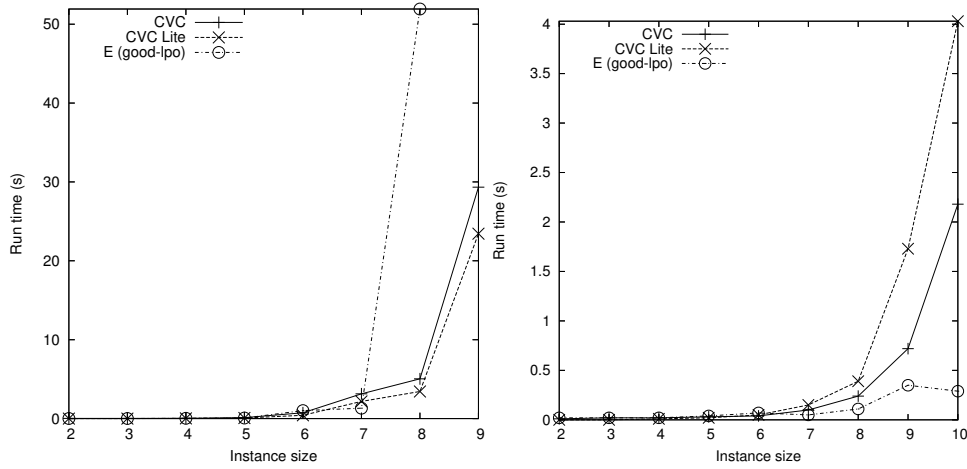
Fig. 5.   Performance on valid (left) and invalid (right) SWAP instances, native input.

Rather mixed results arose for SWAP, as shown in Figure. 5. Up to instance size 5, the systems are very close. Beyond this point, on valid instances, E leads up to size 7, but then is overtaken by CVC and CVC Lite. E could solve instances of size 8, but was much slower than CVC and CVC Lite, which solved instances up to size 9. No system could solve instances of size 10. For invalid instances, E solved easily instances up to size 10 in less than 0.5 sec. CVC and CVC Lite were much slower there, taking 2 sec and 4 sec, respectively. Their asymptotic behaviour seems to be clearly worse.

Consider 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*. Figure 6 displays the systems' performance on valid instances, when the input for E includes this lemma. Although this addition means that the theorem prover is no longer a decision procedure,[13] E

---

[13]Lemma 14 and therefore Theorem 3.6 do not hold, if this lemma is added to presentation $\mathcal{A}$.
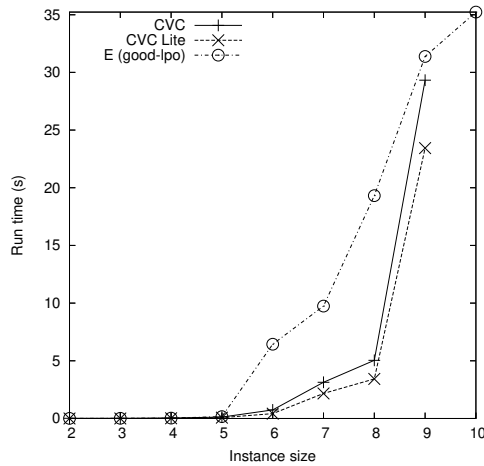
Fig. 6.   Performance on valid SWAP instances with added lemma for E.

terminated also on instances of size 9 and 10, and its plot suggests a better asymptotic behavior. While no system emerged as a clear winner, this experiment shows how a prover that takes a theory presentation in input offers an additional degree of freedom, because useful lemmata may be added.

6.3   Experiments with STOREINV and STOREINV_INVALID
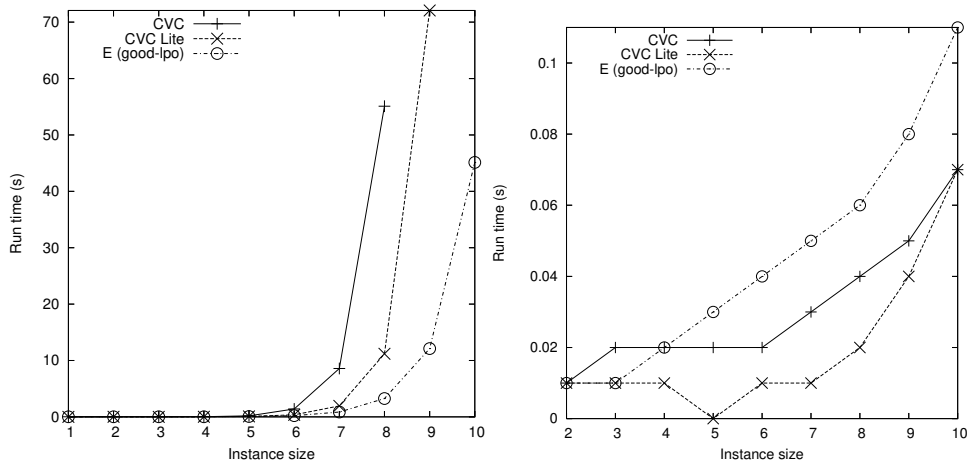
Fig. 7. Performance on valid (left) and invalid (right) STOREINV instances with native input.

The comparison becomes even more favorable for the prover on the STOREINV problems, reported in Figure 7. CVC solved valid instances up to size 8 within the given resource limit. CVC Lite went up to size 9, but E solved instances of size

10, the largest generated. A comparison of absolute 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) solved valid instances in *nearly constant time*, taking less than 0.3 sec for the hardest problem. Altogether, E with a suitable ordering was clearly qualitatively superior than the dedicated systems. For invalid instances, E did not do as well, but the run times there were minimal, with the largest run time for instances of size 10 only about 0.1 sec.

