ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Information and Computation 183 (2003) 140–164

Information and Computation

www.elsevier.com/locate/ic

A rewriting approach to satisfiability procedures☆

Alessandro Armando,[a] Silvio Ranise,[a,b,*] and Michaël Rusinowitch[b]

[a] *DIST–Università degli Studi di Genova, via all'Opera Pia 13, Genova 16145, Italy*
[b] *LORIA & INRIA-Lorraine, 615, rue du Jardin Botanique, BP 101, Villers les Nancy Cedex 54602, France*

Abstract

We show how a well-known superposition-based inference system for first-order equational logic can be used almost directly for deciding satisfiability in various theories including lists, encryption, extensional arrays, extensional finite sets, and combinations of them. We also give a superposition-based decision procedure for homomorphism.

Keywords: Automated deduction; Equational logic; Term rewriting; Superposition; Decision procedures; Lists; Encryption; Arrays with extensionality; Finite sets with extensionality; Homomorphism

1. Introduction

In verification with proof assistants (such as PVS [33], Coq [20], HOL [17], and Nqthm [11]), decision procedures are typically used for eliminating trivial subgoals represented (for instance) as sequents modulo a background theory. These theories axiomatise standard data-types such as arrays, lists, bit-vectors and have proved to be quite useful for, e.g., hardware verification. Elimination of trivial sequents often reduces to the **problem of proving the unsatisfiability of conjunctions of literals modulo a background theory** T, which is the problem we shall consider here.

The rewriting approach permits us the uniform design of decision procedures for eliminating these subgoals and also offers an efficient alternative to congruence closure techniques. This approach was inspired by Greg Nelson's thesis [29] where it is suggested to apply Knuth–Bendix

☆ Some preliminary results have appeared in [2].

* Corresponding author.
E-mail addresses: armando@dist.unige.it (A. Armando), ranise@loria.fr (S. Ranise), rusi@loria.fr (M. Rusinowitch).

0890-5401/03/$ - see front matter
doi:10.1016/S0890-5401(03)00020-8

completion to derive decision procedures. Here, instead of the Knuth–Bendix completion procedure, we apply a standard complete superposition-based inference system for clausal equational logic (given for instance in [32]). This allows us not only to handle pure equality but also several interesting axiomatic theories that were not handled previously that way such as lists, encryption, extensional arrays, and finite sets with extensionality. The proof that the procedures are correct is straightforward w.r.t. other correctness proofs given in the literature (compare for instance our decision procedure for extensional arrays of Section 7 with [38]). In our approach, combining theories is also immediate. As an illustration, we show how to decide a combination of lists and arrays.

A second contribution of the paper is in the same spirit of applying Knuth–Bendix completion to derive a decision procedure for the theory of homomorphism. This is the first decision procedure, to our knowledge, for this theory.

1.1. Related work

We discuss results that are closely related to ours. In previous work, the rewriting approach was mainly used for pure equality theories. For instance, [5] focus on abstracting the control of congruence closure algorithms, in order to give a uniform presentation of several known algorithms. A recent extension to deal with equality modulo AC is presented in [4].

In [30], a decision procedure for the "quantifier-free theory of the LISP list structure" is described. The procedure is obtained as an extension of a congruence closure algorithm with a mechanism which augments the graph by selected instances of the axioms of the theory. The proof of correctness is model theoretic and seems difficult to generalise. A discussion of the difficulties of deriving a general method to obtain decision procedures by extending congruence closure algorithms as well as a decision procedure for the theory of arrays (without extensionality) can be found in [29]. This discussion has motivated our work.

In [38], the first decision procedure for an extensional theory of arrays is presented. The key ingredient is a modified congruence closure algorithm which is capable of handling (so called) partial equations. The correctness proof is rather complex and it takes the main part of the paper; it is model-theoretic and rather ad hoc. In Section 7, we give a decision procedure for the same theory considered in [38]. Our procedure is simpler to understand since it amounts to applying (almost directly) standard equality reasoning in contrast to handling partial equalities and our proof of correctness relies on basic properties of skolemization. As a consequence, the decision procedure (as well as its correctness proof) for the theory of arrays with extensionality can be adapted to similar presentations for sets and multisets.

Also we notice that we can easily derive decision procedures for combinations of theories in a manner closely resembling the combination schema described in [31]. This is exemplified for a combination of the theory of lists and arrays in Section 9. Furthermore, the decision procedures derived in our framework can be extended so to provide the interface functionalities needed for them to be plugged into the Nelson and Oppen combination schema [31].

Let us mention that completion was employed as a semi-decision procedure in some works, such as [8–10,19,21]. Furthermore, the use of the Knuth–Bendix completion procedure to derive decision procedures (for equational theories) whenever termination is guaranteed has been emphasised in [8–10].

Finally, our work is similar in spirit to [6] where automated complexity results are given for certain ground entailment problems by exploiting an ordered resolution calculus. A recent work [26] makes this connection more explicit by providing an automation of the results presented in this paper with a refined complexity characterization by using a more sophisticated paramodulation calculus.

Plan of the paper. We start by giving the formal preliminaries and describing the refutationally complete superposition calculus used to derive satisfiability procedures (Section 2). Then, we describe our two step approach to build satisfiability procedures and we apply it to the quantifier-free theory of equality (Section 3). In Sections 4 and 5, we derive satisfiability procedures for two variants of the theory of lists; the former presented in [37] and the latter in [30]. In Section 6, we briefly discuss how our methodology allows us to build a satisfiability procedure for a theory of encryption which can be found in the verification of security protocols. In Sections 7 and 8, we present satisfiability procedures for the theories of arrays and finite sets with extensionality. In Section 9, we show how our methodology can be applied to combinations of theories by considering the combination of the theory of arrays and a theory of lists. In Section 10, we give a decision procedure for the theory of homorphism. The practical relevance of our methodology for automated verification is shown in Section 11 by using the procedure of Section 7 to prove the correctness of a pipelined ALU [12]. Finally, we draw some conclusions in the last section.

2. Preliminaries

We assume the usual (first-order) syntactic notions of *signature*, (*ground*) *term*, *position*, *substitution*, *replacement*, *rewrite relation* $\rightarrow$, as defined, e.g., in [14].

If Σ is a signature and $\mathcal{X}$ is a set of variables, then $T(\Sigma, \mathcal{X})$ denotes the set of terms built out of the symbols in Σ and the variables in $\mathcal{X}$. $T(\Sigma)$ abbreviates $T(\Sigma, \emptyset)$. 0-ary function symbols are called (*individual*) *constants*.[1] Let l and r be elements of $T(\Sigma, \mathcal{X})$, then $l = r$ is a $T(\Sigma, \mathcal{X})$-*equality* and $\neg(l = r)$ (also written as $l \neq r$) is a $T(\Sigma, \mathcal{X})$-*disequality*. A $T(\Sigma, \mathcal{X})$-*literal* is either a $T(\Sigma, \mathcal{X})$-equality or a $T(\Sigma, \mathcal{X})$-disequality, i.e., an expression of the form $s \bowtie t$ where $\bowtie$ is either $=$ or $\neq$. A $T(\Sigma, \mathcal{X})$-*clause* is a disjunction of literals, i.e., an expression of the form $\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$ (abbreviated with $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$) where $A_1, \ldots, A_n$, $B_1, \ldots, B_m$ are $T(\Sigma, \mathcal{X})$-equalities ($n \geqslant 0$ and $m \geqslant 0$). We simply use the terms *equality*, *disequality*, *literals*, and *clauses* when $T(\Sigma, \mathcal{X})$ is clear from the context. A *flat equality* is an equality of the form $f(t_1, \ldots, t_n) = t_0$ or $t_0 = f(t_1, \ldots, t_n)$ where f is an n-ary function symbol and t_i is either a variable or an individual constant for $i = 0, 1, \ldots, n$ with $n \geqslant 0$. A *distinction* is a disequality $t_1 \neq t_2$, where t_i is either a variable or an individual constant for $i = 1, 2$. A *flat literal* is either a flat equality or a distinction. A *flat clause* is a disjunction of flat literals.

We assume the usual (first-order) notions of interpretation, satisfiability, validity, logical consequence (in symbols, $\models$), and theory (see, e.g. [16]). Let S be a set of ground literals, then we say that S is T-*satisfiable* (T-*unsatisfiable*) iff $T \cup S$ is satisfiable (unsatisfiable, resp.). All the theories we shall consider in this paper contain the quantifier-free theory of equality $\mathcal{E}$.

[1] We adopt the Prolog notation to write variables with capitalised letters and constants with small letters.

Example 2.1. Assume that the axiom of T is $h(f(X,Y)) = f(h(X), h(Y))$ (recall that the capitalised letters X and Y denote implicitly universally quantified variables). We can show the T-unsatisfiability of $\{h(c) = c',\ h(c') = c,\ f(c,c') = h(h(a)),\ f(c',c) = a,\ h(h(h(a))) \neq a\}$.

The *satisfiability problem for a theory* T amounts to establishing whether any given finite set of ground literals is T-satisfiable or not. A *satisfiability procedure for* T is any algorithm that solves the satisfiability problem for T.[2]

2.1. A superposition calculus

We will make use of a superposition calculus, $\mathcal{SP}$, comprising the generating inference rules of Fig. 1 and the contraction inference rules of Fig. 2. In the following, we shall write *Factoring* instead of *Equality Factoring* for conciseness. The relation $\succ$ is a reduction ordering [14], which is total on ground terms. $\succ$ is extended to literals in the following way: $(a \bowtie b) \succ (c \bowtie d)$ if $\{a,b\} \succ\succ \{c,d\}$, where $\succ\succ$ is the multiset extension of $\succ$. Multisets of literals are compared using the multiset extension of $\succ$ on literals.

$\mathcal{SP}$ is taken from [32]. It extends the system from [35] by the *equality factoring rule* [3], so that more ordering restrictions are possible in the non-Horn case (in fact, *Factoring* is useless for Horn clauses [24]).

An inference system including contraction inference rules is refutationally complete if *any fair application of the rules to an unsatisfiable set of clauses will derive the empty clause*. Fairness means that if some inference is possible it will be performed at some step unless one of the parent clauses gets simplified, subsumed, or deleted (see, e.g. [35] for a formal definition). The calculus $\mathcal{SP}$ is known to be refutationally complete for general first-order equational logic [3,32].

In this paper, a *saturation* of a set of clauses by $\mathcal{SP}$ is the final set of clauses generated by a fair derivation from S using rules in $\mathcal{SP}$ with higher priority given to the contraction inference rules. If the saturation terminates for the union of T and any set of ground flat literals then it is a satisfiability procedure for T: if the final set of clauses contains the empty clause then the input set of literals is unsatisfiable; it is satisfiable, otherwise. This is a direct consequence of the refutational completeness of $\mathcal{SP}$. From now on, we shall call $\mathcal{SP}$ any fair application of the inference system with priority given to the contraction inference rules.

3. A rewriting approach

We propose a uniform approach based on superposition inference rules to build satisfiability procedures for a variety of decidable theories. Let $\mathcal{T}$ be a theory axiomatised by a set of clauses $Ax(\mathcal{T})$, Σ be a signature containing the symbols of $\mathcal{T}$, and S be a conjunction of ground literals, represented as a finite set. Our goal is to develop a methodology to derive a procedure capable of checking whether S is $\mathcal{T}$-satisfiable or not.

