Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 174 (2007) 27–45
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Rewrite-Based Decision Procedures

Maria Paola Bonacina[1] Mnacho Echenim[2]

Dipartimento di Informatica
Università degli Studi di Verona
Ca' Vignal 2, Strada Le Grazie 15
37134 Verona, Italy

Abstract

The rewrite-based approach to satisfiability modulo theories consists of using generic theorem-proving strategies for first-order logic with equality. If one can prove that an inference system generates finitely many clauses from the presentation $\mathcal{T}$ of a theory and a finite set of ground unit clauses, then any fair strategy based on that system can be used as a $\mathcal{T}$-satisfiability procedure. In this paper, we introduce a set of sufficient conditions to generalize the entire framework of rewrite-based $\mathcal{T}$-satisfiability procedures to rewrite-based $\mathcal{T}$*-decision procedures*. These conditions, collectively termed *subterm-inactivity*, will allow us to obtain rewrite-based $\mathcal{T}$-decision procedures for several theories, namely those of equality with uninterpreted functions, arrays with or without extensionality and two of its extensions, finite sets with extensionality and recursive data structures.

Keywords: Rewrite-based inference systems, $\mathcal{T}$-decision procedures

1 Introduction

The rewrite-based approach to satisfiability modulo theories introduced in [3] was used in [3,1] to devise decision procedures for satisfiability in several theories of data structures, including the theories of arrays and records. The idea behind this approach is to use generic theorem-proving strategies based on the superposition calculus $\mathcal{SP}$ on input sets consisting of the presentation of the considered theory $\mathcal{T}$ and ground unit clauses. Since such strategies are semi-decision procedures for first-order validity, if one can prove that they *terminate* for any set of ground unit clauses, then they are actually *decision procedures* for $\mathcal{T}$*-satisfiability*. Another feature that makes the rewrite-based approach appealing is that the combination of several theories becomes conceptually simple: if termination is preserved, it suffices to consider the union of the presentations. Preservation of termination requires to

[1] Email: `mariapaola.bonacina@univr.it`

[2] Email: `echenim@sci.univr.it`

doi:10.1016/j.entcs.2006.11.042

show that the inference system is *modular* with respect to termination. Such a modularity result was obtained in [1] by introducing the notion of *variable-inactivity*: if the combined theories are variable-inactive, a strategy based on $\mathcal{SP}$ terminates on their combination, provided it terminates on each individual theory. As far as efficiency is concerned, contrary to the common expectation that a generic theorem-prover would be outperformed by more specialized systems such as CVC ([6,15]) or CVC Lite ([4]), the experimental results of [1] showed that this is not the case, and that such procedures are very efficient on several problems.

The next step is to investigate how to generalize the rewrite-based approach to *$\mathcal{T}$-decision problems*, or deciding $\mathcal{T}$-satisfiability of quantifier-free formulae. Of course, a $\mathcal{T}$-satisfiability procedure could be applied after reduction to disjunctive normal form, but this approach is not practical. Another method would be to investigate how to integrate rewrite-based $\mathcal{T}$-satisfiability procedures with a SAT solver, as done for example in [7,10,2,12,5] for $\mathcal{T}$-satisfiability procedures based on congruence closure. Here, we choose instead to study the problem of whether rewrite-based theorem-proving strategies can be themselves $\mathcal{T}$-decision procedures.

The main contributions of the paper are the following:

- We introduce the notion of *subterm-inactivity* and prove that if a theory $\mathcal{T}$ is subterm-inactive, a fair $\mathcal{SP}$-based strategy is a decision procedure for the $\mathcal{T}$-decision problem.
- The conditions to be met for a theory to be subterm-inactive are easy to test, and all but one can be *tested automatically*. This is a significant advantage, compared to the termination proofs of [3,1] where one has to analyze the inferences that can be carried out starting from the presentation $\mathcal{T}$ and a set of ground unit clauses. Furthermore, the only requirement we impose on the complete simplification ordering $\succ$ assumed by $\mathcal{SP}$ is that $t \succ c$ for every compound term t and constant c.
- We prove that every subterm-inactive theory is also variable-inactive.
- We show that several of the theories considered in [3,1], as well as two extensions of the theory of arrays, are subterm-inactive.

Due to a lack of space, most of the proofs could not be included in this paper. They can all be found in [8].

2 Preliminaries

2.1 Terms, literals and clauses

Given a signature Σ, Σ^n denotes the set of functions in Σ with arity n. Thus, Σ^0 denotes the set of constants in Σ. We consider the standard definitions of Σ-terms, Σ-literals and Σ-clauses. As usual, clauses are assumed to be variable-disjoint. In the following, $\simeq$ is unordered equality, $\bowtie$ is either $\simeq$ or $\not\simeq$. The letters l, r, u, v and t will denote terms, w, x, y, z variables, and all other lower-case letters will denote constants or function symbols. Given a term t, $\mathsf{top}(t)$ is the symbol appearing as

t's top symbol, and $\mathrm{Var}(t)$ denotes the set of variables appearing in t. We will also consider the natural extension of Var to literals and clauses: for example, if C is a clause, then $\mathrm{Var}(C)$ is the set of variables appearing in C.

Given the presentation $\mathcal{T}$ of a theory, a function symbol is *interpreted* if it appears in an axiom of $\mathcal{T}$, and it is *uninterpreted* otherwise. The *$\mathcal{T}$-satisfiability problem* is the problem of deciding whether a set of ground unit clauses is satisfiable in $\mathcal{T}$. The more general *$\mathcal{T}$-decision problem* is the problem of deciding the satisfiability of any ground formula in $\mathcal{T}$. Without loss of generality, we can assume that the considered ground formulae are conjunctions of clauses.

Definition 2.1 Given a signature Σ, a selection function is a function from Σ to $\mathbb{N}$ such that for all $f \in \Sigma$, $\Gamma(f) \in \{1, \dots, \mathrm{arity(f)}\}$. Ω_Σ denotes the set of selection functions for Σ.

A selection function selects an argument in a term. For example, for a function symbol f and selection function Γ, if $\Gamma(f) = i$, then Γ selects the subterm t_i from the term $f(t_1, \dots, t_n)$. The name "selection function" is also used for functions that select a literal in a clause: the two definitions are compatible, since such functions can be seen as selecting an argument of a disjunction operator.

We define the notion of symbol-freeness, which prevents some function or constant symbols from appearing in a clause.