6.4   Experiments with IOS
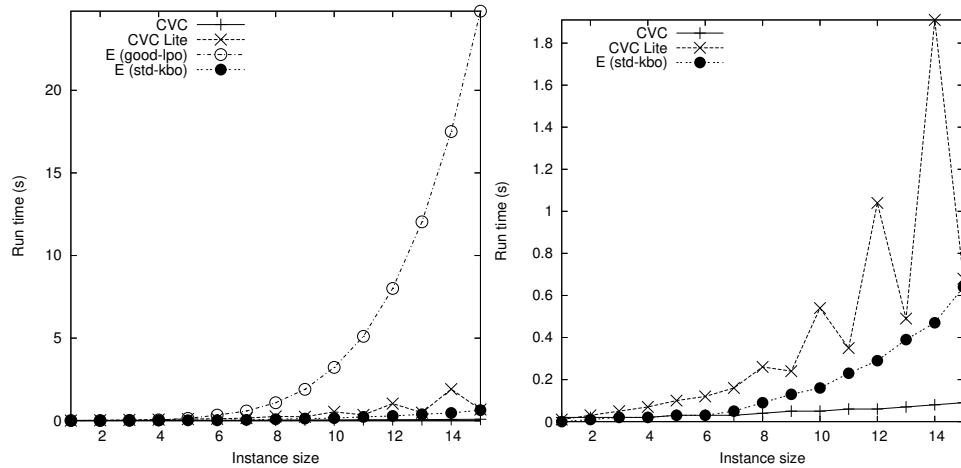
Fig. 8. Performance on the IOS instances: since in the graph on the left the curve for CVC is barely visible, the graph on the right shows a rescaled version of the same data, including only the three fastest systems.

The IOS problems were encoded for CVC and CVC Lite by using their 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 even reported incorrect results.[14] In terms of performance (Figure 8, left), CVC was clearly the best system, as expected from a tool with built-in arithmetic. *E(good-lpo)* was no match, although it still solved all tried instances (Figure 8, left). On the other hand, *E(std-kbo)* proved to be competitive with the systems with built-in arithmetic. At least two reasons explain why *E(std-kbo)* behaved much better than *E(good-lpo)*: first, KBO turned out to be more suitable than LPO for these benchmarks; second, by not preferring initial clauses, the search plan of *E(std-kbo)* did not consider the acyclicity and array axioms early in the search, a choice that turned out to be good. More remarkably, *E(std-kbo)* did better than CVC Lite (Figure 8, right): its curve scales smoothly, while CVC Lite displays oscillating run times, showing worse performance for even instance sizes than for odd ones.

---

[14]This is a known bug, that will not be fixed since CVC is no longer supported [Stump 2005]. CVC Lite 1.1.0 does not support inductive types.

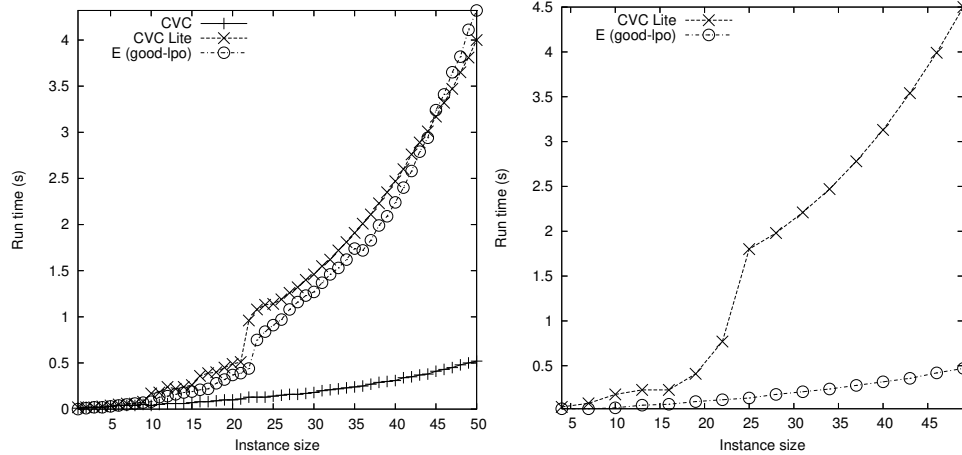## 6.5   Experiments with `QUEUE` and `CIRCULAR_QUEUE`

Fig. 9. Performance of E, CVC and CVC Light on `QUEUE` (left) and `CIRCULAR_QUEUE` (right) with $k = 3$.

Similar to the `IOS` tests, CVC and CVC Light were expected to enjoy a great advantage over E on the `QUEUE` and `CIRCULAR_QUEUE` problems, because both CVC and CVC Light build all theories involved in these benchmarks, namely arrays, records and linear arithmetic, into their decision procedures. The diagram on the left of Figure 9 confirms this expectation, showing that CVC was the fastest system on the `QUEUE`(n) problems. However, *E(good-lpo)* was a good match for CVC Light, and *E(std-kbo)* (not reported in the figure) also solved all tried instances. The plots on the right of Figure 9 refers to the experiments with `CIRCULAR_QUEUE`(n, k), where $k = 3$. It does not include CVC, because CVC cannot handle the *modulo-k* integer arithmetic required for circular queues. Between CVC Lite and E, the latter demonstrated a clear superiority: *E(good-lpo)* exhibited nearly linear performance, and proved the largest instance in less than 0.5 sec, nine times faster than CVC Lite. *E(std-kbo)* behaved similarly.