[2] Notice that the satisfiability of any ground formula can be reduced to the satisfiability of sets of literals by splitting on disjunctions, e.g., checking whether $A \vee B$ is satisfiable in the theory T reduces to checking the T-satisfiability of A or B.

Superposition	$\dfrac{\Gamma \Rightarrow \Delta, l[u'] = r \quad \Pi \Rightarrow \Sigma, u = v}{\sigma(\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[v] = r)}$	if (i),(ii),(iii), and iv)
Paramodulation	$\dfrac{\Gamma, l[u'] = r \Rightarrow \Delta \quad \Pi \Rightarrow \Sigma, u = v}{\sigma(l[v] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma)}$	if (i),(ii),(iii), and iv)
Reflection	$\dfrac{\Gamma, u' = u \Rightarrow \Delta}{\sigma(\Gamma \Rightarrow \Delta)}$	if $\sigma(u' = u) \not\preceq \sigma(\Gamma \cup \Delta)$
Factoring	$\dfrac{\Gamma \Rightarrow \Delta, u = v, u' = v'}{\sigma(\Gamma, v = v' \Rightarrow \Delta, u = v')}$	if (i), $\sigma(u) \not\preceq \sigma(\Gamma)$, and $\sigma(u = v) \not\preceq \sigma(\{u' = v'\} \cup \Delta)$

The substitution σ is the most general unifier of u and u', and u' is not a variable in *Superposition* and *Paramodulation*. Furthermore, we have the following abbreviations:

(i) $\sigma(u) \not\preceq \sigma(v)$,

(ii) $\sigma(u = v) \not\preceq \sigma(\Pi \cup \Sigma)$,

(iii) $\sigma(l[u']) \not\preceq \sigma(r)$, and

(iv) $\sigma(l[u'] = r) \not\preceq \sigma(\Gamma \cup \Delta)$.

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

Subsumption	$\dfrac{S \cup \{C, C'\}}{S \cup \{C\}}$	if for some substitution $\theta(C) \subseteq C'$ and there is no substitution ρ s.t. $\rho(C') = C$
Simplification	$\dfrac{S \cup \{C[l'], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$	if $l' = \theta(l)$, $\theta(l) \succ \theta(r)$, and $C[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\dfrac{S \cup \{\Gamma \Rightarrow \Delta, t = t\}}{S}$	

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

The **first step** of our methodology consists of **flattening all the ground literals in** S. We do this by extending the signature Σ with new constants for all the distinct non-constant subterms in S.

Example 3.1. Consider the following set S of ground literals over the signature $\Sigma := \{store/3, select/2, s/0, s_1/0, a/0, a_1/0, v/0, v_1/0\}$ (f/n denotes the fact that the function symbol f has arity $n \geqslant 0$):

$$S := \left\{ \begin{array}{c} store(s, a, v) = store(s_1, a_1, v_1), \\ select(s, a) = v, \\ select(s_1, a_1) = v_1, \\ a = a_1, \qquad v \neq v_1 \end{array} \right\}.$$

The set $S' := S_1 \cup S_2$ of ground flat literals can be derived from S over the extended signature $\Sigma' := \Sigma \cup \{c_1/0, c_2/0, c_3/0, c_4/0\}$, where

$$S_1 := \left\{ \begin{array}{c} store(s,a,v) = c_1, \\ store(s_1,a_1,v_1) = c_2, \\ select(s,a) = c_3, \\ select(s_1,a_1) = c_4 \end{array} \right\} \quad \text{and} \quad S_2 := \left\{ \begin{array}{cc} c_1 = c_2, & \\ c_3 = v, & \\ c_4 = v_1, & \\ a = a_1, & v \neq v_1 \end{array} \right\}.$$

The literals in S_2 are obtained from the literals in S by substituting the constants c_1, c_2, c_3, and c_4 for the corresponding terms as defined by the equations in S_1.

Indeed, flattening preserves the satisfiability of the set S of ground literals.

Lemma 3.1. *Let $\mathcal{T}$ be a $T(\Sigma,X)$-theory and S be a finite set of $T(\Sigma)$-literals. Then there exists a finite set of flat $T(\Sigma')$-literals S' where Σ' is obtained from Σ by adding a finite number of individual constants such that S' is $\mathcal{T}$-satisfiable iff S is.*

The importance of flattening is twofold. Firstly, it simplifies the proofs to be done as the second step of our methodology (see below for more details). Secondly, flattening is a key ingredient for efficiency. In fact, it replaces all occurrence of a given term with a new constant. For this reason, we can perform any inference *inf* (such as, e.g., *Simplification*) on all occurrences of a given term t with an amount of work proportional to applying *inf* on a single occurrence of t and independent of the number of occurrences. As noted in [5,22], introducing new constants for non-constant subterms is equivalent to building and manipulating a directed acyclic graph (dag) as the data structure to represent terms together with a congruence relation on them. Also [34] advocates the use of dags to obtain polynomial time completion algorithms for ground rewrite systems.

It is easy to see that flattening yields a set of flat literals which is $O(n)$, where n measures the size of S (say the length of the string obtained by concatenating the literals in S written in prefix notation).

Let S' be a set of ground literals obtained by flattening the set S. The **second step** of our methodology consists of **analysing all possible inferences of $\mathcal{SP}$ between clauses in** $Ax(\mathcal{T}) \cup S'$. The analysis consists of three phases. The first phase amounts to finding the types of clauses contained in any saturation of $Ax(\mathcal{T}) \cup S'$ by $\mathcal{SP}$. Indeed, guessing the possible types of clauses is greatly simplified by considering only flat literals in S'. The second phase of the analysis is to prove that only finitely many clauses (which are instances of the types of clauses identified during the first phase) can be generated during saturation. If we are able to prove termination, then we are entitled to conclude that $\mathcal{SP}$ is capable of checking the $\mathcal{T}$-satisfiability of any finite set of ground flat literals S'. This implies that $\mathcal{SP}$ is a satisfiability procedure for $\mathcal{T}$ by Lemma 3.1 and the refutational completeness of the calculus. Finally, the third phase amounts to estimating the computational complexity of the derived satisfiability procedure by carefully scrutinising the types of clauses generated during saturation.

3.1. The quantifier-free theory of equality

The following result says that $\mathcal{SP}$ can be used as a satisfiability procedure for the quantifier-free theory of equality $\mathcal{E}$.[3] In fact, the decision procedure we obtain is just a variant of the Knuth–

[3] We do not claim this result to be new; it is stated here only to give the flavour of our approach in the simple case of the pure equational theory.

Bendix completion procedure (similar to the rational reconstruction of Nelson and Oppen's congruence closure algorithm of [5]). We shall assume now and in the remainder of this paper that the ordering $\succ$ **is such that** $t \succ c$ **for each constant** c **and for each ground term** t **that contains a symbol of arity greater than** 0. Notice that it is easy to satisfy this requirement with a suitable precedence ordering.

For simplicity, we assume that the set S of ground literals have been already flattened and Σ is the signature containing the new constants introduced for flattening. In the following, we describe the three phases of the second step of our methodology for the special case of the theory $\mathcal{E}$.

The first phase consists of finding the types of clauses contained in any saturation of $Ax(\mathcal{E}) \cup S'$ by $\mathcal{SP}$. (Notice that $Ax(\mathcal{E})$ is empty since the axioms of equality are built in $\mathcal{SP}$.)

Lemma 3.2. *Let S be a finite set of flat $T(\Sigma)$-literals. All the saturations of S by $\mathcal{SP}$ contains only* (i) *the empty clause or* (ii) *flat literals.*

Proof. Note that *Simplification* is applicable whenever *Superposition* is. Hence, *Superposition* is useless since *Simplification* has higher priority. There are only three cases to consider. Firstly, *Simplification* replaces ground flat literals with ground flat literals. Secondly, *Paramodulation* generates ground flat literals only. Finally, *Reflection* generates the empty clause (which subsumes all other clauses). □

The second phase amounts to proving that only finitely many clauses can be generated during saturation and the third phase to estimating the computational complexity of the satisfiability procedure.

Theorem 3.1. *$\mathcal{SP}$ is a polynomial satisfiability procedure for $\mathcal{E}$.*

Proof. Let n be the size of the input set of flattened literals. First of all, notice that we only require *Simplification* (since *Superposition* is applicable whenever *Simplification* is and we assume that the latter has higher priority than the former) and *Paramodulation*. Let us consider the following (more specialized and fair) strategy. (1) Exhaustively apply the following instances of the *Simplification* rule: (1.1) find two equalities of the form $f(c_1, \ldots, c_n) = c$ and of the form $f(c_1, \ldots, c_n) = c'$ (where $c, c', c_1, \ldots, c_n$ are constants and $c \succ c'$), replace $f(c_1, \ldots, c_n) = c$ by $c = c'$; (1.2) replace all c by c' in the current set of literals. Then, (2) perform all the *Paramodulation*s between literals of the form $c \neq c'$ and $c = c''$, where c, c', and c'' are constants. It is easy to see that phase (1) terminates. Furthermore, we can perform (1.1) in $O(n)$ (by using back pointers) and (1.2) can be done in $O(1)$ (by using a dag to represent terms). Notice also that operations (1.1) and (1.2) are performed at most $O(n)$ times since there exists $O(n)$ constants and after step (1.2) a constant will be eliminated from all the literals but one. Hence, the computational cost of phase (1) is $O(n^2)$. Now, notice that the set of equations (considered as a rewrite system) obtained after phase (1) is inter-reduced, i.e., no rule can be simplified by another one. In particular each right-hand side of some rule never occurs as a left-hand side of another rule. Thus, at most $O(n)$ *Paramodulation*s are possible during phase (2) and each *Paramodulation* can be found in $O(n)$. To conclude, we have that $\mathcal{SP}$ is an $O(n^2)$ satisfiability procedure for $\mathcal{E}$. □

Notice that our satisfiability procedure is comparable to procedures based on the congruence closure algorithm [30]. However, it is possible to do better (i.e., $O(n \log n)$) by using more sophisticated data structures as described in [15].

4. Theory of lists *à la* Shostak

Let $\Sigma_{\mathcal{L}}$ be a signature containing the function symbols cons/2, car/1, and cdr/1. In [37], Shostak considers the theory $\mathcal{L}_{Sh}$ (called the "convex theory of cons, car, and cdr") obtained by adding to $\mathcal{E}$ the following axioms, denoted by $Ax(\mathcal{L}_{Sh})$:

$$\mathsf{car}(\mathsf{cons}(X, Y)) = X, \tag{1}$$

$$\mathsf{cdr}(\mathsf{cons}(X, Y)) = Y, \tag{2}$$

$$\mathsf{cons}(\mathsf{car}(X), \mathsf{cdr}(X)) = X. \tag{3}$$

We apply our approach to the axioms in $Ax(\mathcal{L}_{Sh})$.