Definition 2.2 (Symbol-freeness) Given a term t, $\Phi(t)$ denotes the set of function and constant symbols appearing in t. Also, let $\Phi(l \bowtie r) = \Phi(l) \cup \Phi(r)$ and $\Phi(C) = \bigcup_{L \in C} \Phi(L)$. Given a set of function and constant symbols Σ', a term t is *Σ'-symbol-free* if $\Phi(t) \cap \Sigma'$ consists only of constants, and t is *strictly Σ'-symbol-free* if this intersection is empty. A literal (resp. clause) is *Σ'-symbol-free* if every term appearing in it is. A clause is *subsymbol-free from Σ'* if every literal in C that contains a function symbol is strictly Σ'-symbol-free.

2.2 Flattening

If a term t is a constant or a variable, then the *depth* of t is $\mathrm{depth}(t) = 0$, otherwise $\mathrm{depth}(f(t_1, \dots, t_n)) = 1 + \max\{\mathrm{depth}(t_i) \mid i = 1, \dots, n\}$. The depth of the literal $l \bowtie r$ is $\max(\mathrm{depth}(l), \mathrm{depth}(r))$. A positive literal is *flat* if $\mathrm{depth}(l) + \mathrm{depth}(r) \leq 1$, and a negative literal is *flat* if its depth is 0.

Definition 2.3 A literal is *strictly flat* if its depth is 0. For a clause C, let $\mathrm{Maxd}(C) = \max\{\mathrm{depth}(t) \mid t \text{ is a term appearing in } C\}$. The clause C is *flat*, respectively, *strictly flat*, if all its literals are.

We will make an intensive use of *flattening*. The operation of flattening consists in transforming a finite set of ground clauses S over a signature Σ, into a finite set of ground clauses S' over a signature Σ', in such a way that:

- Σ' is obtained by adding a finite number of constants to Σ,
- every non-unit clause in S' is strictly flat,

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	$\dfrac{C \vee u' \not\simeq u}{C\sigma}$	(v)
Equational Factoring	$\dfrac{C \vee u \simeq t \vee u' \simeq t'}{(C \vee t \not\simeq t' \vee u \simeq t')\sigma}$	$(i), (vi)$

where the notation $l[u']$ means that u' appears as a subterm in l, 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): $u\sigma \npreceq t\sigma$;

(ii): $\forall L \in D : (u \simeq t)\sigma \npreceq L\sigma$;

(iii): $l[u']\sigma \npreceq r\sigma$;

(iv): $\forall L \in C : (l[u'] \bowtie r)\sigma \npreceq L\sigma$;

(v): $\forall L \in C : (u' \simeq u)\sigma \nprec L\sigma$;

(vi): $\forall L \in \{u' \simeq t'\} \cup C : (u \simeq t)\sigma \nprec L\sigma$.

Fig. 1. Expansion inference rules of $\mathcal{SP}$: in expansion rules, what is below the inference line is added to the clause set that contains what is above the inference line.

- every unit clause in S' is flat,
- for all sets $\mathcal{T}$, $\mathcal{T} \cup S$ and $\mathcal{T} \cup S'$ are equisatisfiable.

This flattening operation is fairly straightforward, and it is more general than the one in [3], where only unit clauses are considered. As an example, consider the set $S = \{f(f(a)) \simeq b \vee f(c) \not\simeq d\}$: by introducing fresh constants c_1, c_2 and c_3, we obtain the equisatisfiable set

$$S' = \{f(a) \simeq c_1,\ f(c_1) \simeq c_2,\ f(c) \simeq c_3,\ c_2 \simeq b \vee c_3 \not\simeq d\}.$$

2.3 Rewrite-based inference systems

A *simplification ordering* $\succ$ is an ordering that is *stable*, *monotonic* and contains the *subterm ordering*: if $s \succ t$, then $c[s]\sigma \succ c[t]\sigma$ for any context c and substitution σ, and if t is a subterm of s then $s \succ t$. A *complete simplification ordering*, or CSO, is a simplification ordering that is total on ground terms. We write $t \prec s$ if $s \succ t$. More details on orderings can be found, e.g., in [11].

In the sequel, except stated otherwise, we will assume that for the considered CSO, if t is a compound term and c a constant, then $t \succ c$. This condition is part

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

where $D \gtrdot C$ if $D \underline{\gtrdot} C$ and $C \not\underline{\gtrdot} D$; and $D \underline{\gtrdot} C$ if $C\sigma \subseteq D$ (as multisets) for some substitution σ. In practice, theorem provers apply also subsumption of variants: if $D \underline{\gtrdot} C$ and $C \underline{\gtrdot} D$, the oldest clause is retained.

Fig. 2. Contraction inference rules of $\mathcal{SP}$: in contraction rules, what is above the double inference line is removed from the clause set and what is below the double inference line is added to the clause set.

of the $\mathcal{T}$*-goodness requirement* for all the theories considered in [1]. We refer to this requirement simply as the *goodness requirement.*

The *superposition calculus*, or $\mathcal{SP}$ (see [13]), is a *rewrite-based inference system* which is refutationally complete for first-order logic with equality. It consists of *expansion rules* (see Figure 1) and *contraction rules* (see Figure 2), and is based on a CSO on terms which is extended to literals and clauses in the standard way. Given a CSO $\succ$, we write $\mathcal{SP}_\succ$ for $\mathcal{SP}$ equipped with $\succ$. A clause C is *redundant* with respect to $\mathcal{SP}$ in a set of clauses S, if S can be derived from $S \cup \{C\}$ by application of a contraction rule in $\mathcal{SP}$. Since $\mathcal{SP}$ is the only inference system in this article, we write redundant for redundant with respect to $\mathcal{SP}$. An inference is *redundant* in S, if either its conclusion or one of its premises is redundant in S.

An $\mathcal{SP}$*-strategy* is given by $\mathcal{SP}$ together with a search plan that controls the application of the inference rules. An $\mathcal{SP}_\succ$*-derivation* is a sequence

$$S_0 \vdash_{\mathcal{SP}_\succ} S_1 \vdash_{\mathcal{SP}_\succ} \dots S_i \vdash_{\mathcal{SP}_\succ} \dots,$$

where each S_i is a set of clauses, obtained by applying an expansion or a contraction rule to clauses in S_{i-1}. The *limit* of such a derivation is the set of *persistent clauses*:

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

A derivation $S_0 \vdash_{\mathcal{SP}_\succ} \dots S_n \vdash_{\mathcal{SP}_\succ} \dots$ is *fair* with respect to $\mathcal{SP}_\succ$ if all expansion inferences in $\mathcal{SP}_\succ$ with premises in S_∞ are redundant in some S_j for $j \geq 0$. A search plan is *fair* if all the derivations it controls are fair, and an $\mathcal{SP}_\succ$-strategy is fair if its search plan is. A set of clauses S is *saturated* if every clause generated from clauses in S by an $\mathcal{SP}$-inference is redundant.