6.6   Experiments with "Real-World" Problems

While synthetic benchmarks test scalability, "real-world" problems such as those from the UCLID suite [Lahiri and Seshia 2004] test performance on huge sets of literals. UCLID is a system that reduces all problems to propositional form without theory reasoning. Thus, in order to get problems relevant to our study, we used haRVey [Déharbe and Ranise 2003] to extract $\mathcal{T}$-satisfiability problems from various UCLID inputs. This resulted in 55,108 proof tasks in the *combination of the theory of integer offsets and the quantifier-free theory of equality*, so that $\mathcal{I}$-reduction was applied next. We ran E on all of them, using a cluster of 3 PC's with 2.4GHz Pentium-4 processors. All other parameters were the same as for the synthetic benchmarks.
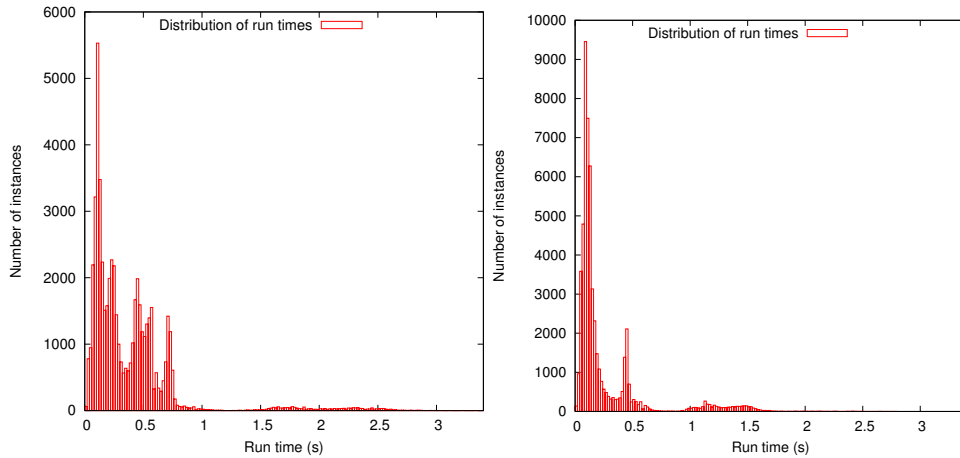
Fig. 10. Distribution of run times for E in automatic mode (left) and using an optimized strategy (right) for the UCLID test set.

These problems (all valid) turned out to be easy for E in automatic mode, where the prover chooses automatically ordering and search plan. It could solve *all* problems, taking less than 4 sec on the hardest one, with average 0.372 sec and median 0.25 sec. Figure 10 shows a histogram of run times: the vast majority of problems was solved in less than 1 sec and very few needed 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, very similar to *E(std-kbo)*, the performance improved by about 40% (Figure 10, right): the average is 0.249 sec, the median 0.12 sec, the longest time 2.77 sec, and most problems were solved in less than 0.5 sec.

7. DISCUSSION

The application of automated reasoning to verification has long shown the importance of decision procedures for satisfiability in decidable theories. The most common approach to these procedures, popularized as the *"little" proof engines* paradigm [Shankar 2002], works by building each theory $\mathcal{T}$ into a dedicated inference engine. For $\mathcal{T}$-satisfiability procedures, that decide conjunctions of ground $\mathcal{T}$-literals, the mainstay is the congruence closure algorithm, enhanced by building $\mathcal{T}$ into the algorithm (e.g., [Nelson and Oppen 1980; Stump et al. 2001; Lahiri and Musuvathi 2005b; 2005a; Dutertre and de Moura 2006; Nieuwenhuis and Oliveras 2007]). Such procedures are combined in accordance with the scheme of [Nelson and Oppen 1979] or its variants (e.g., [Tinelli and Harandi 1996; Barrett et al. 2002b; Ganzinger 2002; Ghilardi 2004; Baader and Ghilardi 2005; Ranise et al. 2005]). Recent systematic treatments appeared in [Krstić and Conchon 2003; Manna and Zarba 2003; Conchon and Krstić 2003; Ranise et al. 2004; Ganzinger et al. 2004; Ghilardi et al. 2005].

For $\mathcal{T}$-decision procedures, that decide arbitrary quantifier-free $\mathcal{T}$-formulæ, the so-called "eager" approaches seek efficient reductions of the problems to SAT and

submit them to SAT solvers (e.g., [Jackson and Vaziri 2000; Bryant and Velev 2001; Bryant et al. 2002; Seshia et al. 2003; Meir and Strichman 2005]). The so-called "lazy" approaches (e.g., [de Moura et al. 2002; Barrett et al. 2002a; Flanagan et al. 2003; Déharbe and Ranise 2003; Barrett and Berezin 2004; Ganzinger et al. 2004; Detlefs et al. 2005; Bozzano et al. 2005; Bozzano et al. 2005; Nieuwenhuis and Oliveras 2007]) integrate $\mathcal{T}$-satisfiability procedures based on congruence closure with SAT solvers, usually based on the Davis-Putnam-Logemann-Loveland procedure (e.g., [Chang and Lee 1973; Moskewicz et al. 2001]). The resulting systems are called SMT solvers.

By symmetry with *little proof engines*, we used *"big" proof engines* (e.g., [Stickel 2002]), for theorem-proving strategies for full first-order logic with equality, as implemented in state-of-the-art general-purpose theorem provers (e.g., [McCune 2003; Weidenbach et al. 1999; Riazanov and Voronkov 2002; Schulz 2002]). There has always been a continuum between big and little engines of proof, as testified by the research on *reasoning modulo a theory* in big engines. The *rewriting approach to $\mathcal{T}$-satisfiability* aims at a cross-fertilization where big engines work *as* little engines. The general idea is to explore how the technology of big engines (orderings, inference rules, search plans, algorithms, data structures, implementation techniques) may be applied selectively and efficiently "in the small," that is, to decide specific theories.

This exploration finds its historical roots in the relationship between congruence closure and Knuth-Bendix completion [Knuth and Bendix 1970] in the ground case: the application of ground completion to compute congruence closure was discovered as early as [Lankford 1975]; the usage of congruence closure to generate canonical rewrite systems from sets of ground equations was investigated further in [Gallier et al. 1993; Plaisted and Sattler-Klein 1996]; more recently, a comparison of ground completion and congruence closure algorithms was given in [Bachmair et al. 2003]; and ground completion and congruence closure were included in an abstract framework for canonical inference in [Bonacina and Dershowitz 2007]. The central component of the rewriting approach is the inference system $\mathcal{SP}$, that is an offspring of a long series of studies on completion for first-order logic with equality. These systems were called by various authors *rewrite-based*, *completion-based*, *superposition-based*, *paramodulation-based*, *contraction-based*, *saturation-based* or *ordering-based*, to emphasize one aspect or the other. Relevant surveys include [Plaisted 1993; Bonacina 1999; Nieuwenhuis and Rubio 2001; Dershowitz and Plaisted 2001].