Lemma 4.1. *Let S be a finite set of flat $T(\Sigma_{\mathcal{L}})$-literals. Then, the clauses occurring in the saturations of $S \cup Ax(\mathcal{L}_{Sh})$ can only belong to the following categories:* (i) *the empty clause,* (ii) *ground flat literals,* (iii) *literals in $Ax(\mathcal{L}_{Sh})$, and* (iv) *equalities of the form* $\mathsf{cons}(b, \mathsf{cdr}(a)) = a$ *or* $\mathsf{cons}(\mathsf{car}(a), b) = a$, *where a and b are constants.*

Proof. The proof is by induction on the length of the derivations. We observe that $Ax(\mathcal{L}_{Sh})$ is saturated w.r.t. $\mathcal{SP}$. The base case is simple and therefore is omitted. By the induction hypothesis there are four types of clauses to consider. For inferences with *Reflection* the result is obvious. *Deletion* and *Subsumption* do not create new clauses. In the following, let *replacement* be either a *Superposition* or a *Paramodulation* step, for the sake of conciseness. Also, we consider only *replacement*s between literals of type (ii), (iii), and (iv) that are not covered by Lemma 3.2. A *replacement* between a ground flat literal and (1) or (2) yields a ground flat literal (i.e., a literal of type (ii)). A *replacement* between a ground flat literal and (3) gives a literal of type (iv). Replacements between type (ii) and (iv) give type (ii) or type (iv). Replacements between (iv) and (iv) give tautologies. Finally, a *replacement* between (1) or (2) and (iv) yields a ground flat literal (i.e., a literal of type (ii)) or a literal in one of the following forms: $\mathsf{car}(a) = \mathsf{car}(a)$ or $\mathsf{cdr}(a) = \mathsf{cdr}(a)$ (where a is a constant), which are immediately discarded by *Deletion*. □

Lemma 4.2. *Let S be a finite set of flat $T(\Sigma_{\mathcal{L}})$-literals. All the saturations of $S \cup Ax(\mathcal{L}_{Sh})$ are finite.*

Proof. By Lemma 4.1 we know that the saturations of $S \cup Ax(\mathcal{L}_{Sh})$ can only contain literals of type (i)–(iv). It is trivial to see that only a finite number of such literals can be built out of a finite set of symbols and variables (up to variable renaming). □

Theorem 4.1. *$\mathcal{SP}$ is a polynomial satisfiability procedure for $\mathcal{L}_{Sh}$.*

Proof. Let n be the size of the input set of flattened literals. At most $O(n^3)$ flat literals can be created by *Superposition* and *Paramodulation*. The size of the current set of literals in a derivation is always bounded by $O(n^3)$. Applying an inference takes polynomial time in the size of the set of literals. Hence, overall we have a polynomial satisfiability procedure. □

5. Theory of lists *à la* Nelson and Oppen

Let $\Sigma_{\mathcal{L}_{NO}}$ be a signature containing the function symbols in $\Sigma_{\mathcal{L}}$ (cf. Section 4) and the unary predicate symbol atom. In [30], Nelson and Oppen consider the theory $\mathcal{L}_{NO}$ (called the "theory of LISP list structure") obtained by adding to $\mathcal{E}$ the following axioms, denoted by $Ax(\mathcal{L}_{NO})$: the axioms (1), (2) of Section 4, and

$$\neg\mathsf{atom}(X) \Rightarrow \mathsf{cons}(\mathsf{car}(X), \mathsf{cdr}(X)) = X, \tag{4}$$

$$\neg\mathsf{atom}(\mathsf{cons}(X, Y)). \tag{5}$$

Let G be a set of ground literals built over $\Sigma_{\mathcal{L}_{NO}}$. We notice that each occurrence of the form $\mathsf{atom}(t)$ ($\neg\mathsf{atom}(t)$) in G can be replaced with an equality (disequality, respectively). Let G' be the set of $T(\Sigma_{\mathcal{L}})$-literals obtained by replacing all the literals in G of the form $\neg\mathsf{atom}(t)$ and $\mathsf{atom}(t')$ with $\exists X_0, X_1 \cdot t = \mathsf{cons}(X_0, X_1)$ and $t' \neq \mathsf{cons}(X_0, X_1)$, respectively, where t, t' are ground terms and X_0, X_1 are variables.

Property 5.1. *G is $Ax(\mathcal{L}_{NO})$-satisfiable iff G' is $\{(1), (2)\}$-satisfiable.*

Now, let $G := S \cup A \cup N$ where S is a set of flat $T(\Sigma_{\mathcal{L}})$-literals, A is a set of ground atoms of the form $\mathsf{atom}(t)$, and N is a set of ground literals of the form $\neg\mathsf{atom}(t)$, where t is a $T(\Sigma_{\mathcal{L}})$-term. Furthermore, let $G' := S \cup A' \cup N'$ where A' is obtained by replacing all atoms $\mathsf{atom}(t)$ in A with $t \neq \mathsf{cons}(X_0, X_1)$, N' is obtained from N by replacing all literals $\neg\mathsf{atom}(t)$ with $t = \mathsf{cons}(sk_1, sk_2)$, where t is a ground term, sk_{1^t} and sk_{2^t} are Skolem constants, and X_0, X_1 are variables. Up to adding some flat equalities in S we can also assume that the term t is a constant and hence literals of the form $t = \mathsf{cons}(sk_1, sk_2)$ are flat since t, sk_{1^t}, and sk_{2^t} are constants.

Property 5.2. *G is $Ax(\mathcal{L}_{NO})$-satisfiable iff $S \cup N'$ is $(\{(1), (2)\} \cup A')$-satisfiable.*

This is an immediate consequence of Property 5.1 and of Skolemization.

We are left with the problem of building a satisfiability procedure for theories axiomatised by the finitely many axioms in $(\{(1), (2)\} \cup A')$. To this end, we apply our two step methodology to such a set of axioms.

Lemma 5.1. *Let S be a finite set of flat $T(\Sigma_{\mathcal{L}})$-literals. The clauses occurring in the saturations of $S \cup \{(1), (2)\} \cup A'$ by $\mathcal{SP}$ can only be the* (i) *empty clause,* (ii) *ground flat literals, or* (iii) *literals in $\{(1), (2)\} \cup A'$.*

Proof. The proof is by induction on the length of the derivations. The set $\{(1),(2)\} \cup A'$ is saturated w.r.t. $\mathcal{SP}$. The base case is simple and therefore is omitted. By the induction hypothesis there are three types of clauses to consider. For inferences with *Reflection* the result is obvious. *Deletion* and *Subsumption* do not create new clauses. Here, we consider only *replacement*s between literals of type ii) and literals in A', since the other cases are covered by Lemma 4.1. A *Paramodulation* between a ground flat equality and a disequality of the form $\mathsf{cons}(X,Y) \neq c$ (where c is a constant and X, Y are variables) can only generate a ground flat disequality. □

Lemma 5.2. *Let S be a finite set of flat $T(\Sigma_{\mathcal{L}})$-literals. All the saturations of $S \cup \{(1),(2)\} \cup A'$ by $\mathcal{SP}$ are finite.*

Proof. By Lemma 5.1, we know that the saturations of $S \cup \{(1),(2)\} \cup A'$ by $\mathcal{SP}$ can only contain literals of type (i)–(iii). It is trivial to see that only a finite number of flat literals can be built out of a finite set of symbols and variables (recall that A' is a finite set). □

Theorem 5.1. *$\mathcal{SP}$ is a polynomial satisfiability procedure for theories presented by sets of axioms of the form $\{(1),(2)\} \cup A'$.*

Proof. Let n be the size of the input set of flattened literals. At most $\mathrm{O}(n^2)$ flat literals can be created by *Superposition* during saturation. The size of the current set of literals in a derivation is always bounded by $\mathrm{O}(n^2)$. Other inferences take $\mathrm{O}(n^2)$ time according to Section 3.1. Hence, overall the satisfiability procedure is polynomial. □

A satisfiability procedure for $\mathcal{L}_{NO}$ is as follows. Given as input a finite set of ground literals built out of the symbols in $\Sigma_{\mathcal{L}_{NO}}$, the procedure first replaces all the literals of the form $\mathsf{atom}(t)$ and $\neg\mathsf{atom}(t')$ with $t \neq \mathsf{cons}(X_0, X_1)$ and $t' = \mathsf{cons}(sk_{1'}, sk_{2'})$ where t, t' are $T(\Sigma_{\mathcal{L}})$-terms, X_0, X_1 are distinct variables, and $sk_{1'}, sk_{2'}$ are Skolem constants. Let $S \cup A$ be the resulting set of literals, where S is the set of $T(\Sigma_{\mathcal{L}})$-literals and A is a set of $T(\Sigma_{\mathcal{L}}, \mathcal{V})$-literals, where $\mathcal{V}$ is the set of distinct variables. Then, it invokes the procedure of Theorem 5.1 to check the $(\{(1),(2)\} \cup A)$-satisfiability of S.

6. Theory of encryption

Let $\Sigma_{\mathcal{N}}$ be a signature containing the function symbols $\mathsf{enc}/2$, $\mathsf{dec}/2$, respectively, denoting encryption and decryption with a symmetric key. The encryption operation takes a *clear-text* and a key and produces a *cipher-text*. The decryption operation inverses the encryption by extracting a clear-text from a cipher-text using the same key. The axioms, denoted with $Ax(\mathcal{N})$, that are satisfied by the operators are the following:

$$\mathsf{enc}(\mathsf{dec}(X,Y),Y) = X, \tag{6}$$

$$\mathsf{dec}(\mathsf{enc}(X,Y),Y) = X. \tag{7}$$

Our satisfiability approach is also relevant for this theory.

Lemma 6.1. *Let S be a finite set of flat $T(\Sigma_{\mathcal{N}})$-literals. Then, the clauses occurring in the saturations of $S \cup Ax(\mathcal{N})$ can only belong to the following categories:* (i) *the empty clause,* (ii) *ground flat literals or* (iii) *literals in $Ax(\mathcal{N})$*

The proof of this lemma is exactly the same as in the previous section. For instance a *Superposition* of $\mathsf{enc}(\mathsf{dec}(X,Y),Y) = X$ and $\mathsf{dec}(a,b) = c$ generates $\mathsf{dec}(c,b) = a$.

Theorem 6.1. *$\mathcal{SP}$ is a polynomial satisfiability procedure for $\mathcal{N}$.*

The proof of this theorem can easily be adapted from the one in the previous section thereby obtaining a quadratic complexity result.

7. The theory of arrays with extensionality

Let $\mathcal{A}_s^e$ be the many-sorted theory with sorts ELEM, INDEX, and ARRAY, with function symbols store and select of type ARRAY, INDEX, ELEM $\rightarrow$ ARRAY and ARRAY, INDEX $\rightarrow$ ELEM, respectively.[4] Furthermore, let $Ax(\mathcal{A}_s^e)$ be the set of axioms obtained by adding the following axioms to $\mathcal{E}$:

$$\mathsf{select}(\mathsf{store}(A,I,E),I) = E, \tag{8}$$

$$I \neq J \Rightarrow \mathsf{select}(\mathsf{store}(A,I,E),J) = \mathsf{select}(A,J), \tag{9}$$

$$\forall I.(\mathsf{select}(A,I) = \mathsf{select}(B,I)) \Rightarrow A = B, \tag{10}$$

where A and B are variables of sort ARRAY, I and J are variables of sort INDEX, and E is a variable of sort ELEM. $\Sigma_{\mathcal{A}_s^e}$ denotes a signature containing the function symbols select, store, and a finite set of function symbols s.t. **if f is a function symbol of type $s_0,\ldots,s_{n-1} \rightarrow s_n$ distinct from** select **and** store**, then s_i is either** INDEX **or** ELEM**, for all $i = 0,1,\ldots,n$ and $n \geqslant 1$**. Furthermore, we assume that $\Sigma_{\mathcal{A}_s^e}$ admits at least one ground term for each sort, i.e., it is a sensible signature. Finally, let $Ax(\mathcal{A}_s) := \{(8),(9)\}$ and $Ax(\mathcal{A}_s^e) := Ax(\mathcal{A}_s) \cup \{(10)\}$.