A clause C is *variable-inactive for* $\succ$ (see [1]) if no maximal literal in C is an equation $t \simeq x$, where $x \notin \mathrm{Var}(t)$. A set of clauses is *variable-inactive for* $\succ$ if all its clauses are variable-inactive for $\succ$. A presentation $\mathcal{T}$ is *variable-inactive for* $\succ$ if the limit S_∞ of a fair $\mathcal{SP}_\succ$-derivation from $S_0 = \mathcal{T} \cup S$ is variable-inactive. When no confusion is possible, we will say that a clause (resp. a set of clauses or a theory presentation) is variable-inactive, without any mention of $\succ$.

We conclude the preliminaries with the notion of *depth-preservation.* Intuitively, this notion prevents the clauses generated by the expansion inference rules from becoming arbitrarily large.

Definition 2.4 Let C, C' and D be clauses, and suppose D is generated from C by a unary inference: this inference is *depth-preserving* if $\mathrm{Maxd}(D) \leq \mathrm{Maxd}(C)$. Suppose D is generated from C and C' by a binary inference: this inference is *depth-preserving* if $\mathrm{Maxd}(D) \leq \max\{\mathrm{Maxd}(C), \mathrm{Maxd}(C')\}$.

3 Subterm-inactivity

The proofs that the superposition calculus terminates on satisfiability problems for different theories are based on an enumeration of the kinds of clauses that can be generated by the inferences (see [3,1,8]). However, the number of clauses in S_∞ can be exponentially large (it can contain for example up to $O(2^{n^2})$ clauses in the theory of arrays, see [1] for details), and in general, such proofs consist of showing that all generated clauses belong to one of several categories: if each of these categories contains a finite number of clauses, so will S_∞. These proofs can be quite long, and at each new inference, a new category to deal with may arise. In this section, we introduce a set of conditions guaranteeing that a fair strategy based on $\mathcal{SP}_\succ$ is a decision procedure for the considered theory. These conditions are easy to verify and more importantly, almost all can be verified automatically.

Informally, we will consider $\mathcal{T}$-decision problems whose clauses can be divided into three disjoint sets:

- a set T_g of ground clauses,
- a set T_1 of non-ground clauses representing properties that can be deduced by considering one interpreted function symbol,
- a set T_2 of non-ground clauses representing the way two interpreted function symbols may interact in $\mathcal{T}$.

This pattern applies to $\mathcal{T}$-decision problems in several theories of interest such as, for example, the theory of arrays.

Example 3.1 The theory of arrays $\mathcal{A}$, based on the signature $\Sigma_\mathcal{A} = \{\mathrm{select}, \mathrm{store}\}$, where select has arity 2 and store has arity 3, is axiomatized as follows:

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

$$\forall x, z, w, v.\ (z \simeq w \vee \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w)). \tag{2}$$

The theory of arrays with extensionality $\mathcal{A}^\mathrm{e}$ is defined by axioms (1) and (2), along with the following extensionality axiom:

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

A rewrite-based $\mathcal{T}$-decision procedure for the theory $\mathcal{A}^{\mathrm{e}}$ takes as input a set T_{g} of ground clauses, together with $\{(1),(2),(3)\}$. This set can itself be decomposed into two disjoint subsets: $T_2 = \{(1),(2)\}$ which describes the way select and store interact, and $T_1 = \{(3)\}$ which describes the equality property that can be deduced from the select function.

Of course, these sets can interact with each other, and it is necessary to control these interactions as much as possible in order to guarantee termination. Before giving any formal definition, we informally enumerate the requirements that should be satisfied by these sets and which conditions are imposed to satisfy them.

General properties

(i) Each clause in T_2 expresses a single property verified when combining at most two interpreted function symbols, and each clause in T_1 expresses a property that can be deduced by considering a single interpreted function symbol (**closure**);

(ii) Any $\mathcal{SP}_{\succ}$-inference generating a persistent clause is depth-preserving (**flatness**).

Binary inferences

(i) There is no binary $\mathcal{SP}_{\succ}$-inference between a clause in T_2 and one in T_1 (**interaction-freeness**);

(ii) A binary $\mathcal{SP}_{\succ}$-inference between a clause in $T_1 \cup T_2$ and a clause in T_{g} generates a clause in T_1 or in T_{g} (**closure + negative disconnection**);

(iii) A binary $\mathcal{SP}_{\succ}$-inference between two clauses in T_2 generates a clause which is deleted eventually (**saturation**);

(iv) A binary $\mathcal{SP}_{\succ}$-inference between two clauses in T_1 generates a clause in T_1 or in T_{g} (**closure**).

Unary inferences

(i) A unary inference within T_2 generates a clause that is deleted eventually (**saturation**);

(ii) A unary inference within T_1 generates a clause that is in T_1, in T_{g}, or is deleted eventually (**variable-inactivity preservation**).

In the following subsections we formally define these notions.

3.1 Restrictions on T_2

The conditions we impose on T_2 are termed collectively *saturation closure*. Informally, these conditions ensure that T_2 is saturated, and that every clause generated by a binary inference involving a clause in T_2 is in T_1 or in T_{g}.

Definition 3.2 (Ordered flatness) A clause C is *ordered flat* if it only contains strictly flat literals except for one, say $l \bowtie r$. Furthermore, it must be $r \prec l$, and r must contain only function symbols appearing in l.

Example 3.3 Consider the following clauses:

$$C = f(a) \simeq b \vee c \not\simeq d,$$
$$C' = f(g(a)) \simeq g(a) \vee c \not\simeq d.$$

These two clauses are ordered flat.