The gist of the rewriting approach to $\mathcal{T}$-satisfiability is to show that a sound and refutationally complete "big" engine, such as $\mathcal{SP}$, is guaranteed to generate finitely many clauses from $\mathcal{T}$-satisfiability problems. By adding termination to soundness and completeness, one gets a decision procedure: an $\mathcal{SP}$-strategy that combines $\mathcal{SP}$ with a fair search plan is a $\mathcal{T}$-satisfiability procedure. Depending on the theory, termination may require some problem transformation, termed $\mathcal{T}$-*reduction*, which is fully mechanizable for all the theories we have studied. We emphasize that the inference system is *not* adapted to the theories. The only requirement is on the ordering: $\mathcal{SP}$ is parametric with respect to a CSO, and the termination proof for $\mathcal{T}$ may require that this ordering satisfies some property named $\mathcal{T}$-*goodness*. For all considered theories, $\mathcal{T}$-goodness is very simple and easily satisfied by common

orderings such as RPO's and KBO's.

We proved termination of $\mathcal{SP}$ on several new theories, including one with infinite axiomatization[15]. We gave a *general modularity theorem* for the *combination of theories*, and carried out an experimental evaluation, to test the pratical feasibility of the rewriting approach. The modularity theorem states sufficient conditions (no shared function symbols, variable-inactive theories) for $\mathcal{SP}$ to terminate on a combination of theories if it terminates on each theory separately. The "no shared function symbols" hypothesis is common for combination results. Variable-inactivity is satisfied by all equational theories with no trivial models. First-order theories that fail to be variable-inactive in an intuitive way are not stably infinite, and therefore cannot be combined by the Nelson-Oppen scheme either. On the other hand, it follows from work in [Bonacina et al. 2006] that variable-inactive theories are stably infinite. The quantifier-free theories of *equality*, *lists*, *arrays* with or without extensionality, *records* with or without extensionality, *integer offsets* and *integer offsets modulo*, all satisfy these requirements, so that any fair $\mathcal{SP}$-strategy is a satisfiability procedure for any of their combinations. The theories of arrays and possibly empty lists are *not convex*, and therefore cannot be combined by the Nelson-Oppen scheme without case analysis.

A different approach to big engines in the context of theory reasoning, and especially combination of theories, appeared in [Ganzinger et al. 2006]: the combination or extension of theories is conceived as mixing total and partial functions, and a new inference system with *partial superposition* is introduced to handle them. Because the underlying notion of validity is modified to accomodate partial functions, the emphasis of [Ganzinger et al. 2006] is on defining the new inference system and proving its completeness. The realization of such an approach requires to implement the new inference system. The essence of our methodology, on the other hand, is to leave the inference system and its completeness proof unchanged, and prove termination to get decision procedures. This allows us to take existing theorem provers "off the shelf."

For the experimental evaluation, we designed six sets of synthetic benchmarks on the theory of arrays with extensionality and combinations of the theories of arrays, records and integer offsets or integer offsets modulo. For "real-world problems," we considered satisfiability benchmarks extracted from the UCLID suite. Our experimental comparison between the $\mathcal{SP}$-based E prover and the validity checkers CVC and CVC Lite is a first of its kind, and offers many elements for reflection and suggestions for future research.

The analysis of the traces of the theorem prover showed that these satisfiability problems behave very differently compared to more typical theorem-proving problems. Classical proof tasks involve a fairly large set of axioms, a rich signature, many universally quantified variables, many unit clauses usable as rewrite rules and many mixed positive/negative literal clauses. The search space is typically infinite, and only a very small part of it gets explored. In $\mathcal{T}$-satisfiability problems, input presentations are usually very small and there is a large number of ground rewrite rules generated by flattening. The search space is finite, but nearly all of it has

---

[15]Our termination result for the theory of integer offsets was generalized in [Bonacina and Echenim 2007b] to the theories of *recursive data structures* as defined in [Oppen 1980].

to be explored, before unsatisfiability (validity) can be shown. Table I compares the behavior of E on some medium-difficulty unsatisfiable array problems and some representative TPTP problems of similar difficulty for the prover.

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 data are for E in automatic mode. STORECOMM(60)/1 is one of the problems in STORECOMM(n) for $n = 60$, STOREINV(5) is STOREINV(n) for $n = 5$ and SWAP(8)/3 is one of the problems in SWAP(n) for $n = 8$. The others are representative problems from TPTP 3.0.0. The sum of processed and remaining clauses may be smaller than the sum of initial and generated clauses, because E removes newly generated trivial clauses, as well as unprocessed clauses whose parents become redundant, without counting them as processed. The final column shows the percentage of *all* inferences (expansion and contraction) that did *not* contribute to the final proof.

Table I.   Performance characteristics of array and TPTP problems.

Most search plans and features of first-order provers were designed assuming the search space characteristics of typical first-order problems. Thus, the theorem prover turned out to be competitive with the little-engines systems, although it was optimized for different search problems. This means that not only is using a theorem prover already a viable option in practice, but there is a clear potential to improve both performance and usability, by studying implementation techniques of first-order inferences that target $\mathcal{T}$-satisfiability, by designing *theory-specific search plans*, and by equipping the prover with the ability to recognize which theories appear in the input set. The prover also terminated in many cases beyond the known termination results (cf. Figure 6 and the runs with *E(std-kbo)*, whose ordering is not $\mathcal{A}$-good). Thus, theorem provers are not as brittle as one may fear with respect to termination, and still offer the flexibility of adding useful lemmata to the presentation, as shown in Section 6.2.

The above remarks suggest that stronger termination results may be sought. The complexity of the rewrite-based procedures may be improved by adopting *theory-specific search plans*. Methods to extract models from saturated sets can be investigated to complement proof finding with *model generation*, which is important for applications. For instance, in verification, a model represents a counter-example to a conjecture of correctness of a system. The ability to generate models marks the difference between being able to tell that there are errors (by reporting "satisfiable"), and being able to give some information on the errors (by reporting "satisfiable" and a model). Since we do not expect the rewrite-based approach to work for *full linear arithmetic*, another quest is how to integrate it with methods for arithmetic [Rueß and Shankar 2004] or other theories such as *bit-vectors* [Cyrluk et al. 1996]. Research on integration with the latter theory began in [Kirchner et al. 2005].