Lemma 7.1. *Let S be a set of $T(\Sigma_{\mathcal{A}_s^e})$-literals and let S' be obtained from S by replacing all the inequalities of the form $t \neq t'$ with $\exists i.\mathsf{select}(t,i) \neq \mathsf{select}(t',i)$, where t and t' are terms of sort* ARRAY*. Then S is $\mathcal{A}_s^e$-satisfiable iff S' is $\mathcal{A}_s$-satisfiable.*

Proof. We must show that $S \cup \mathcal{A}_s^e$ is satisfiable iff $S' \cup \mathcal{A}_s$ is or, equivalently, that $S \cup Ax(\mathcal{A}_s^e)$ is satisfiable iff $S' \cup Ax(\mathcal{A}_s)$ is. The 'only if' case is trivial. For the 'if' case, we must construct a model satisfying axiom (10) starting from a model which is not required to do so. Formally, let I be a (many-sorted) model of $S' \cup Ax(\mathcal{A}_s)$. We define the binary relation $\sim$ over ARRAY^I to hold whenever $\mathsf{select}^I(a,i) = \mathsf{select}^I(b,i)$ for all $i \in$ INDEX^I, and we define $\sim$ over the INDEX^I and ELEM^I to be the identity relation. We now show that $\sim$ is a $\Sigma_{\mathcal{A}_e^s}$-congruence. It is clearly an equivalence. To prove that $\sim$ is a congruence it remains to show that if $a \sim b$, then $\mathsf{store}^I(a,i,e) \sim \mathsf{store}^I(b,i,e)$ for all

[4] Notice that the use of sorts allows us to avoid problematic terms such as store(a, store(a, i, e), select(a, store(a, i, e))).

$i \in \text{INDEX}^I$ and $e \in \text{ELEM}^I$. (The case for select trivially follows from the definition of $\sim$. For a function symbol f in $\Sigma_{\mathcal{A}_e^s}$ distinct from select and store, congruence immediately follows from the definition of $\sim$, the properties of identity, and the requirement on the type of f, namely, if f has type $s_0, \ldots, s_{n-1} \to s_n$ then s_i is either INDEX or ELEM, for all $i = 0, 1, \ldots, n$ and $n \geqslant 1$.) Let us assume that $a \sim b$ but $\mathsf{store}^I(a,i,e) \not\sim \mathsf{store}^I(b,i,e)$ for some $i \in \text{INDEX}^I$ and $e \in \text{ELEM}^I$, i.e., that $\mathsf{select}^I(\mathsf{store}^I(a,i,e),k) \neq \mathsf{select}^I(\mathsf{store}^I(b,i,e),k)$ for some $i,k \in \text{INDEX}^I$ and $e \in \text{ELEM}^I$. There are two cases to consider. If $k = i$ then, since I is a model of (8), we can conclude that $e \neq e$, a contradiction. Otherwise (i.e., if $k \neq i$), since I is a model of (9), we can conclude that $\mathsf{select}^I(a,k) \neq \mathsf{select}^I(b,k)$. This is in contradiction with the assumption $a \sim b$. To conclude the proof, it is sufficient to check that $I' = I/\sim$ is a model of $S' \cup Ax(\mathcal{A}_s^e)$. □

Theorem 7.1. *Let S be a set of $T(\Sigma_{\mathcal{A}_s^e})$-literals and S' be obtained from S by replacing all the inequalities of the form $t \neq t'$ with $\mathsf{select}(t, sk_{t,t'}) \neq \mathsf{select}(t', sk_{t,t'})$, where t and t' are terms of sort* ARRAY*, and $sk_{t,t'}$ is a Skolem constant of type* INDEX*. Then S is $\mathcal{A}_s^e$-satisfiable iff S' is $\mathcal{A}_s$-satisfiable.*

Proof. The theorem readily follows from Lemma 7.1 and basic properties of skolemization. □

Let us assume as given a satisfiability procedure for $\mathcal{A}_s$. Then, a **satisfiability procedure for the theory of arrays with extensionality** $\mathcal{A}_s^e$ is as follows. Given as input a finite set S of $T(\Sigma_{\mathcal{A}_s^e})$-literals, the procedure first replaces every occurrence of literals of the form $t \neq t'$ with $\mathsf{select}(t, sk_{t,t'}) \neq \mathsf{select}(t', sk_{t,t'})$, where t and t' are terms of sort ARRAY, and $sk_{t,t'}$ is a Skolem constant of type INDEX (notice that, for each $t \neq t'$, $sk_{t,t'}$ is a constant, distinct from all the others in the signature). Then, it feeds the resulting set of literals to a satisfiability procedure for $\mathcal{A}_s$.

It is worth noticing that our satisfiability procedure can be straightforwardly generalised to multi-dimensional arrays if we view them as arrays of arrays.

7.1. A satisfiability procedure for $\mathcal{A}_s$

We are left with the task of developing a satisfiability procedure for $\mathcal{A}_s$. In order to do this, we apply our two step methodology to the unsorted theory $\mathcal{A}$ whose axioms $Ax(\mathcal{A})$ are obtained from $Ax(\mathcal{A}_s)$ by forgetting sorts. We shall assume that the ordering $\succ$ **is s.t. any term that contains** select **or** store **is $\succ$-bigger than all ground terms not containing them; moreover, all non-constant symbols are greater than the constant ones**. Using an LPO ordering [14], this can easily be ensured by a suitable precedence relation.

Lemma 7.2. *Let S be a finite set of flat $T(\Sigma_{\mathcal{A}})$-literals. The clauses occurring in the saturations of $S \cup Ax(\mathcal{A})$ by $\mathcal{SP}$ can only be:*

(i) *the empty clause;*
(ii) *the axioms in $Ax(\mathcal{A})$;*
(iii) *ground flat literals;*
(iv) *clauses of the form $t_1 \bowtie t_2 \vee c_1 = c_1' \vee \cdots \vee c_n = c_n'$ where $c_1, c_1', \ldots, c_n, c_n'$ ($n \geqslant 0$) are individual constants, $\bowtie$ is either $=$ or $\neq$, and t_j ($j = 1, 2$) is either an individual constant or a term of the form $\mathsf{select}(c, c_i)$ (for some individual constants c and c_i);*

(v) *clauses of the form* $\mathsf{select}(c,x) = \mathsf{select}(c',x) \vee c_1 = k_1 \vee \cdots \vee c_n = k_n$, *where* k_i (for $i = 1,\ldots,n$) *is either the variable* x *or is one among the individual constants* $c, c_1, \ldots, c_n$ ($n \geqslant 0$).

Proof. The proof is by induction on the length of the derivations. The base case is simple and therefore omitted. By the induction hypothesis there are five types of clauses produced after n inference steps: (i)–(v). For inferences with *Reflexion* or *Factoring* on one clause the result is obvious. *Deletion* and *Subsumption* do not create new clauses. For the sake of brevity, let *replacement* be either a *Superposition* or a *Paramodulation* step. Let us consider inference steps involving two clauses. There are several cases to consider according to the categories the clauses belong to:

(ii)–(ii): A *Superposition* can be applied to the axioms in $Ax(\mathcal{A})$ but it generates the trivial clause $i = i \vee \mathsf{select}(a,i) = e$ which is immediately eliminated by *Deletion*. No new clause can be produced this way.

(ii)–(iii): A *Superposition* from a flat equality into axiom (8) produces a ground flat equality, i.e., a clause of type (iii), whereas a *Superposition* into axiom (9) produces a clause of type (v).

(iii)–(iii): The only possible inference is *Simplification* or *Paramodulation* between a ground flat equality and a ground flat literal. It produces only ground flat literals, i.e., a clause of type (iii).

(iii)–(iv): A *Simplification* or a *replacement* produces a clause of type (iv).

(iii)–(v): A *replacement* produces a clause of type (iv) or (v).

(iv)–(iv): A *replacement* produces a clause of type (iv).

(iv)–(v): A *replacement* produces a clause of type (iv).

(v)–(v): A *replacement* produces a clause of type (v).

There are no possible inference between axioms and clauses of type (iv) or (v). □

It is interesting to notice that clauses of type (v) in Lemma 7.2 closely resemble to the partial equations introduced in [38].

Lemma 7.3. *Let* S *be a finite set of flat* $T(\Sigma_{\mathcal{A}})$*-literals. All the saturations of* $S \cup Ax(\mathcal{A})$ *by* $\mathcal{SP}$ *are finite.*

The proof of this lemma is analogous to that of Lemma 4.2 and therefore it is omitted.

Theorem 7.2. $\mathcal{SP}$ *is an* $O(2^{n^2})$ *satisfiability procedure for* $\mathcal{A}$.

Proof. Let n be the size of the input set of flattened literals. At most $O(2^{n^2})$ clauses can be generated by saturation. Hence the decision procedure takes time $O(2^{n^2})$. □

Since we assume a sensible signature $\Sigma_{\mathcal{A}_s^e}$, the relationship between the sorted theory $\mathcal{A}_s$ and the unsorted theory $\mathcal{A}$ is obvious.

Lemma 7.4. *Let* S *be a conjunction of ground literals, then* S *is* $\mathcal{A}_s$*-satisfiable iff it is* $\mathcal{A}$*-satisfiable.*

It is worth noticing that we can easily extend our satisfiability procedure for $\mathcal{A}$ to support constant arrays (as done in [38]). In fact, this amounts to extending the set of axioms $Ax(\mathcal{A})$ with

the following axiom: $\mathsf{select}(\mathsf{const}(E), I) = E$. Then, it is easy to see that the resulting set of axioms is saturated w.r.t. $\mathcal{SP}$ and the proofs in this section can be straightforwardly adapted to take into account the new axiom.

A remark about the computational complexity of the satisfiability procedure for $\mathcal{A}_s^e$ is in order. Its worst-case time is that of the satisfiability procedure for $\mathcal{A}$ (i.e., $O(2^{n^2})$), since the size of the set of input literals obtained by using Theorem 7.1 is $O(n)$. Finally, it is interesting to notice that the satisfiability procedure for $\mathcal{A}$ is similar to the algorithm described in [29] for the same theory.

8. A theory of finite sets with extensionality

We develop a satisfiability procedure for a theory of finite sets along the same line of Section 7. We notice that the satisfiability procedure for arrays with extensionality can be indirectly used as a satisfiability procedure for finite sets with extensionality if we represent finite sets by characteristic functions which, in turn, can be encoded by arrays of booleans indexed over the elements of the sets. We do not follow this possibility here since we want to demonstrate the flexibility of our methodology by showing how easy it is to adapt the proofs for theories presented by sets of axioms with similar shape.