Definition 3.4 (Internal closure) Let S be a set of clauses and Γ be a selection function. S is Γ-*internally closed* if for every clause $C \in S$ and every non-strictly flat literal $L = l \bowtie r$ in C:

- If L is negative, then:
  icn.1: for every subterm u of l of depth 1, $\mathrm{Var}(C) \subseteq \mathrm{Var}(u)$,
  icn.2: every positive literal in a clause of S is $\Phi(L)$-symbol-free.
- If L is positive, then we must have $r \prec l$ and:
  icp.1: $\mathrm{depth}(l) = 2$, and l contains a unique subterm u of depth 1,
  icp.2: $\mathrm{Var}(C) = \mathrm{Var}(l)$,
  icp.3: if $\mathrm{top}(r) \neq \mathrm{top}(l)$, then $\mathrm{depth}(r) = 0$ and $\mathrm{Var}(C) \subseteq \mathrm{Var}(u)$,
  icp.4: if $\mathrm{top}(r) = \mathrm{top}(l)$, then $\mathrm{depth}(r) = 1$, $r|q_f = l|q_f$, $l|q_f$ appears nowhere else in l or r, and $\mathrm{Var}(l) \setminus \mathrm{Var}(u) = \{l|q_f\}$.

By also imposing that T_2 is saturated and that every literal appearing in T_2 that contains a constant is strictly flat (formally, that every clause is subsymbol-free from Σ^0), we obtain the following definition of saturation closure:

Definition 3.5 (Saturation-closure) Let $\Gamma \in \Omega_\Sigma$, a set of clauses S is Γ-*saturation-closed* if

- it is saturated,
- every clause in S is subsymbol-free from Σ^0,
- every clause in S is ordered flat,
- S is Γ-internally closed.

3.2 Restrictions on T_1

The restrictions imposed to T_1 prevent its clauses from interacting with T_2, and control the clauses generated by inferences involving these clauses.

Definition 3.6 (Weak flatness) A clause C is *weakly flat*, if C only contains literals with terms of depth at most 1, and at least one non-ground literal $l \bowtie r$ which is not strictly flat. Furthermore, if C contains a literal $x \bowtie t$, then t is of depth 0.

Example 3.7 The clause $C = f(a) \simeq b \vee f(x) \not\simeq d$ is weakly flat.

Definition 3.8 (Variable-inactivity preservation) Given a function $\Gamma \in \Omega_\Sigma$, a clause C is Γ-*variable-inactive preserving* if and only if:

vip-1: For every variable $x \in \mathrm{Var}(C)$ and for every literal L in C which is not strictly flat, x is a variable of a term of depth 1 in L.

vip-2: If C contains a negative literal $l \not\simeq r$ with $\mathrm{top}(l) = \mathrm{top}(r) = f$, then C also contains a literal $x \simeq t$ such that either t is a variable and $\mathrm{Var}(C) \subseteq \{x, t\}$, or $\mathrm{Var}(C) = \{x\}$. Furthermore, let $q_f = \Gamma(f)$, then:

a. if t is a variable, then $\{x, t\} = \{l|q_f, r|q_f\}$,

b. if t is a constant, then there is a constant c (not necessarily equal to t) such that $\{x, c\} = \{l|q_f, r|q_f\}$.

A set of clauses S is Γ-variable-inactive preserving if every clause in S is.

Example 3.9 Let $\Sigma_I = \{\mathrm{Inj}\}$, where Inj is a predicate of arity 1, and consider the theory $\mathcal{A}_I$, based on the signature $\Sigma_{\mathcal{A}} \cup \Sigma_I$, which is axiomatized by axioms (1) and (2) of Example 3.1, and the following axiom denoted by (**inj**):

$$\mathrm{Inj}(x) \Leftrightarrow \forall z, w.\ (z \not\simeq w \supset \mathrm{select}(x, z) \not\simeq \mathrm{select}(x, w)).$$

Intuitively, the predicate Inj is true for array a if and only if all the elements in a are pairwise distinct (a is injective). Consider the following clausal form, logically equivalent to $\mathrm{Inj}(a)$:

$$C = z \simeq w \vee \mathrm{select}(a, z) \not\simeq \mathrm{select}(a, w).$$

This clause contains a single literal that is not strictly flat, $L = \mathrm{select}(a, z) \not\simeq \mathrm{select}(a, w)$. We have $\mathrm{Var}(C) = \{z, w\}$, and these two variables appear in terms of depth 1 in L. Let Γ be any function in Ω_Σ such that $\Gamma(\mathrm{select}) = 2$. Since $z \simeq w$ is also a literal in C and condition *(vip.2.a)* holds on C, this clause is Γ-variable-inactive preserving.

Definition 3.10 (External closure) Let C be a clause, S' be a set of clauses, $\Gamma \in \Omega_\Sigma$, and for every $f \in \Sigma$, let $q_f = \Gamma(f)$. C is Γ-*externally closed from* S' if for every positive literal $l \simeq r$ in C such that $\mathrm{top}(l) = f$,

ec.1: $\mathrm{top}(l) = \mathrm{top}(r)$,

ec.2: all the other literals in C are strictly flat,

ec.3: $l|q_f = r|q_f$ is the only variable in C and this variable appears nowhere else in l or r.

ec.4: every negative literal in a clause of S' is $\{f\}$-symbol-free.

A set of clauses S is Γ-*externally closed from* S' if every clause in S is.

Example 3.11 Let $\Sigma_S = \{\mathrm{Swap}\}$, where Swap is a predicate that has arity 4, and consider the theory $\mathcal{A}_S$, based on signature $\Sigma_{\mathcal{A}} \cup \Sigma_S$ and axiomatized by (1), (2) (the axioms of $\mathcal{A}$, see Example 3.1) and the following axiom denoted by (**swp**):

$$\begin{aligned}\mathrm{Swap}(x, y, z_1, z_2) \Leftrightarrow\ & \mathrm{select}(x, z_1) \simeq \mathrm{select}(y, z_2) \wedge \\ & \mathrm{select}(x, z_2) \simeq \mathrm{select}(y, z_1) \wedge \\ & \forall w.\ (w \not\simeq z_1 \wedge w \not\simeq z_2 \supset \mathrm{select}(x, w) \simeq \mathrm{select}(y, w)).\end{aligned}$$

Given constants b, b', i and i', the atom $\mathrm{Swap}(b, b', i, i')$ is true if and only if b' is identical to b, except that the elements at indices i and i' are swapped. Consider the clause

$$D = w \simeq i \vee w \simeq i' \vee \mathrm{select}(b, w) \simeq \mathrm{select}(b', w).$$

Let Γ be any function in Ω_Σ such that $\Gamma(\text{select}) = 2$, and let $S' = \{(1), (2)\}$. It is simple to check that D satisfies conditions *(ec.1)* to *(ec.4)*, and is therefore Γ-externally closed from S'.