Most verification problems involve arbitrary quantifier-free $\mathcal{T}$-formulæ, or, equivalently, sets of ground $\mathcal{T}$-clauses. Thus, a major open issue is how to apply big engines towards solving general $\mathcal{T}$-decision problems. Since $\mathcal{SP}$ is an inference system for general first-order clauses, a set of ground $\mathcal{T}$-clauses may be submitted to an $\mathcal{SP}$-based prover. Showing that an $\mathcal{SP}$-strategy is a $\mathcal{T}$-decision procedure requires extending the termination results from sets of ground literals to sets of ground clauses. Sufficient conditions for termination of $\mathcal{SP}$ on $\mathcal{T}$-decision problems were given recently in [Bonacina and Echenim 2007a]. However, verification problems of practical interest typically yield large sets with huge non-unit clauses, and first-order provers are not designed to deal with large disjunctions as efficiently as SAT solvers.

In practice, a more plausible approach could be to follow a simple "lazy" scheme (e.g., [Barrett et al. 2002a; Flanagan et al. 2003]), and integrate a rewrite-based $\mathcal{T}$-satisfiability procedure with a SAT solver that generates assignments. The rewrite-based $\mathcal{T}$-solver produces a proof, whenever it detects unsatisfiability, and can be made incremental to interact with the SAT solver according to the "lazy" scheme. However, the state of the art in SMT solvers indicates that a tight integration of the two solvers is required to achieve high performances. SAT solvers are based on case analysis by backtracking, whereas rewrite-based inference engines are *proof-confluent*, which means they need no backtracking. While proof confluence is an advantage in first-order theorem proving, this dissimilarity means that a tight integration of SAT solver and rewrite-based $\mathcal{T}$-solver requires to address the issues posed by the interplay of two very diverse kinds of control.

In current work we are taking a different route: we are exploring ways to decompose $\mathcal{T}$-decision problems, in such a way that the big engine acts as a pre-processor for an SMT solver, doing as much theory reasoning as possible in the pre-processing phase. In this way we hope to combine the strength of a prover, such as E, in equational reasoning with that of an SMT solver in case analysis. Even more general problems require to reason with universally quantified variables, that SMT solvers handle only by heuristics, following the historical lead of [Detlefs et al. 2005]. In summary, big engines are strong at reasoning with equalities, universally quantified variables and Horn clauses. Little engines are strong at reasoning with propositional logic, non-Horn clauses and arithmetic. The reasoning environments of the future will have to harmonize their forces.

REFERENCES

ARKOUDAS, K., ZEE, K., KUNCAK, V., AND RINARD, M. 2004. Verifying a file system implementation. In *Proceedings of the 6th International Conference on Formal Engineering Methods*, J. Davies, W. Schulte, and M. Barnett, Eds. Lecture Notes in Computer Science, vol. 3308. Springer-Verlag, Berlin, Germany, 373–390.

ARMANDO, A., BONACINA, M. P., RANISE, S., RUSINOWITCH, M., AND SEHGAL, A. K. 2002. High-performance deduction for verification: A case study in the theory of arrays. In *Notes of the 2nd VERIFY Workshop, Federated Logic Conference*. Number 07/2002 in Technical Reports. DIKU, U. Copenhagen, Copenhagen, Denmark, 103–112.

ARMANDO, A., BONACINA, M. P., RANISE, S., AND SCHULZ, S. 2005a. Big proof engines as little proof engines: new results on rewrite-based satisfiability procedures. In *Notes of the 3rd Workshop on Pragmatics of Decision Procedures in Automated Reasoning, CAV-17*. 33–41.

ARMANDO, A., BONACINA, M. P., RANISE, S., AND SCHULZ, S. 2005b. On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal. In *Proceedings of the 5th Workshop on Frontiers of Combining Systems*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, Germany, 65–80.

ARMANDO, A., RANISE, S., AND RUSINOWITCH, M. 2003. A rewriting approach to satisfiability procedures. *Information and Computation 183,* 2, 140–164.

BAADER, F. AND GHILARDI, S. 2005. Connecting many-sorted structures and theories through adjoint functions. In *Proceedings of the 5th Workshop on Frontiers of Combining Systems*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, Germany, 31–47.

BACHMAIR, L., TIWARI, A., AND VIGNERON, L. 2003. Abstract congruence closure. *Journal of Automated Reasoning 31,* 2, 129–168.

BARRETT, C. W. AND BEREZIN, S. 2004. CVC Lite: A new implementation of the Cooperating Validity Checker. In *Proceedings of the 16th International Conference on Computer-Aided Verification*, R. Alur and D. A. Peled, Eds. Lecture Notes in Computer Science, vol. 3114. Springer-Verlag, Berlin, Germany, 515–518.

BARRETT, C. W., DILL, D. L., AND STUMP, A. 2000. A framework for cooperating decision procedures. In *Proceedings of the 17th International Conference on Automated Deduction*, D. McAllester, Ed. Lecture Notes in Artificial Intelligence, vol. 1831. Springer-Verlag, Berlin, Germany, 79–97.

BARRETT, C. W., DILL, D. L., AND STUMP, A. 2002a. Checking satisfiability of first-order formulas by incremental translation to SAT. In *Proceedings of the 14th International Conference on Computer-Aided Verification*, K. G. Larsen and E. Brinksma, Eds. Lecture Notes in Computer Science, vol. 2404. Springer-Verlag, Berlin, Germany, 236–249.

BARRETT, C. W., DILL, D. L., AND STUMP, A. 2002b. A generalization of Shostak's method for combining decision procedures. In *Proceedings of the 4th Workshop on Frontiers of Combining Systems*, A. Armando, Ed. Lecture Notes in Computer Science, vol. 2309. Springer-Verlag, Berlin, Germany.