Let $\mathcal{S}_s^e$ be the many-sorted theory with sorts ELEM, BOOL, and SET, with function symbols ins of type ELEM, SET $\rightarrow$ SET, mem of type ELEM, SET $\rightarrow$ BOOL, and the constant true of sort BOOL. Furthermore, let $Ax(\mathcal{S}_s^e)$ be obtained by adding to $\mathcal{E}$ the following axioms:

$$\mathsf{mem}(E, \mathsf{ins}(E, S)) = \mathsf{true}, \tag{11}$$

$$E \neq F \Rightarrow \mathsf{mem}(E, \mathsf{ins}(F, S)) = \mathsf{mem}(E, S), \tag{12}$$

$$\forall E.(\mathsf{mem}(E, S_1) = \mathsf{mem}(E, S_2)) \Rightarrow S_1 = S_2, \tag{13}$$

where S, S_1 and S_2 are variables of sort SET, and E is a variable of sort ELEM. $\Sigma_{\mathcal{S}_s^e}$ denotes a signature containing the function symbols mem, ins, and a finite set of function symbols such that if f is a function symbol of type $s_0, \ldots, s_{n-1} \rightarrow s_n$ distinct from mem and ins, then s_i is ELEM, for all $i = 0, 1, \ldots, n$ and $n \geqslant 1$. We also assume that $\Sigma_{\mathcal{S}_s^e}$ admits at least one ground term for each sort. Finally, let $Ax(\mathcal{S}_s) := \{(11), (12)\}$ and $Ax(\mathcal{S}_s^e) := Ax(\mathcal{S}_s) \cup \{(13)\}$.

Theorem 8.1. *Let S be a set of $T(\Sigma_{\mathcal{S}_s^e})$-literals and S' be obtained from S by replacing all the inequalities of the form $t \neq t'$ with $\mathsf{mem}(sk_{t,t'}, t) \neq \mathsf{mem}(sk_{t,t'}, t')$, where t and t' are terms of sort* SET, *and $sk_{t,t'}$ is a Skolem constant of sort* ELEM. *Then S is $\mathcal{S}_s^e$-satisfiable iff S' is $\mathcal{S}_s$-satisfiable.*

The proof of this fact is a straightforward adaptation of the proof of Theorem 7.1 and therefore it is omitted.

Given as input a finite set S of $T(\Sigma_{\mathcal{S}_s^e})$-literals, a satisfiability procedure for $\mathcal{S}_s^e$ replaces every occurrence of literals of the form $t \neq t'$ with $\mathsf{mem}(sk_{t,t'}, t) \neq \mathsf{mem}(sk_{t,t'}, t')$, where t and t' are terms of sort SET, and $sk_{t,t'}$ is a Skolem constant of sort ELEM. Then, it feeds the resulting set of literals to a satisfiability procedure for $\mathcal{S}_s$, which we assume available.

8.1. A satisfiability procedure for $\mathcal{S}_s$

We apply our two step methodology to the axioms in $Ax(\mathcal{S})$ which are the axioms in $Ax(\mathcal{S}_s)$ with sort information neglected in order to derive a satisfiability procedure for $\mathcal{S}_s$. We shall assume that the ordering $\succ$ is s.t. any term that contains mem or ins is $\succ$-bigger than all ground terms not containing them; moreover, all non-constant symbols are greater than the constant ones.

The proof of the following result is an easy adaptation of Theorem 7.2 and Lemma 7.4 and hence it is omitted.

Theorem 8.2. *$\mathcal{SP}$ is an $\mathrm{O}(2^{n^2})$ satisfiability procedure for $\mathcal{S}_s$.*

Two observations are in order. First, the computational complexity of the satisfiability procedure for $\mathcal{S}_s^e$ is $\mathrm{O}(2^{n^2})$. Second, the procedure developed here can form the core tool for theorem proving in larger fragments of finite set theory.

Example 8.1. Consider the problem of showing that the finite set $\{a, b, c\}$ is equal to the finite set $\{b, c, a\}$. This can be checked by showing the $\mathcal{S}_s^e$-unsatisfiability of the following disequality:

$$\mathsf{ins}(a, \mathsf{ins}(b, \mathsf{ins}(c, mty))) \neq \mathsf{ins}(b, \mathsf{ins}(c, \mathsf{ins}(a, mty))), \tag{14}$$

where *mty* is a constant of sort SET (intuitively denoting the empty set). By flattening and Theorem 8.1, the task of checking the $\mathcal{S}_s^e$-unsatisfiability of (14) can be reduced to proving the $\mathcal{S}$-unsatisfiability of the following set of literals:

$$\left\{\begin{array}{c} \mathsf{ins}(c, mty) = c_0,\ \mathsf{ins}(b, c_0) = c_1,\ \mathsf{ins}(a, c_1) = c_2, \\ \mathsf{ins}(a, mty) = c_4,\ \mathsf{ins}(c, c_4) = c_5,\ \mathsf{ins}(b, c_5) = c_6, \\ \mathsf{mem}(sk_e, c_2) = c_3,\ \mathsf{mem}(sk_e, c_6) = c_7 \\ c_3 \neq c_7, \end{array}\right\} \tag{15}$$

where sk_e is a Skolem constant, and $c_1, \ldots, c_7$ are the constants introduced by flattening. The $\mathcal{S}$-unsatisfiability of (15) can readily be established by using Theorem 8.2; the proof roughly amounts to compare sk_e with the elements a, b, and c for equality and to derive the empty clause from $c_3 \neq c_7$ in each case.

9. A combination of theories

To emphasise the flexibility of our approach, we show how easy it is to combine the satisfiability procedures for the theory of lists *á la* Shostak (cf. Section 4) and the theory of arrays (cf. Section 7.1).

Let $\Sigma_{\mathcal{U}}$ be a signature containing the function symbols select/2, store/3, car/1, cdr/1, and cons/2. Let $Ax(\mathcal{U})$ be the set of axioms obtained by adding to $\mathcal{E}$ the axioms in $Ax(\mathcal{A}) \cup Ax(\mathcal{L}_{Sh})$. We shall assume that the simplification ordering $\succ$ (total on ground terms) satisfies the requirements of Section 7.1.

Lemma 9.1. *Let S be a finite set of ground flat $T(\Sigma_{\mathcal{U}})$-literals. The clauses occurring in the saturations of $S \cup Ax(\mathcal{U})$ by $\mathcal{SP}$ can only be of the type* (i), (iii), (iv), *and* (v) *given in Lemma* 7.2, *of the types* (i), (ii), *and* (iv) *given in Lemma* 4.1, *or elements of $Ax(\mathcal{U})$.*

Proof. The set $Ax(\mathcal{U})$ is saturated w.r.t. $\mathcal{SP}$. Hence the proof is as that of Lemma 4.1 and of Lemma 7.2. □

Lemma 9.2. *Let S be a finite set of ground flat $T(\Sigma_{\mathcal{U}})$-literals. All the saturations of $S \cup Ax(\mathcal{U})$ by $\mathcal{SP}$ are finite.*

The proof of this lemma is analogous to that of Lemma 5.2.

Theorem 9.1. *$\mathcal{SP}$ is an $O(2^{n^3})$ satisfiability procedure for $\mathcal{U}$.*

We notice that the derived satisfiability procedure is an instance of the Nelson–Oppen combination schema [31] since it amounts to propagating equalities between constants derived in one theory to the other.

10. The theory of homomorphism

In this section, we present an adaptation of the Knuth–Bendix completion procedure [23] to work modulo the theory of homomorphism. The completion process always terminates for ground equations and gives a decision procedure for this theory. Notice that the word problem for ground associative-commutative (AC) theories is decidable [28] but for ground AC + distributivity is undecidable [27]. A direct modification of the proof of this last result would show that ground AC + homomorphism is undecidable too.

Let $\Sigma_{\mathcal{H}}$ be a signature containing the unary function symbol h and let $\mathcal{H}$ be the theory obtained by adding instances of the following axiom schema, denoted with $Ax(\mathcal{H})$, to $\mathcal{E}$:

$$\mathsf{h}(f(x_1,\ldots,x_n)) = f(\mathsf{h}(x_1),\ldots,\mathsf{h}(x_n)), \tag{16}$$

where f is any n-ary function symbol ($n > 0$) in a subset Σ' of $\Sigma_{\mathcal{H}} \setminus \{\mathsf{h}\}$. We want to decide the $\mathcal{H}$-unsatisfiability of the set of ground literals ψ.

Example 10.1. $\{\mathsf{h}(c) = c', \mathsf{h}(c') = c, f(c,c') = \mathsf{h}(\mathsf{h}(a)), \mathsf{h}(\mathsf{h}(\mathsf{h}(a))) \neq a,\ f(c',c) = a\}$ is $\mathcal{H}$-unsatisfiable.

By Lemma 3.1, **we can assume that ψ is a set of flat literals**. Our decision procedure consists of two steps. First, we complete the set of ground equalities in ψ modulo $\mathcal{H}$ in order to get a rewrite system R. Second, for each inequality $s \neq t$ in ψ, we compute the normal form $s\downarrow_R$ of s and the normal form $t\downarrow_R$ of t (w.r.t. R). Then, if there exists an inequality $s' \neq t'$ in ψ s.t. $s'\downarrow_R$ is identical to $t'\downarrow_R$, ψ is $\mathcal{H}$-unsatisfiable; otherwise, ψ is $\mathcal{H}$-satisfiable.

A standard completion procedure would not work in the first step. In order to avoid the classical orientation problems which are frequently the source of failure in completion, we apply a

variant of Knuth–Bendix ordering designed in such a way that every derived critical pair is ground. This is not sufficient since the standard completion would now diverge by generating infinitely many rules. To solve this problem we treat every rule as a rule scheme and compute the minimal critical pairs (for a well-chosen ordering) between all instances of these rule schemes. This computation is by reduction to Presburger arithmetic. Then finally we notice that only finitely many new rules can be derived that way.

10.1. Orientation

We introduce an ordering over ground terms which allows us to orient equalities as rewrite rules in such a way that a superposition between a ground equality and an equality in $Ax(\mathcal{H})$ can only generate a ground equality.

We first define a weight function on the symbols in $\Sigma_{\mathcal{H}}$, denoted with $[e]$ where e is in $\Sigma_{\mathcal{H}}$: $[c] = 1$, for each constant symbol c in $\Sigma_{\mathcal{H}}$; $[\mathsf{h}] = 0$; and $[f] = 1$, for f in $\Sigma_{\mathcal{H}}$ s.t. f is not a constant and f is not h. The weight of a ground term t, denoted with $[t]$, is the sum of the weight of the symbols (of $\Sigma_{\mathcal{H}}$) occurring in it. Then, we consider a total precedence $\succ$ on symbols s.t. $\mathsf{h} \succ f \succ c$, for all constant symbol c and all non-constant symbol f distinct from h of $\Sigma_{\mathcal{H}}$. In the following $f^0(t)$ stands for t and $f^n(t)$ abbreviates $f(f^{n-1}(t))$ for $n \geqslant 1$, where f is a unary function symbol and t is any term. The ordering on ground terms we shall use is defined as follows (similarly to the Knuth–Bendix ordering [23]): $s \succ t$ iff

1. $[s] > [t]$ or
2. $[s] = [t]$, s is of the form $f(s_1, \ldots, s_m)$, t is of the form $g(t_1, \ldots, t_n)$, and one of the following condition holds:
   2.1. $f \succ g$,
   2.2. $f = g$, $m = n$ and $(s_1, \ldots, s_m) \succ\succ_{lex} (t_1, \ldots, t_m)$ (where $\succ\succ_{lex}$ denotes the lexicographic extension of $\succ$).