Definition 3.12 (Immunity) Given two sets of clauses S and S' and a function $\Gamma \in \Omega_\Sigma$, S is Γ-*immune from* S' if and only if

- every clause in S is weakly flat,
- every clause in S is Γ-variable-inactive preserving,
- S is Γ-externally closed from S'.

Definition 3.13 (Interaction-freeness) Given two sets of clauses S and S' and a function $\Gamma \in \Omega_\Sigma$, S is Γ-*interaction-free from* S' if the following condition is satisfied: let f be a function symbol, let $p_f = \Gamma(f)$, and suppose that

- either f occurs at the same time in a positive literal of a clause in S and in a literal of a clause in S',
- or f occurs at the same time in a positive literal of a clause in S' and in a literal of a clause in S.

Then for all clauses $C \in S \cup S'$ containing a literal $L = l \bowtie r$, such that f appears in l or in r, the following conditions must hold:

if.1: f only appears as the top symbol of l or r,

if.2: if $C \in S$, then $l|p_f$ is a constant, and if r is neither a constant nor a variable, then $r|p_f$ is a constant,

if.3: if $C \in S'$ and L is negative, then $l|p_f$ is a term of depth 1,

if.4: if $C \in S'$ and L is positive, then $u = l|p_f$ is a term of depth 1, and if r is neither a constant nor a variable, then $r|p_f$ is either a constant or a variable in $\text{Var}(u)$.

Example 3.14 Consider $S = \{D\}$, where D is the clause of Example 3.11 and $S' = \{(1), (2)\}$. The only function symbol these sets have in common is select. Let Γ be any function in Ω_Σ such that $\Gamma(\text{select}) = 1$. Then it is clear that D satisfies conditions *(if.1)* and *(if.2)*, and that the clauses in S' satisfy condition *(if.4)*. Thus, S is Γ-interaction-free from S'. Similarly, consider the clause C from Example 3.9, then $\{C\}$ is also Γ-interaction-free from S'.

3.3 Restrictions on T_g

We finally define the notion of *flat disconnection* for T_g.

Definition 3.15 (Positive flatness) A clause C is *positively flat* if each time C contains a positive literal which is not strictly flat, this literal is flat and all the other literals in C are strictly flat.

Example 3.16 Consider the following clauses:

$$C = f(a) \simeq b \vee c \not\simeq d,$$
$$C' = f(f(a)) \not\simeq b \vee f(c) \not\simeq d,$$

The clauses C and C' are both positively flat.

Definition 3.17 (Negative disconnection) Let C be a clause and S' be a set of clauses. C is *negatively disconnected from* S' if whenever C contains a negative literal $l \not\simeq r$ such that $\mathrm{depth}(l) \geq 2$, every positive literal of a clause in S' is $\Phi(C)$-symbol-free. A set of clauses S is *negatively disconnected from* S' if every clause in S is negatively disconnected from S'.

Definition 3.18 (Flat-disconnection) Given a clause C and a set of clauses S', C is *flat-disconnected from* S' if and only if C is

- positively flat,
- negatively disconnected from S'.

A set of clauses S is *flat-disconnected from* S' if every clause in S is.

Example 3.19 Any set of flattened ground clauses is flat-disconnected from any other set of clauses S'. Indeed, such a set is trivially positively flat; since all its negative literals are strictly flat, there is no literal $l \not\simeq r$ with $\mathrm{depth}(l) \geq 2$, and the set is also negatively disconnected from S'.

3.4 Subterm-inactivity

We introduce the fundamental notion of *subterm-inactivity*, which guarantees the termination of $\mathcal{SP}$ on the decision problem in the considered theory.

Definition 3.20 (Subterm-inactivity) Let T_{g}, T_1 and T_2 be three disjoint sets of clauses. The tuple $\langle T_{\mathrm{g}}, T_1, T_2\rangle$ is *subterm-inactive* if there exist two functions Γ and Γ' in Ω_Σ such that:

- T_{g} only contains ground clauses and is flat-disconnected from $T_1 \cup T_2$,
- T_1 is Γ-immune and Γ'-interaction-free from T_2,
- T_2 is Γ-saturation-closed.

A presentation of a theory $\mathcal{T}$ is *subterm-inactive* if there exists a partition $T_{\mathrm{g}} \uplus T_1 \uplus T_2$ of $\mathcal{T}$ such that $\langle T_{\mathrm{g}}, T_1, T_2\rangle$ is subterm-inactive.

Since we can flatten any set of ground clauses, we can safely add it to a subterm-inactive presentation:

Proposition 3.21 *If $\mathcal{T}$ is a subterm-inactive presentation, then for every set of ground clauses S, there exists a set of ground clauses S' such that $S' \cup \mathcal{T}$ is equisatisfiable to $S \cup \mathcal{T}$, and $S' \cup \mathcal{T}$ is subterm-inactive.*

We will give several examples of subterm-inactive theories in the following section. Before that, we state the main results we obtain under the subterm-inactivity hypothesis:

Theorem 3.22 *Given a set of clauses $\mathcal{T} = T_{\mathrm{g}} \uplus T_1 \uplus T_2$ such that $\langle T_{\mathrm{g}}, T_1, T_2\rangle$ is a subterm-inactive tuple:*

(i) *If D is a persistent clause generated by an $\mathcal{SP}$-inference in $\mathcal{T}$, then the inference is depth-preserving and:*
   - *either D is ground and $\langle T_{\mathrm{g}} \cup \{D\}, T_1, T_2\rangle$ is subterm-inactive,*
   - *or D is not ground and $\langle T_{\mathrm{g}}, T_1 \cup \{D\}, T_2\rangle$ is subterm-inactive.*

(ii) *A fair $\mathcal{SP}_{\succ}$-strategy is a decision procedure for $\mathcal{T}$.*

(iii) *$\mathcal{T}$ is variable-inactive.*

The proof of Theorem 3.22, and especially of *(i)* requires considering all possible inferences that can be applied to the sets T_{g}, T_1 and T_2. The complete treatment of the different cases and the other proofs can all be found in [8].