BONACINA, M. P. 1999. A taxonomy of theorem-proving strategies. In *Artificial Intelligence Today – Recent Trends and Developments*, M. J. Wooldridge and M. Veloso, Eds. Lecture Notes in Artificial Intelligence, vol. 1600. Springer-Verlag, Berlin, Germany, 43–84.

BONACINA, M. P. AND DERSHOWITZ, N. 2007. Abstract canonical inference. *ACM Transactions on Computational Logic 8,* 1 (January), 180–208.

BONACINA, M. P. AND ECHENIM, M. 2007a. Rewrite-based decision procedures. In *Proceedings of the 6th STRATEGIES Workshop, Federated Logic Conference 2006*, M. Archer, T. Boy de la Tour, and C. Munoz, Eds. ENTCS, vol. 174(11). Elsevier, Amsterdam, The Netherlands, 27–45.

BONACINA, M. P. AND ECHENIM, M. 2007b. Rewrite-based satisfiability procedures for recursive data structures. In *Proceedings of the 4th Workshop on Pragmatics of Decision Procedures in Automated Reasoning, Federated Logic Conference 2006*, B. Cook and R. Sebastiani, Eds. ENTCS, vol. 174(8). Elsevier, Amsterdam, The Netherlands, 55–70.

BONACINA, M. P., GHILARDI, S., NICOLINI, E., RANISE, S., AND ZUCCHELLI, D. 2006. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, U. Furbach and N. Shankar, Eds. Lecture Notes in Artificial Intelligence, vol. 4130. Springer-Verlag, Berlin, Germany, 513–527.

BONACINA, M. P. AND HSIANG, J. 1995. Towards a foundation of completion procedures as semidecision procedures. *Theoretical Computer Science 146*, 199–242.

BOZZANO, M., BRUTTOMESSO, R., CIMATTI, A., JUNTTILA, T., RANISE, S., SEBASTIANI, R., AND VAN ROSSUM, P. 2005. Efficient satisfiability modulo theories via delayed theory combination. In *Proceedings of the 17th International Conference on Computer-Aided Verification*, K. Etessami

and S. Rajamani, Eds. Lecture Notes in Computer Science, vol. 3576. Springer-Verlag, Berlin, Germany, 335–349.

BOZZANO, M., BRUTTOMESSO, R., CIMATTI, A., JUNTTILA, T., VAN ROSSUM, P., SCHULZ, S., AND SEBASTIANI, R. 2005. MathSAT: Tight integration of SAT and mathematical decision procedures. *Journal of Automated Reasoning 35,* 1–3 (October), 265–293.

BRYANT, R. E., LAHIRI, S. K., AND SESHIA, S. A. 2002. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *Proceedings of the 14th International Conference on Computer-Aided Verification*, K. G. Larsen and E. Brinksma, Eds. Lecture Notes in Computer Science, vol. 2404. Springer-Verlag, Berlin, Germany, 78–92.

BRYANT, R. E. AND VELEV, M. N. 2001. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic. *ACM Transactions on Computational Logic 2,* 1, 93–134.

CAFERRA, R., LEITSCH, A., AND PELTIER, N. 2004. *Automated Model Building.* Kluwer Academic Publishers, Amsterdam, The Netherlands.

CHANG, C.-L. AND LEE, R. C.-T. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London, U.K.

CONCHON, S. AND KRSTIĆ, S. 2003. Strategies for combining decision procedures. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, H. Garavel and J. Hatcliff, Eds. Lecture Notes in Computer Science, vol. 2619. Springer-Verlag, Berlin, Germany, 537–553.

CYRLUK, D., LINCOLN, P., AND SHANKAR, N. 1996. On Shostak's decision procedure for combination of theories. In *Proceedings of the 13th International Conference on Automated Deduction*, M. A. McRobbie and J. K. Slaney, Eds. Lecture Notes in Artificial Intelligence, vol. 1104. Springer-Verlag, Berlin, Germany, 463–477.

CYRLUK, D., MÖLLER, O., AND RUESS, H. 1996. An efficient decision procedure for a theory of fixed-sized bitvectors with composition and extraction. Tech. Rep. UIB96-08, Fakultät für Informatik, Universität Ulm, Ulm, Germany.

DE MOURA, L., OWRE, S., RUESS, H., RUSHBY, J., AND SHANKAR, N. 2004. The ICS decision procedures for embedded deduction. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, D. Basin and M. Rusinowitch, Eds. Lecture Notes in Artificial Intelligence, vol. 3097. Springer-Verlag, Berlin, Germany, 218–222.

DE MOURA, L., RUESS, H., AND SOREA, M. 2002. Lazy theorem proving for bounded model checking over infinite domains. In *Proceedings of the 18th International Conference on Automated Deduction*, A. Voronkov, Ed. Lecture Notes in Artificial Intelligence, vol. 2392. Springer-Verlag, Berlin, Germany, 438–455.

DÉHARBE, D. AND RANISE, S. 2003. Light-weight theorem proving for debugging and verifying units of code. In *Proceedings of the 1st SEFM*. IEEE Computer Society Press, 220–228.

DERSHOWITZ, N. 1991. A maximal-literal unit strategy for Horn clauses. In *Proceedings of the 2nd Workshop on Conditional and Typed Rewriting Systems (CTRS'90)*, S. Kaplan and M. Okada, Eds. Lecture Notes in Computer Science, vol. 516. Springer-Verlag, Berlin, Germany, 14–25.

DERSHOWITZ, N. AND PLAISTED, D. A. 2001. Rewriting. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier, Amsterdam, The Netherlands, 535–610.

DETLEFS, D. L., NELSON, G., AND SAXE, J. B. 2005. Simplify: a theorem prover for program checking. *Journal of the ACM 52,* 3, 365–473.

DOWNEY, P. J., SAMET, H., AND SETHI, R. 1978. Off-line and on-line algorithms for deducing equalities. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages*. ACM Press, 158–170.