Lemma 10.1. *The relation $\succ$ is transitive, irreflexive, and monotonic (i.e., $s \succ t$ implies $f(\ldots, s, \ldots) \succ f(\ldots, t, \ldots)$, where f is in $\Sigma_{\mathcal{H}}$). Furthermore, $\succ$ is well-founded and it satisfies:*

- *$f(c_1, \ldots, c_n) \succ \mathsf{h}^i(c_{n+1})$ for all $i \geqslant 0$, all f that are not constants and are different from h,*
- *$\mathsf{h}(f(x_1, \ldots, x_n)) \succ f(\mathsf{h}(x_1), \ldots, h(x_n))$ for all ground terms x_i $(i = 1, \ldots, n)$, and*
- *$\mathsf{h}^i(c) \succ \mathsf{h}^j(c')$ for all $i > j$ and for all constants c, c' in $\Sigma_{\mathcal{H}}$.*

Proof. The lemma is proved in exactly the same way as for the Knuth–Bendix ordering [23]. □

We denote by $l \to r$ the rule obtained by orienting an equality $l = r$ when $l \succ r$. Given a rewrite system R, we shall sometimes write $s \downarrow_R t$ to express that t is the normal form of s by R.

10.2. Computation of critical pairs

Now, we are in the position to orient the equalities in ψ by means of the ordering $\succ$ defined in Section 10.1 and to perform a completion on the resulting set of rewrite rules using superposition rules. Unfortunately, with a naive approach, the number of rules generated by completion would

be infinite. For instance, from $\mathsf{h}(c) = c$, $f(c, c') = c$, and $Ax(\mathcal{H})$ we can generate $f(c, \mathsf{h}^n(c')) = c$ for $n \geqslant 0$. To cope with this problem, we will consider any rewrite rule r as a rule scheme (denoted $Gen(r, R)$ or $Gen(r)$ and defined below) and we compute all superpositions between instances of two rule schemes in one step by using a special purpose inference rule (cf. *Homomorphism* rule below).

Some preliminary definitions and lemmas are mandatory. We define an *f-term* as a ground term with f as root symbol and for which the only other occurrences of non-constant function symbols are occurrences of h, where f can be any symbol in $\Sigma_{\mathcal{H}}$ (in particular, f can possibly be h). We define an *f-rule* as a rewrite rule with an f-term as left-hand side and an h-term or a constant symbol as right-hand side. For instance $f(c, \mathsf{h}^2(c'))$ is an f-term and $f(c, \mathsf{h}^2(c')) = \mathsf{h}^3(c)$ or $f(c, \mathsf{h}^2(c')) = c$ is an f-rules. Examples of h-rules are $\mathsf{h}^2(c') = c$ or $\mathsf{h}^2(c') = \mathsf{h}(c)$.

In the following, let R_{h} be a convergent set of h-rules. We recall that Σ' is the subset of $\Sigma_{\mathcal{H}} \setminus \{\mathsf{h}\}$ such that if f of arity n is in Σ', then $\mathsf{h}(f(x_1, \ldots, x_n)) = f(\mathsf{h}(x_1), \ldots, \mathsf{h}(x_n))$ is in $Ax(\mathcal{H})$.

Lemma 10.2. *The set $R_{\mathsf{h}} \cup \{\mathsf{h}(f(x_1, \ldots, x_n)) = f(\mathsf{h}(x_1), \ldots, \mathsf{h}(x_n)) | f \in \Sigma'\}$ is convergent (we shall denote it by $R_{\mathsf{h}} \cup H$).*

Lemma 10.3. *If we consider unary terms as words (for instance $\mathsf{h}^j(c)$ as $\mathsf{h}^j c$) then the set of ancestors $\{w | w \to_{R_{\mathsf{h}}}^* w'\}$ of a term w' by R_{h} is a context-free language.*

Proof. Let us build a grammar for the ancestors. For each constant c we introduce a non-terminal X_c and a grammar production $X_c \triangleright c$. For each rule $h^j(c) \to h^i(c')$ we introduce the grammar production $X_{c'} \triangleright h^{j-i} X_c$ (recall that we always have $j \geqslant i$ by the chosen ordering). Note that (by induction on the length of the derivation) $h^j(c) \to^* h^i(c')$ iff $X_{c'} \triangleright^* h^{j-i}(c)$. □

Lemma 10.4. *Given constants c, c' and two h-terms $h^j(c), h^i(c')$, the set $\{n | n \in N$, such that $h^n(h^j(c)) \to_{R_{\mathsf{h}}}^* h^i(c')\}$ is linear, i.e., the union of a finite set of non-negative integers and a finite set of arithmetic sequences. We denote it by $P_{j,c,i,c'}$.*

Proof. The set of h-terms with constant c is obviously regular. Hence the set of $\mathsf{h}^n \mathsf{h}^j c$ that reduces to $\mathsf{h}^i c'$ is the intersection of a regular language $\mathsf{h}^* \mathsf{h}^j c$ with a context-free language and therefore context-free. The set of lengths of words of a context-free language is linear. (For details, see [39, Exercise 6.8 at page 142].) □

Let J be the set of constants that occur in a left-hand side of R_{h}. If $c \in J$ we say that c is *bounded* (in R_{h}).

Lemma 10.5. *Given an h-term $\mathsf{h}^j(c)$ and two constants c, c' s.t. c' is not bounded, the set $\{n | \exists i \in N, \mathsf{h}^n(\mathsf{h}^j(c)) \to_{R_{\mathsf{h}}}^* \mathsf{h}^i(c')\}$ is an interval $[u, \infty]$ denoted by $P_{j,c,\_,c'}$.*

Proof. Note that $\mathsf{h}^i(c')$ is R_{h}-irreducible. If there exists u, v with $\mathsf{h}^v(\mathsf{h}^j(c)) \downarrow_{R_{\mathsf{h}}} \mathsf{h}^u(c')$ then for all $g \in N$ we have $\mathsf{h}^{v+g}(\mathsf{h}^j(c)) \to_{R_{\mathsf{h}}}^* \mathsf{h}^{u+g}(c')$. Note that we consider $[\infty, \infty]$ as the empty set. □

Given an f-rule $r : f(t_1, \ldots, t_n) \to t_{n+1}$, we define $\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H}$ to be the rule $(\mathsf{h}^n(f(t_1, \ldots, t_n)) \downarrow_{R_{\mathsf{h}} \cup H}) \to (\mathsf{h}^n(t_{n+1}) \downarrow_{R_{\mathsf{h}} \cup H})$. By the convergence of $R_{\mathsf{h}} \cup H$ this is well defined.

Definition 10.1. For $f \in \Sigma'$, $Gen(r, R_{\mathsf{h}})$ is the set $\{\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H} | n \in N\}$ where r denotes any f-rule $f(t_1, \ldots, t_n) \to t_{n+1}$. For $f \notin \Sigma'$ we define $Gen(r, R_{\mathsf{h}}) = \{r\}$. We shall omit the argument R_{h} in Gen when it is clear from the context.

Now, we derive a finite description for $Gen(r, R_{\mathsf{h}})$. We first classify the elements in $Gen(r)$ according to their bounded arguments. More specifically we introduce the equivalence relation $\sim$ on f-rules in $Gen(r)$:

Definition 10.2. Given two normalised (by R_{h}) rules

$$\begin{array}{lll} r_1 & : & f(\mathsf{h}^{l_1}(c_1), \ldots, \mathsf{h}^{l_n}(c_n)) \to \mathsf{h}^{l_{n+1}}(c_{n+1}) \quad \text{and} \\ r_2 & : & f(\mathsf{h}^{j_1}(d_1), \ldots, \mathsf{h}^{j_n}(d_n)) \to \mathsf{h}^{j_{n+1}}(d_{n+1}), \end{array}$$

such that $r_1, r_2 \in Gen(r)$, we have $r_1 \sim r_2$ iff for all k, $c_k = d_k$ and for all $c_k \in J$, $l_k = j_k$.

For instance, if $R_{\mathsf{h}} = \{\mathsf{h}(c) \to c\}$ then

$$(g(\mathsf{h}^3(c'), c) \to \mathsf{h}^2(c')) \sim (g(\mathsf{h}^2(c'), c) \to \mathsf{h}^3(c')).$$

We have the following lemma, whose proof is simple and therefore omitted.

Lemma 10.6. *The equivalence $\sim$ defined on $Gen(r)$ has finite index* (*i.e., the number of classes is finite*).

Let us say that in the rules $f(t_1, \ldots, t_n) \to t_{n+1}$ and $f(s_1, \ldots, s_n) \to s_{n+1}$, t_i and s_i $(1 \leqslant i \leqslant n+1)$ are corresponding arguments of these rules. We are now in the position to give a finite representation for the equivalence class of a rule r' in $Gen(r)$. This equivalence class (to be denoted by $C_{r,r'}$) will contain all rules $\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H}$ that are equivalent to r'. Hence n has to be taken in such a way that each argument of f in $\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H}$ has the same constant symbol than the corresponding argument of r'. Moreover if an argument t of r' has a bounded constant then the corresponding argument s in $\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H}$ has to be identical to t. To summarise the discussion, n has to be taken in the intersection of some well-chosen sets of type P_{l_m, c_m, j_m, d_m} or $P_{l_m, c_m, \_, d_m}$ (to be denoted by $P_{r,r'}$).

Definition 10.3. Let us consider f-rules $r : f(\mathsf{h}^{l_1}(c_1), \ldots, \mathsf{h}^{l_n}(c_n)) = \mathsf{h}^{l_{n+1}}(c_{n+1})$ and $r' : f(\mathsf{h}^{j_1}(d_1), \ldots, \mathsf{h}^{j_n}(d_n)) = \mathsf{h}^{j_{n+1}}(d_{n+1})$. Then, we define $C_{r,r'} = \{r" \in Gen(r) | r' \sim r"\}$.

Let us compute $C_{r,r'}$ more explicitly. We introduce

$$P_{r,r'} = \left(\bigcap_{\substack{1 \leqslant m \leqslant n+1 \\ d_m \in J}} P_{l_m, c_m, j_m, d_m} \right) \cap \left(\bigcap_{\substack{1 \leqslant m \leqslant n+1 \\ d_m \notin J}} P_{l_m, c_m, \_, d_m} \right).$$

This set is computable since it is the case for each of its components. It has been defined so that we have:

$$C_{r,r'} = \{\mathsf{h}^n(r) \downarrow_{R_\mathsf{h} \cup H} \mid n \in P_{r,r'}\}.$$

Let $p_{r,r'}$ be the minimal element of $P_{r,r'}$. Note that $p_{r,r'}$ is also computable since it can be defined by a formula of Presburger arithmetic:

$$P_{r,r'}(x) \wedge (\forall y P_{r,r'}(y) \Rightarrow x \leqslant y).$$

We denote by $n(p,l,c,d)$ the natural number n (when it exists) such that $\mathsf{h}^p(\mathsf{h}^l(c)) \downarrow_{R_\mathsf{h}} \mathsf{h}^n(d)$. Then

$$\begin{aligned} C_{r,r'} = \{ & f(\mathsf{h}^{t_1}(d_1), \ldots, \mathsf{h}^{t_n}(d_n)) \to \mathsf{h}^{t_{n+1}}(d_{n+1}) | \text{ for } 1 \leqslant m \leqslant n+1, \\ & t_m = j_m \text{ if } d_m \in J \text{ and} \\ & t_m = p' - p_{r,r'} + n(p_{r,r'}, l_m, c_m, d_m) \text{ if } d_m \notin J, \text{ where } p' \in P_{r,r'}\}. \end{aligned}$$

We define the size of an h-rule $\mathsf{h}^a(b) \to \mathsf{h}^c(d)$ to be $a + c$. Given two f-rules their non-trivial critical pairs (when they exist) are h-rules. In that case we call *minimal non-trivial critical pair* any of these critical pairs which is minimal for the size defined above. By reduction to Presburger arithmetic, we can prove the following fact.