4 Variations on the theory of arrays

In what follows, we consider the theory of arrays (see Example 3.1) and two of its extensions. It was shown in [3] that a satisfiability problem in $\mathcal{A}^{\mathrm{e}}$ can be reduced to an equisatisfiable satisfiability problem in $\mathcal{A}$, and that the superposition calculus provides a satisfiability procedure for $\mathcal{A}$: a proof that the limit S_∞ is finite can be found in [1]. This kind of analysis requires long proofs: for $\mathcal{A}$, the clauses in S_∞ can belong to any one of 14 classes of clauses. We have the following result:

Theorem 4.1 *The presentation of $\mathcal{A}$ is subterm-inactive.*

Proof. We prove that the tuple $\langle \emptyset, \emptyset, \{(1), (2)\}\rangle$ is subterm-inactive.

- The only inference that can be applied to $\{(1), (2)\}$ is a superposition between (1) and (2). This generates the clause $z \simeq z \vee \mathrm{select}(x, z) \simeq v$, which is deleted. Thus, this set is saturated.
- It is trivial to check that the clauses in $\{(1), (2)\}$ are ordered flat. Since they do not contain any constants, they are also subsymbol-free from Σ^0.
- The maximal literals in (1) and (2) are both positive and one can check that $\{(1), (2)\}$ is Γ-internally closed, for any selection function Γ such that $\Gamma(\mathrm{select}) = 2$.

□

Thus, by Theorem 3.22 *(ii)*, we deduce that:

Corollary 4.2 *Any fair $\mathcal{SP}_{\succ}$-strategy is a decision procedure for the theory of arrays with or without extensionality.*

4.1 An injectivity predicate

Next, we consider the theory $\mathcal{A}_I$ of arrays augmented with an injectivity predicate, as defined in Example 3.9:

$$\mathrm{Inj}(x) \Leftrightarrow \forall z, w.\ (z \not\simeq w \supset \mathrm{select}(x, z) \not\simeq \mathrm{select}(x, w)).$$

We assume that each occurrence of the injectivity predicate has a constant as an argument. There is no loss of generality under this assumption. For example, the clause $\mathrm{Inj}(f(a)) \vee B$ can be safely replaced by the clause $\mathrm{Inj}(b) \vee B$ and the flat literal $f(a) \simeq b$, where b is a fresh constant. Still without loss of generality, we may suppose that if the injectivity predicate appears in a non-unit clause, then this clause is of the form $\mathrm{Inj}(a) \vee \neg p$ or $\neg\mathrm{Inj}(a) \vee \neg p$, where p is a propositional variable. Indeed, such a formula can be obtained from S by repeatedly replacing clauses of the form $\mathrm{Inj}(a) \vee D$ (resp. $\neg\mathrm{Inj}(a) \vee D$), where D is not a propositional variable, by the clauses $\mathrm{Inj}(a) \vee \neg p_a$ and $p_a \vee D$ (resp. $\neg\mathrm{Inj}(a) \vee \neg p_a$ and $p_a \vee D$), where p_a is a fresh propositional variable. The formula thus obtained is equisatisfiable to S (see [14] for details).

We remove all occurrences of the predicate Inj in the following way. For every constant a, we consider the clause C_a and its negated form C'_a, respectively defined by:

$$C_a = z \simeq w \vee \mathrm{select}(a, z) \not\simeq \mathrm{select}(a, w),$$
$$C'_a = (sk_1 \not\simeq sk_2 \wedge \mathrm{select}(a, sk_1) \simeq \mathrm{select}(a, sk_2)),$$

where sk_1 and sk_2 are fresh Skolem constants. Note that, by definition, $\mathrm{Inj}(a)$ is logically equivalent to $\forall z, w.\ (z \not\simeq w \supset \mathrm{select}(a, z) \not\simeq \mathrm{select}(a, w))$, and C_a is the clausal form of the latter formula. Thus, C_a and $\mathrm{Inj}(a)$ are logically equivalent; similarly, C'_a and $\neg\mathrm{Inj}(a)$ are also logically equivalent.

We can therefore safely replace every clause of the form $\mathrm{Inj}(a) \vee \neg p$ by $C_a \vee \neg p$, and every clause of the form $\neg\mathrm{Inj}(a) \vee \neg p$ by the clausal form of $C'_a \vee \neg p$.

Example 4.3 Let $S = \{\neg\mathrm{Inj}(a) \vee \mathrm{Inj}(b)\}$. By introducing the fresh propositional variable p_a we obtain the set $S' = \{\neg\mathrm{Inj}(a) \vee \neg p_a,\ \mathrm{Inj}(b) \vee p_a\}$, and after the aforementioned transformation we get

$$\begin{aligned} S'' = \{\ & sk_1 \not\simeq sk_2 \vee \neg p_a, \\ & \mathrm{select}(a, sk_1) \simeq \mathrm{select}(a, sk_2) \vee \neg p_a, \\ & z \simeq w \vee \mathrm{select}(b, z) \not\simeq \mathrm{select}(b, w) \vee p_a\} \end{aligned}$$

where z and w are implicitly universally quantified variables.

Given a set of clauses S, the reduced set of clauses thus obtained is equisatisfiable to S, and we have the following:

Lemma 4.4 *Let $\{a_1, \ldots, a_n\}$ be a set of constants, for all $i \in \{1, \ldots, n\}$ let p_i be a propositional variable (or the negation of a propositional variable) and define*

$$C_i = \forall z, w.\ z \simeq w \vee \mathrm{select}(a_i, z) \not\simeq \mathrm{select}(a_i, w).$$

The theory $\mathcal{A} \cup \{C_i \vee p_i \mid i = 1, \ldots, n\}$ is subterm-inactive.

Proof. We show that the tuple $\langle \emptyset, \{C_1 \vee p_1, \ldots, C_n \vee p_n\}, \{(1), (2)\}\rangle$ is subterm-inactive. In Theorem 4.1, we showed that $\{(1), (2)\}$ is Γ-saturation-closed with $\Gamma(\mathrm{select}) = 2$. Consider any function $\Gamma' \in \Omega_\Sigma$ such that $\Gamma'(\mathrm{select}) = 1$, and a clause $C_i \vee p_i$. This clause is Γ-immune from $\{(1),(2)\}$: we have shown that C_i is Γ-variable-inactive preserving in Example 3.9, and it is simple to verify that $C_i \vee p_i$ is also Γ-externally closed from $\{(1),(2)\}$; the other conditions are trivial to verify. It is also Γ'-interaction-free from $\{(1),(2)\}$, hence the result.