DOWNEY, P. J., SETHI, R., AND TARJAN, R. E. 1980. Variations on the common subexpression problem. *Journal of the ACM 27,* 4, 758–771.

DUTERTRE, B. AND DE MOURA, L. 2006. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the 18th International Conference on Computer-Aided Verification*, T. Ball and R. B. Jones, Eds. Lecture Notes in Computer Science, vol. 4144. Springer-Verlag, Berlin, Germany, 81–94.

Fermüller, C., Leitsch, A., Hustadt, U., and Tammet, T. 2001. Resolution decision procedures. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 2. Elsevier, Amsterdam, The Netherlands, Chapter 25, 1793–1849.

Flanagan, C., Joshi, R., Ou, X., and Saxe, J. B. 2003. Theorem proving using lazy proof explication. In *Proceedings of the 15th International Conference on Computer-Aided Verification*, W. A. Hunt Jr. and F. Somenzi, Eds. Lecture Notes in Computer Science, vol. 2725. Springer-Verlag, Berlin, Germany, 355–367.

Gallier, J., Narendran, P., Plaisted, D. A., Raatz, S., and Snyder, W. 1993. Finding canonical rewriting systems equivalent to a finite set of ground equations in polynomial time. *Journal of the ACM 40,* 1, 1–16.

Ganzinger, H. 2002. Shostak light. In *Proceedings of the 18th International Conference on Automated Deduction*, A. Voronkov, Ed. Lecture Notes in Artificial Intelligence, vol. 2392. Springer-Verlag, Berlin, Germany, 332–347.

Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., and Tinelli, C. 2004. DPLL(T): Fast decision procedures. In *Proceedings of the 16th International Conference on Computer-Aided Verification*, R. Alur and D. A. Peled, Eds. Lecture Notes in Computer Science, vol. 3114. Springer-Verlag, Berlin, Germany, 175–188.

Ganzinger, H., Ruess, H., and Shankar, N. 2004. Modularity and refinement in inference systems. Tech. Rep. CSL-SRI-04-02, SRI International, Menlo Park, CA, U.S.A.

Ganzinger, H., Sofronie, V., and Waldmann, U. 2006. Modular proof systems for partial functions with Evans equality. *Information and Computation 240,* 10, 1453–1492.

Ghilardi, S. 2004. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning 33,* 3-4, 221–249.

Ghilardi, S., Nicolini, E., and Zucchelli, D. 2005. A comprehensive framework for combined decision procedures. In *Proceedings of the 5th Workshop on Frontiers of Combining Systems*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, Germany, 1–30.

Horn, A. 1951. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic 16*, 14–21.

Jackson, D. and Vaziri, M. 2000. Finding bugs with a constraint solver. In *Proceedings of ISSTA 2000*, M. J. Harrold, Ed. ACM SIGSOFT.

Kaufmann, M., Manolios, P., and Moore, Eds., J. S. 2000. *Computer Aided Reasoning : ACL2 Case Studies.* Kluwer, Amsterdam, The Netherlands.

Kirchner, H., Ranise, S., Ringeissen, C., and Tran, D. K. 2005. On superposition-based satisfiability procedures and their combination. In *Proceedings of the 2nd ICTAC*, D. Hung and M. Wirsing, Eds. Lecture Notes in Computer Science, vol. 3722. Springer-Verlag, Berlin, Germany, 594–608.

Knuth, D. E. and Bendix, P. B. 1970. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, J. Leech, Ed. Pergamon Press, Oxford, U.K., 263–297.

Krstić, S. and Conchon, S. 2003. Canonization for disjoint unions of theories. In *Proceedings of the 19th International Conference on Automated Deduction*, F. Baader, Ed. Lecture Notes in Computer Science, vol. 2741. Springer-Verlag, Berlin, Germany.

Lahiri, S. K. and Musuvathi, M. 2005a. An efficient decision procedure for UTVPI constraints. In *Proceedings of the 5th Workshop on Frontiers of Combining Systems*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, Germany, 168–183.

Lahiri, S. K. and Musuvathi, M. 2005b. An efficient Nelson-Oppen decision procedure for difference constraints over rationals. In *Notes of the 3rd Workshop on Pragmatics of Decision Procedures in Automated Reasoning, CAV-17.* 2–9.

Lahiri, S. K. and Seshia, S. A. 2004. The UCLID decision procedure. In *Proceedings of the 16th International Conference on Computer-Aided Verification*, R. Alur and D. A. Peled, Eds. Lecture Notes in Computer Science, vol. 3114. Springer-Verlag, Berlin, Germany, 475–378.

Lankford, D. S. 1975. Canonical inference. Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX, U.S.A.

Lynch, C. and Morawska, B. 2002. Automatic decidability. In *Proceedings of the 17th IEEE Symposium in Logic in Computer Science*, G. Plotkin, Ed. IEEE Computer Society Press, Los Alamitos, CA, U.S.A.

Lynch, C. and Tran, D.-K. 2007. Automatic decidability and combinability revisited. In *Proceedings of the 21st International Conference on Automated Deduction*, F. Pfenning, Ed. Lecture Notes in Artificial Intelligence, vol. 4603. Springer-Verlag, Berlin, Germany, 328–344.

Manna, Z. and Zarba, C. G. 2003. Combining decision procedures. In *Formal methods at the crossroads: from panacea to foundational support*. Lecture Notes in Computer Science, vol. 2757. Springer-Verlag, Berlin, Germany, 381–422.

McCune, W. W. 2003. Otter 3.3 reference manual. Tech. Rep. ANL/MCS-TM-263, MCS Division, Argonne National Laboratory, Argonne, IL, U.S.A.

Meir, O. and Strichman, O. 2005. Yet another decision procedure for equality logic. In *Proceedings of the 17th International Conference on Computer-Aided Verification*, K. Etessami and S. Rajamani, Eds. Lecture Notes in Computer Science, vol. 3576. Springer-Verlag, Berlin, Germany, 307–320.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th DAC*, D. Blaauw and L. Lavagno, Eds. 530–535.