Lemma 10.7. *Given two f-rules r_1, r_2, the minimal non-trivial critical pairs between rules in $Gen(r_1)$ and $Gen(r_2)$, are computable.*

Proof. Consider two f-rules

$$\begin{aligned} r_1 \quad &: \quad f(\mathsf{h}^{l_{1,1}}(c_{1,1}), \ldots, \mathsf{h}^{l_{1,n}}(c_{1,n})) \to \mathsf{h}^{l_{1,n+1}}(c_{1,n+1}) \quad \text{and} \\ r_2 \quad &: \quad f(\mathsf{h}^{l_{2,1}}(c_{2,1}), \ldots, \mathsf{h}^{l_{2,n}}(c_{2,n})) \to \mathsf{h}^{l_{2,n+1}}(c_{2,n+1}). \end{aligned}$$

It is sufficient to compute the minimal non-trivial critical pairs between rules that are compatible. This amounts to checking if there are rules in C_{r_1,r'_1} and C_{r_2,r'_2} with the same left-hand sides and different right-hand sides, where r'_1, r'_2 range over a finite set of representatives for the equivalence classes of $\sim$. As above we denote for $i = 1, 2$:

$$r'_i : f(\mathsf{h}^{j_{i,1}}(d_{i,1}), \ldots, \mathsf{h}^{j_{i,n}}(d_{i,n})) \to \mathsf{h}^{j_{i,n+1}}(d_{i,n+1}),$$

$$\begin{aligned} C_{r_i,r'_i} = \{ & f(\mathsf{h}^{t_{i,1}}(d_{i,1}), \ldots, \mathsf{h}^{t_{i,n}}(d_{i,n})) = \mathsf{h}^{t_{i,n+1}}(d_{i,n+1}) | \\ & \text{for } 1 \leqslant m \leqslant n+1, \\ & t_{i,m} = j_{i,m} \text{ if } d_{i,m} \in J \text{ and} \\ & t_{i,m} = p'_i - p_{r_i,r'_i} + n(p_{r_i,r'_i}, l_{i,m}, c_{i,m}, d_{i,m}) \text{ if } d_{i,m} \notin J, \text{ where } p'_i \in P_{r_i,r'_i}\} \end{aligned}$$

We have a superposition between a rule of C_{r_1,r'_1} and one of C_{r_2,r'_2} if for all $1 \leqslant m \leqslant n$: $d_{1,m} = d_{2,m}$ and $t_{1,m} = t_{2,m}$. In particular there exists $p'_1 \in P_{r_1,r'_1}, p'_2 \in P_{r_2,r'_2}$ such that for all $m \leqslant n$ such that $d_{i,m} \notin J$ we have:

$$p'_1 - p_{r_1,r'_1} + n(p_{r_1,r'_1}, l_{1,m}, c_{1,m}, d_{1,m}) = p'_2 - p_{r_2,r'_2} + n(p_{r_2,r'_2}, l_{2,m}, c_{2,m}, d_{2,m})$$

and for the critical pair to be non-trivial we need moreover:

if $d_{1,n+1} = d_{2,n+1} \in J$ then $j_{1,n+1} \neq j_{2,n+1}$

if $d_{1,n+1} = d_{2,n+1} \notin J$ then

$$p'_1 - p_{r_1,r'_1} + n(p_{r_1,r'_1}, l_{1,n+1}, c_{1,n+1}, d_{1,n+1}) \neq p'_2 - p_{r_2,r'_2} + n(p_{r_2,r'_2}, l_{2,n+1}, c_{2,n+1}, d_{2,n+1})$$

Since finding a minimal solution (p'_1, p'_2) to the above constraints amounts to solving a formula in Presburger arithmetic, the minimal non-trivial critical pairs in R are computable. □

10.3. Completion procedure

We now give the three inference rules defining the binary transition relation over sets of equalities (denoted with $\vdash$), which models our completion procedure (modulo $\mathcal{H}$). The first is the *Deletion* rule of Fig. 2. The second is the *Simplification* rule, obtained as an instance for unit clauses of the *Simplification* rule of Fig. 2 (i.e., $E \cup \{l[s] = r, s = t\} \vdash E \cup \{l[t] = r, s = t\}$, if $l[s] \succ r$ and $s \succ t$). The third is a special purpose inference which allows us to take into account finitely many selected instances of the axioms in $Ax(\mathcal{H})$ which suffices for correctness.

$$\textit{Homomorphism}: \quad E \cup \{r_1, r_2\} \vdash E \cup \{r_1, r_2, h_1, \ldots, h_k\},$$

where the r_i are f-rules and the h_j are the minimal critical pairs of $Gen(r_1, E_{\mathsf{h}})$ and $Gen(r_2, E_{\mathsf{h}})$, and where E_{h} is the subset of all h-rules in E. We recall that by Lemma 3.1, we assume that the initial set of rules is *flat* , which means by definition that the arguments of the non-constant symbols are constants.

Lemma 10.8. *When initially given a set of flat rules, the inference rules Simplification and Homomorphism only generate equations of type*

$$f(\mathsf{h}^{i_1}(c_1), \ldots, \mathsf{h}^{i_n}(c_n)) = \mathsf{h}^{i_{n+1}}(c_{n+1})$$

or of type $\mathsf{h}^i(c) = \mathsf{h}^{i'}(c')$.

Theorem 10.1. *Completion with priority given to the rule Simplification always terminates.*

Proof. Note that any sequence of *Simplification* applications always terminates. Let $E_0, E_1, E_2 \ldots$ be an infinite derivation such that E_i is the result of applying *Homomorphism* to E_{i-1} followed by a maximal sequence of *Simplification* applications. We assume that the set of constants is $\{c_1, \ldots, c_k\}$. Let $M_j = (m_1^j, \ldots, m_k^j)$ be the exponents of h in the h-rules of E_j. That is, if there is a rule in E_j with left-hand side $\mathsf{h}^m(c_i)$ then $m_i^j = m$. Note that there are no two rules of this type for the same constant c_i (otherwise one simplifies another) and therefore the vector M_j is well-defined. When no rule exists we put ∞ as a coordinate with $n < \infty$ for all integers.

The component-wise ordering on vectors M_j is well-founded and we always have $M_j \leqslant M_{j-1}$. Hence after some finite number of steps the left-hand sides of h-rules remain the same. Also the right-hand sides of rules may be simplified but only finitely many time (the reduction relation is well-founded too). Finally after some finite number of steps the set of h-rules is constant. Note also that this subset of rules is canonical. We shall denote it by R_{h}. In particular at most one rule applies to an h-term $\mathsf{h}^n(c)$.

Homomorphism generates only h-rules. Hence after a finite number of steps, say K, it will not produce any new rule. Note that the arguments of left-hand sides of f-rules are of type $\mathsf{h}^i(c_j)$ with $i < M_K(j)$ when c_j is bounded. □

Theorem 10.2. *Let E be the final finite set of rules obtained by the terminating completion procedure above. Let R_{h} be the final set of* h *rules in E. Then, $\overline{E} \cup H$ is convergent where $\overline{E}$ is the union of all sets $Gen(r, R_{\mathsf{h}})$ for all r in E.*

Proof. Since the (possibly infinite) set of rules in $\overline{E} \cup H$ terminates it is enough to show that all critical pairs are trivial. Note that R_{h} is convergent (critical pairs are trivial) as well as $R_{\mathsf{h}} \cup Ax(\mathcal{H})$ by Lemma 10.2. We discuss the different cases. (We only consider f rules where $f \in \Sigma'$ since the other cases are simpler):

Critical pairs between an f-rule and H: Let $r \in E$ and let $r' \in Gen(r)$. Then r' is equal by definition to $\mathsf{h}^n(r) \downarrow_{R_{\mathsf{h}} \cup H}$ which is equal to

$$f(\mathsf{h}^n(t_1) \downarrow_{R_{\mathsf{h}} \cup H}, \ldots, \mathsf{h}^n(t_n) \downarrow_{R_{\mathsf{h}} \cup H}) = \mathsf{h}^n(t_{n+1} \downarrow_{R_{\mathsf{h}} \cup H}).$$

Hence by superposition with H one gets the equation:

$$f(\mathsf{h}^{n+1}(t_1) \downarrow_{R_{\mathsf{h}} \cup H}, \ldots, \mathsf{h}^{n+1}(t_n) \downarrow_{R_{\mathsf{h}} \cup H}) = \mathsf{h}^{n+1}(t_{n+1} \downarrow_{R_{\mathsf{h}} \cup H})$$

which is also equal to $\mathsf{h}^{n+1}(r) \downarrow_{R_{\mathsf{h}} \cup H}$ and therefore is reduced to a trivial one by another rule in $Gen(r)$.

Critical pairs between an f-rule and R_{h}: There are no superpositions between r' and a rule $\mathsf{h}^k(a) \to b$ since the left-hand side of r' is in normal form w.r.t R_{h}.

Critical pairs between two f-rules: If rules $R_1 : l_1 \to r_1 \in Gen(r)$ and $R_2 : l_2 \to r_2 \in Gen(r')$ have the same left-hand side l_1 then it means that $r_1 = r_2$ can be derived by superposition of r and r' and therefore it is reduced to a trivial equation by a rule in R_{h} (otherwise a non-trivial h-rule can be generated and R_{h} would not be the final set of h-rules derived by completion). □

Corollary 10.1. *Given a set of ground equations E_0, and the set E derived from E_0 by completion then $E_0 \cup H \models a = b$ iff $a \downarrow_{\overline{E} \cup H} = b \downarrow_{\overline{E} \cup H}$.*

11. The verification of a pipelined ALU

As remarked in [7], proofs of correctness of pipelined processor units are stressful benchmarks for reasoning about uninterpreted functions and axioms about memories. In the following, we show how the methodology presented in this paper allows us to use an equational prover *off-the-shelf* to check the validity of the formula encoding the correctness of the three-stage pipelined ALU described in [12].

We consider the sorted signature $\Sigma_{\mathcal{A}_s^e}$ and the set $Ax(\mathcal{A}_s^e)$ of axioms in Section 7. The interpreted function symbol select models reading from memories (such as register files) whereas the function symbol store models writing to memories. The uninterpreted function symbols in $\Sigma_{\mathcal{A}_s^e} \setminus \{\mathsf{select}, \mathsf{store}\}$ symbolically model data and instructions, abstracting from their concrete implementation. As an example, the combinational part of an ALU can be modeled by a ternary function symbol *alu* whose first argument is the operation code and the remaining two are the addresses of the operands in the register file.

In [12], both the behavioral specification of the ALU and its three-stage pipeline implementation are described by means of transition functions which are compactly described by (possibly

nested) if-then-else expressions (ite's for short). As a result, the formula expressing the fact that the implementation satisfies the specification consists of an equality between ite's (see [12] for a complete description of the modelling and verification details). Given an equality α between ite's, there are techniques to efficiently transform α into a formula in disjunctive normal form whose unsatisfiability is equivalent to the validity of α (see [18]). As a result, we are left with the problem of checking the unsatisfiability of conjunctions of literals in $\mathcal{A}_s^e$. To solve this problem, we now show how to build a satisfiability procedure for $\mathcal{A}_s^e$ along the lines of Section 7.