Since these conditions are satisfied for every clause of the form $C_i \vee p_i$, it is clear that $\langle \emptyset, \{C_1 \vee p_1, \ldots, C_n \vee p_n\}, \{(1),(2)\}\rangle$ is subterm-inactive and the proof is complete. □

Thus, by Theorem 3.22 *(ii)*, an $\mathcal{SP}_{\succ}$-strategy together with a fair search plan can be used to test the satisfiability of $\mathcal{A} \cup \{C_i \vee p_i \mid i = 1, \ldots, n\}$. We therefore have the following result:

Corollary 4.5 *A fair $\mathcal{SP}_{\succ}$-strategy is a decision procedure for $\mathcal{A}_I$.*

4.2 A Swap predicate

We now turn to the theory $\mathcal{A}_S$ of arrays augmented with a swap predicate, as defined in Example 3.11:

$$\begin{aligned}\mathrm{Swap}(x,y,z_1,z_2) \Leftrightarrow\ & \mathrm{select}(x,z_1) \simeq \mathrm{select}(y,z_2) \wedge \\ & \mathrm{select}(x,z_2) \simeq \mathrm{select}(y,z_1) \wedge \\ & \forall w.\ (w \not\simeq z_1 \wedge w \not\simeq z_2 \supset \mathrm{select}(x,w) \simeq \mathrm{select}(y,w)).\end{aligned}$$

Consider a ground $\mathcal{A}_S$-formula S. Similar to the case of the injectivity predicate, up to adding flat equalities to S, we assume that each occurrence of the swap predicate only has constants as arguments, and that the clauses in which this predicate appears are of the form $\mathrm{Swap}(b,b',i,i') \vee \neg p$ or $\neg\mathrm{Swap}(b,b',i,i') \vee \neg p$.

We remove all occurrences of the predicate Swap in the following way: for every tuple of constants $G = \langle b, b', i, i'\rangle$, where b and b' are of sort array and i and i' are of sort index, we consider the formula D_G and its negated form D'_G respectively defined by:

$$\begin{aligned}D_G &= \mathrm{select}(b,i) \simeq \mathrm{select}(b',i') \wedge \mathrm{select}(b,i') \simeq \mathrm{select}(b',i) \wedge \\ &\qquad \forall w.\ (w \not\simeq i \wedge w \not\simeq i' \supset \mathrm{select}(b,w) \simeq \mathrm{select}(b',w)) \\ D'_G &= \mathrm{select}(b,i) \not\simeq \mathrm{select}(b',i') \vee \mathrm{select}(b,i') \not\simeq \mathrm{select}(b',i) \vee \\ &\qquad (sk \not\simeq i \wedge sk \not\simeq i' \wedge \mathrm{select}(b,sk) \not\simeq \mathrm{select}(b',sk)),\end{aligned}$$

where sk is a fresh Skolem constant. By definition, $\mathrm{Swap}(b,b',i,i')$ and D_G are logically equivalent, and one can check that $\neg\mathrm{Swap}(b,b',i,i')$ and D'_G are also logically equivalent.

Given a tuple $G = \langle b, b', i, i'\rangle$, we can therefore safely replace every formula of the form $\mathrm{Swap}(b,b',i,i') \vee \neg p$ by the clausal form of $D_G \vee \neg p$, and every formula of the form $\neg\mathrm{Swap}(b,b',i,i') \vee \neg p$ by the clausal form of $D'_G \vee \neg p$. The set we obtain is equivalent to S. In the following lemma, the clause Dk comes from the clausal form of D_G; the rest of the clausal form of D_G is ground, as is the clausal form of D'_G: they are not needed to prove subterm-inactivity.

Lemma 4.6 *Let $\{b_k, b'_k, i_k, i'_k \mid k = 1,\dots,m\}$ be a set of constants and for every $k \in \{1,\dots,m\}$, let p_k be a propositional variable (or the negation of a propositional variable) and consider the clause*

$$D_k = w \simeq i_k \ \vee\ w \simeq i'_k \ \vee\ \mathrm{select}(b_k, w) \simeq \mathrm{select}(b'_k, w).$$

The theory $\mathcal{A} \cup \{D_k \vee p_k \mid k = 1,\dots,m\}$ is subterm-inactive.

Proof. We prove that $\langle \emptyset, \{D_1 \vee p_1, \dots, D_m \vee p_m\}, \{(1),(2)\}\rangle$ is subterm-inactive. It is simple to check that $\{(1),(2)\}$ is Γ-saturation-closed with $\Gamma(\mathrm{select}) = 2$. Consider a clause D_k, it can be seen by applying Definition 3.12 that $D_k \vee p_k$ is Γ-immune from $\{(1),(2)\}$ (see also Example 3.11). As shown in Example 3.14, D_k is Γ'-interaction-free from $\{(1),(2)\}$ for any Γ' such that $\Gamma'(\mathrm{select}) = 1$, and it is simple to show that $D_k \vee p_k$ is also Γ'-interaction-free from $\{(1),(2)\}$. Hence, for every k, $\langle \emptyset, \{D_k \vee p_k\}, \{(1),(2)\}\rangle$ is subterm-inactive. Since this is true for every k, we have the result. □

As for Corollary 4.5, we deduce:

Theorem 4.7 *A fair $\mathcal{SP}_{\succ}$-strategy is a decision procedure for $\mathcal{A}_S$.*

Finally, consider the theory $\mathcal{A}'$ axiomatized by (1), (2), (**inj**) and (**swp**), and let Γ and Γ' be selection functions such that $\Gamma(\mathrm{select}) = 2$ and $\Gamma'(\mathrm{select}) = 1$. Given the sets $A = \{C_i \vee p_i \mid i = 1,\dots,n\}$ and $B = \{D_k \vee p'_k \mid k = 1,\dots,m\}$, and for the selection functions Γ and Γ' defined above,

- $\langle \emptyset, A, \{(1),(2)\}\rangle$ is subterm-inactive by Lemma 4.4,
- $\langle \emptyset, B, \{(1),(2)\}\rangle$ is subterm-inactive by Lemma 4.6.

We deduce that the tuple $\langle \emptyset, A \cup B, \{(1),(2)\}\rangle$ is also subterm-inactive. We therefore have the following result:

Theorem 4.8 *A fair $\mathcal{SP}_{\succ}$-strategy is a decision procedure for $\mathcal{A}'$.*

4.3 A non-obvious example

The next example shows that although the conditions required for a tuple to be subterm-inactive are quite strong, some of them are tight, and allow us to point out some non-obvious results.