Necula, G. and Lee, P. 1998. Efficient representation and validation of proofs. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, V. Pratt, Ed. IEEE Computer Society Press, Los Alamitos, CA, U.S.A., 93–104.

Nelson, G. and Oppen, D. C. 1979. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems 1,* 2, 245–257.

Nelson, G. and Oppen, D. C. 1980. Fast decision procedures based on congruence closure. *Journal of the ACM 27,* 2, 356–364.

Nieuwenhuis, R. and Oliveras, A. 2007. Fast congruence closure and extensions. *Information and Computation 205,* 4, 557–580.

Nieuwenhuis, R. and Rubio, A. 2001. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier, Amsterdam, The Netherlands, 371–443.

Oppen, D. C. 1980. Reasoning about recursively defined data structures. *Journal of the ACM 27,* 3.

Owre, S., Rushby, J. M., and Shankar, N. 1992. PVS: a prototype verification system. In *Proceedings of the 11th International Conference on Automated Deduction*, D. Kapur, Ed. Lecture Notes in Artificial Intelligence, vol. 607. Springer-Verlag, Berlin, Germany, 748–752.

Plaisted, D. A. 1993. Equational reasoning and term rewriting systems. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Vol. I: Logical Foundations. Oxford University Press, Oxford, U.K., 273–364.

Plaisted, D. A. and Sattler-Klein, A. 1996. Proof lengths for equational completion. *Information and Computation 125,* 2, 154–170.

Ranise, S. and Déharbe, D. 2003. Light-weight theorem proving for debugging and verifying pointer manipulating programs. In *Proceedings of the 4th International Workshop on First-order Theorem Proving*, I. Dahn and L. Vigneron, Eds. Number DSCI–II710/03 in Technical Reports. Universidad Politécnica de Valencia, Valencia, Spain, 119–132.

Ranise, S., Ringeissen, C., and Tran, D. K. 2004. Nelson-Oppen, Shostak and the extended canonizer: A family picture with a newborn. In *Proceedings of the 1st ICTAC*, K. Araki and Z. Liu, Eds. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany.

Ranise, S., Ringeissen, C., and Zarba, C. G. 2005. Combining data structures with nonstably infinite theories using many-sorted logic. In *Proceedings of the 5th Workshop on Frontiers of Combining Systems*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, Germany, 48–64.

Riazanov, A. and Voronkov, A. 2002. The design and implementation of VAMPIRE. *Journal of AI Communications 15,* 2/3, 91–110.

Ruess, H. 2004. Personal communication on ICS 2.0 (e-mail message to Alessandro Armando).

Ruess, H. and Shankar, N. 2001. Deconstructing Shostak. In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, J. Halpern, Ed. IEEE Computer Society Press, Los Alamitos, CA, U.S.A.

Ruess, H. and Shankar, N. 2004. Solving linear arithmetic constraints. Tech. Rep. CSL-SRI-04-01, SRI International, Menlo Park, CA, U.S.A.

Schulz, S. 2002. E – A brainiac theorem prover. *Journal of AI Communications 15,* 2–3, 111–126.

Schulz, S. 2004. System description: E 0.81. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, D. Basin and M. Rusinowitch, Eds. Lecture Notes in Artificial Intelligence, vol. 3097. Springer-Verlag, Berlin, Germany, 2223–228.

Schulz, S. and Bonacina, M. P. 2005. On handling distinct objects in the superposition calculus. In *Notes of the 5th International Workshop on Implementation of Logics, LPAR-11.* 66–77.

Seshia, S. A., Lahiri, S. K., and Bryant, R. E. 2003. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proceedings of the 40th DAC*, L. Fix and L. Lavagno, Eds. 425–430.

Shankar, N. 2002. Little engines of proof. Invited talk, Federated Logic Conference, Copenhagen, Denmark; and course notes of Fall 2003, `http://www.csl.sri.com/users/shankar/LEP.html`.

Shostak, R. E. 1978. An algorithm for reasoning about equality. *Communications of the ACM 21,* 7, 583–585.

Shostak, R. E. 1984. Deciding combinations of theories. *Journal of the ACM 31,* 1, 1–12.

Stickel, M. E. 2002. Herbrand Award speech. Federated Logic Conference, Copenhagen, Denmark.

Stump, A. 2005. Personal communication on CVC 1.0a (e-mail message to Alessandro Armando).

Stump, A., Barrett, C. W., and Dill, D. L. 2002. CVC: a Cooperating Validity Checker. In *Proceedings of the 14th International Conference on Computer-Aided Verification*, K. G. Larsen and E. Brinksma, Eds. Lecture Notes in Computer Science, vol. 2404. Springer-Verlag, Berlin, Germany, 500–504.

Stump, A., Barrett, C. W., Dill, D. L., and Levitt, J. 2001. A decision procedure for an extensional theory of arrays. In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, J. Halpern, Ed. IEEE Computer Society Press, Los Alamitos, CA, U.S.A.

Stump, A. and Dill, D. L. 2002. Faster proof checking in the Edinburgh Logical Framework. In *Proceedings of the 18th International Conference on Automated Deduction*, A. Voronkov, Ed. Lecture Notes in Artificial Intelligence, vol. 2392. Springer-Verlag, Berlin, Germany, 392–407.

Tinelli, C. and Harandi, M. 1996. A new correctness proof of the Nelson-Oppen combination procedure. In *Proceedings of the 1st Workshop on Frontiers of Combining Systems*, F. Baader and K. Schulz, Eds. Applied Logic Series, vol. 3. Kluwer Academic Publishers, Amsterdam, The Netherlands.

van Dalen, D. 1989. *Logic and Structure.* Springer-Verlag, Berlin, Germany. Second edition.

Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C., and Topić, D. 1999. System description: Spass version 1.0.0. In *Proceedings of the 16th International Conference on Automated Deduction*, H. Ganzinger, Ed. Lecture Notes in Artificial Intelligence, vol. 1632. Springer-Verlag, Berlin, Germany, 378–382.