First of all, both the flattening step (cf. Lemma 3.1) and the preprocessing to eliminate disequalities between terms of sort ARRAY (cf. Theorem 7.1) are routine manipulation of expressions which can be automated with little effort. We are left with the problem of checking the $\mathcal{A}_s$-satisfiability of sets of flat $T(\Sigma_{\mathcal{A}_s^e})$-literals (cf. Section 7.1). We solve this problem by using the state-of-the-art equational theorem prover E [36], which implements a variant of the superposition calculus described in [3]. The calculus is slightly different from the one described in this paper (i.e., $\mathcal{SP}$), but the proofs of Section 7.1 can be adapted with minor modifications. Hence, Theorem 7.2 is the theoretical support of the claim that E is turned into a procedure checking the $\mathcal{A}_s$-satisfiability of flat $T(\Sigma_{\mathcal{A}_s^e})$-literals.

Assuming that E is fed with flat $T(\Sigma_{\mathcal{A}_s^e})$-literals only, all we need to do is to choose a suitable ordering over terms (cf. Section 7.1). We exploit the facilities of E to define an LPO ordering which extends the following precedence over function symbols: $\mathsf{select} \succ \mathsf{store} \succ alu \succ sk_1 \succ \cdots \succ sk_m \succ c_1 \succ \cdots \succ c_n$, where $sk_1, \ldots, sk_m$ are Skolem constants (cf. Theorem 7.1) and $c_1, \ldots, c_n$ are all the remaining constants occurring in the literals. Although we do not exploit any of the sophisticated features of E (such as, e.g., selection functions), the prover can establish the $\mathcal{A}_s$-unsatisfiability of all the sets of literals in a few seconds. Below, we list a subset of one such set:

$$\left\{ \begin{array}{l} src_1 \neq dest\_ex,\ dest\_ex = src_2, \\ \mathsf{store}(\mathsf{store}(\lambda_1, dest\_ex, \lambda_2), dest, \\ \qquad alu(op, \mathsf{select}(\mathsf{store}(\lambda_1, dest\_ex, \lambda_2), src_1), \\ \qquad\qquad \mathsf{select}(\mathsf{store}(\lambda_1, dest\_ex, \lambda_2), src_2))) \neq \\ \mathsf{store}(\mathsf{store}(\lambda_1, dest\_ex, \lambda_2), dest, alu(op, \mathsf{select}(\lambda_1, src_1), \lambda_2)) \end{array} \right\}, \tag{17}$$

where λ_1 abbreviates $\mathsf{store}(regfile, dest\_wb, result)$ whereas λ_2 stands for $alu(op\_ex, arg_1, arg_2)$. Since the last literal above is a disequality of the form $t \neq t'$ between terms of sort ARRAY, it is replaced with $\mathsf{select}(t, sk_{t,t'}) \neq \mathsf{select}(t', sk_{t,t'})$. Afterwards, the resulting set of literals is flattened and fed to E together with the axioms in $Ax(\mathcal{A}_s)$. The prover readily derives the empty clause, establishing the $\mathcal{A}_s$-unsatisfiability of the input set of flat literals. Finally, by Theorem 7.1, we are entitled to infer the $\mathcal{A}_s^e$-unsatisfiability of (17).

Two observations are in order. Firstly, contrary to [12] we do not undertake any significant programming effort to build the theorem prover required to undertake the correctness proof of the ALU. Instead, we used a state-of-the-art prover almost off-the-shelf. Secondly, our approach is more flexible than that of [12] to handle richer theories. For example, we can exploit for free the capabilities to handle associative-commutative function symbols available in E which can be helpful in this kind of correctness proof. To do this, the prover described in [12] must undergo a major extension.

12. Conclusions

We have shown how to apply a generic inference system to derive decision procedures for the theories of lists, encryption, arrays (with extensionality), finite sets (with extensionality), and combinations of them. A decision procedure (based on superposition) for the theory of homomorphism has been presented for the first time.

We envisage two main directions for future research. Firstly, our approach might be extended using different automated deduction techniques from e.g. [13,25]. Secondly, we want to investigate possible cross-fertilisations with techniques used in heuristic theorem provers to effectively incorporating decision procedures, see e.g. [1].

Acknowledgments

The authors would like to thank M.P. Bonacina, C. Lynch, P. Narendran, C. Ringeissen, and L. Vigneron for their careful reading and suggestions about an earlier version of this paper. We would also like to thank D. Kapur and R. Nieuwenhuis for their comments. The anonymous referees helped improving the paper.

References

[1] A. Armando, S. Ranise, A practical extension mechanism for decision procedures: the case study of universal presburger arithmetic, J. Univ. Comput. Sci. 7 (2) (2001) 124–140.
[2] A. Armando, S. Ranise, M. Rusinowitch, Uniform derivation of decision procedures by superposition, in: Computer Science Logic (CSL01), Paris, France, 10–13 September, 2001.
[3] L. Bachmair, H. Ganzinger, Rewrite-based equational theorem proving with selection and simplification, J. Logic Comput. 4 (3) (1994) 217–247.
[4] L. Bachmair, I.V. Ramakrishnan, A. Tiwari, L. Vigneron, Congruence closure modulo associativity and commutativity, in: Frontiers of Comb. Sys.'s (FroCos'2000), LNCS, vol. 1794, 2000, pp. 245–259.
[5] L. Bachmair, A. Tiwari, Abstract congruence closure and specializations, in: D.A. McAllester (Ed.), 17th CADE, LNAI, vol. 1831, 2000, pp. 64–78.
[6] D. Basin, H. Ganzinger, Automated complexity analysis based on ordered resolution, J. Assoc. Comput. Mach. 48 (1) (2001) 70–109.
[7] N. Bjørner, Integrating decision procedures for temporal verification, Ph.D. thesis, Stanford University, 1999.
[8] M.P. Bonacina, On Completion Theorem Proving, January 1991, Thesis of Dottorato di Ricerca, Dipartimento di Scienze dell'Informazione, Università di Milano, Italy.
[9] M.P. Bonacina, J. Hsiang, Completion procedures as semidecision procedures, in: M. Okada, S. Kaplan (Eds.), Proc. of the Second International Workshop on Conditional and Typed Term Rewriting Systems (CTRS-90), LNCS, vol. 516, Springer, Berlin, 1991, pp. 206–232.
[10] M.P. Bonacina, J. Hsiang, Towards a foundation of completion procedures as semidecision procedures, Theor. Comput. Sci. 146 (July) (1995) 199–242.
[11] R.S. Boyer, J.S. Moore, A Computational Logic, Academic Press, New York, 1979.
[12] J.R. Burch, D.L. Dill, Automatic verification of pipelined microprocessors control, in: D.L. Dill (Ed.), Proceedings of the Sixth International Conference on Computer-Aided Verification CAV, Standford, California, USA, vol. 818, Springer, Berlin, 1994, pp. 68–80.
[13] R. Caferra, N. Peltier, Decision procedures using model building techniques, in: CSL: 9th Workshop on Computer Science Logic, LNCS, vol. 1092, 1995.

[14] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen (Ed.), Hand. of Theoretical Comp. Science, 1990, pp. 243–320.
[15] P.J. Downey, R. Sethi, R.E. Tarjan, Variations on the common subexpression problem, J. ACM 27 (4) (1980) 758–771.
[16] H.B. Enderton, A Mathematical Introduction to Logic, Academic Press, New York, 1972.
[17] M.J.C. Gordon, T.F. Melham, Introduction to HOL: A Theorem Proving Environment for Higher Order Logic, Cambridge University Press, Cambridge, 1993.
[18] J. Goubault, J. Posegga, Bdds and automated deduction, 1994.
[19] J. Hsiang, M. Rusinowitch, On word problems in equational theories, in: T. Ottmann (Ed.), Proceedings of the Fourteenth EATCS International Conference on Automata, Languages and Programming, Karlsruhe, West Germany, LNCS, vol. 267, Springer, Berlin, 1987, pp. 54–71.
[20] G. Huet, G. Kahn, C. Paulin-Mohring, The Coq Proof Assistant: a tutorial, Technical Report 204, INRIA-Rocquencourt, 1997.
[21] G. Huet, A complete proof of correctness of the Knuth–Bendix completion algorithm, J. Comput. Syst. Sci. 23 (1) (1981) 11–21.
[22] D. Kapur, Shostak's congruence closure as completion, in: H. Comon (Ed.), Proceedings of the 8th International Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 1232, Springer, Berlin, 1997.
[23] D.E. Knuth, P.E. Bendix, Simple word problems in universal algebra, in: J. Leech (Ed.), Computational Problems in Abstract Algebra, Pergamon Press, Oxford, 1970, pp. 263–297.
[24] E. Kounalis, M. Rusinowitch, On word problems in horn theories, JSC 11 (1&2) (1991) 113–128.
[25] A. Leitsch, Deciding horn classes by hyperresolution, in: CSL: 3rd Workshop on Computer Science Logic, LNCS, 1990.
[26] C. Lynch, B. Morawska, Automatic decidability, in: Logic in Computer Science (LICS'02) 26 (2002) 7–18.
[27] C. Marché, The word problem of ACD-ground theories is undecidable, Int. J. Foundations Comput. Sci. 3 (1) (1992) 81–92.
[28] P. Narendran, M. Rusinowitch, Any ground associative-commutative theory has a finite canonical system, in: 4th RTA Conf., Como, Italy, 1991.
[29] G. Nelson, Techniques for Program Verification, Technical Report CSL-81-10, Xerox Palo Alto Research Center, June 1981.
[30] G. Nelson, D.C. Oppen, Fast decision procedures based on congruence closure, J. ACM 27 (2) (1980) 356–364.
[31] G. Nelson, D.C. Oppen, Simplification by Cooperating Decision Procedures, Technical Report STAN-CS-78-652, Stanford CS Department, April 1978.
[32] R. Nieuwenhuis, A. Rubio, Paramodulation-based theorem proving, in: A. Robinson, A. Voronkov (Eds.), Hand. of Automated Reasoning, 2001.
[33] S. Owre, J.M. Rushby, N. Shankar, PVS: a prototype verification system, in: D. Kapur (Ed.), 11th International Conference on Automated Deduction (CADE), Saratoga, NY, LNAI, vol. 607, Springer, Berlin, 1992, pp. 748–752.
[34] D.A. Plaisted, A. Sattler-Klein, Proof lengths for equational completion, Inform. Comput. 125 (2) (1996) 154–170.
[35] M. Rusinowitch, Theorem-proving with resolution and superposition, JSC 11 (1&2) (1991) 21–50.
[36] S. Schulz, System abstract: E 0.61, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Proc. of the 1st IJCAR, Siena, LNAI, vol. 2083, Springer, Berlin, 2001, pp. 370–375.
[37] R.E. Shostak, Deciding combination of theories, J. ACM 31 (1) (1984) 1–12.
[38] A. Stump, D.L. Dill, C.W. Barrett, J. Levitt, A decision procedure for an extensional theory of arrays, in: LICS'01, 2001.
[39] J.D. Ullman, A.V. Aho, J.E. Hopcroft, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading. MA, 1974.