Example 4.9 Consider the following predicate:

$$\mathrm{Const}_y(x) \Leftrightarrow \forall z.\ \mathrm{select}(x,z) \simeq y,$$

that expresses the property that an array represents a constant function. It is easy to check that given two constants a and e, $\mathcal{T} = \mathcal{A} \cup \{\mathrm{Const}_e(a)\}$ is not subterm-inactive: there exists no Γ such that $\mathrm{Const}_e(a)$ is Γ-immune from any other set (condition (ec.1) does not hold), or Γ-saturation-closed (condition *(icp.1)* does not hold). Actually, $\mathcal{T}$ is not even variable-inactive; consider the following set:

$$S = \{\mathrm{store}(a,i,e_1) \simeq a',\ \mathrm{Const}_e(a),\ \mathrm{Const}_{e'}(a')\},$$
$$\Leftrightarrow \{\mathrm{store}(a,i,e_1) \simeq a',\ \mathrm{select}(a,z) \simeq e,\ \mathrm{select}(a',z) \simeq e'\}.$$

A superposition of the unit clause $\mathrm{store}(a, i, e_1) \simeq a'$ into the axiom $z \simeq w \vee \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w)$ yields the clause

$$w \simeq i \ \vee\ \mathrm{select}(a, w) \simeq \mathrm{select}(a', w). \tag{4}$$

Simplifications of this clause by $\mathrm{select}(a', z) \simeq e'$ and $\mathrm{select}(a, z) \simeq e$ yield the clause $w \simeq i \vee e \simeq e'$. This clause is not variable-inactive, and since it cannot be deleted, S_∞ is not variable-inactive either.

5 A collection of decision procedures

The approach based on subterm-inactivity allows us to re-obtain other termination results for $\mathcal{SP}$ on $\mathcal{T}$-satisfiability problems and generalize them to $\mathcal{T}$-decision problems.

5.1 *Finite sets with or without extensionality*

The theory of finite sets is based on the signature $\Sigma_{set} = \{\mathrm{member}, \mathrm{insert}\}$, where member and insert both have arity 2. Intuitively, $\mathrm{member}(e, s)$ is true if e is an element of the s, and $\mathrm{insert}(e, s)$ inserts element e into the set s. The theory is defined by the following presentation, denoted by $\mathcal{FS}$:

$$\forall x, v.\ \mathrm{member}(v, \mathrm{insert}(v, x)) \simeq \mathsf{true}, \tag{5}$$

$$\forall x, v, w.\ \ v \not\simeq w \Rightarrow \mathrm{member}(v, \mathrm{insert}(w, x)) \simeq \mathrm{member}(v, x). \tag{6}$$

The theory of finite sets with extensionality is presented by $\mathcal{FS}^{\mathrm{e}}$, which consists of axioms (5) and (6) along with the following extensionality axiom:

$$\forall x, y.\ (\forall v.(\mathrm{member}(v, x) \simeq \mathrm{member}(v, y))) \Rightarrow x \simeq y. \tag{7}$$

It was proved in [3, Theorem 8.1] that any $\mathcal{FS}^{\mathrm{e}}$-decision problem can be reduced to an $\mathcal{FS}$-decision problem. We have the following result:

Lemma 5.1 *$\langle \emptyset, \emptyset, \{(5), (6)\}\rangle$ is subterm-inactive.*

Proof. All one has to do is to verify that $\{(5), (6)\}$ is saturation-closed. This is the case, since the superposition of (5) into (6) generates $v \simeq v \vee \mathrm{member}(v, x) \simeq \mathsf{true}$, and this clause can be deleted by the Deletion inference rule. Thus, $\{(5), (6)\}$ is saturated, and it is simple to check that it is subsymbol-free from Σ^0 and that both clauses are ordered-flat. Let Γ be any function in Ω_Σ such that $\Gamma(\mathrm{member}) = 1$. Conditions *(icp.1)*, *(icp.2)* and *(icp.3)* hold on axiom (5), and conditions *(icp.1)*, *(icp.2)* and *(icp.4)* hold on axiom (6), so that $\{(5), (6)\}$ is Γ-internally closed. □

5.2 *Recursive data structures*

The class of recursive data structures includes the theory of integer offsets and the theory of acyclic lists. The members of this class are denoted $\mathcal{RDS}_k$, where k represents the number of *selectors* in the theory. The theory $\mathcal{RDS}_k$ is based on the following signature:

$$\begin{aligned} \Sigma_{\mathcal{RDS}_k} &= \{\mathsf{cons}\} \cup \Sigma_{sel}, \\ \Sigma_{sel} &= \{\mathsf{sel}_1, \ldots, \mathsf{sel}_k\}, \end{aligned}$$

where cons has arity k, and the sel_i's all have arity 1. The function symbols $\mathsf{sel}_1, \ldots, \mathsf{sel}_k$ stand for the *selectors*, and cons stands for the *constructor*. This theory is axiomatized by the following (infinite) set of axioms, denoted $Ax(\mathcal{RDS}_k)$:

$$\begin{aligned} \mathsf{sel}_i(\mathsf{cons}(x_1, \ldots, x_i, \ldots, x_k)) &\simeq x_i \quad \text{for } i = 1, \ldots, k \\ \mathsf{cons}(\mathsf{sel}_1(x), \ldots, \mathsf{sel}_k(x)) &\simeq x, \\ t[x] &\not\simeq x, \end{aligned}$$

where x and the x_i's are (implicitly) universally quantified variables and $t[x]$ is any compound Σ_{sel}-term where the variable x occurs. The axioms $t[x] \not\simeq x$ are termed *acyclicity axioms* and prevent the theory from entailing equations such as $\mathsf{sel}_1(\mathsf{sel}_2(x)) \simeq x$. For the sake of clarity, we also define the set

$$Ac(n) \;=\; \{\forall x.\, t[x] \not\simeq x \mid \mathrm{depth}(t) \leq n\}.$$

Example 5.2 Consider the case where $k = 2$. If we write $\mathrm{car}(x)$ instead of $\mathsf{sel}_1(x)$ and $\mathrm{cdr}(x)$ instead of $\mathsf{sel}_2(x)$, then our axioms become:

$$\begin{aligned} \mathrm{car}(\mathsf{cons}(x, y)) &\simeq x, \\ \mathrm{cdr}(\mathsf{cons}(x, y)) &\simeq y, \\ \mathsf{cons}(\mathrm{car}(x), \mathrm{cdr}(x)) &\simeq x, \\ t[x] &\not\simeq x, \end{aligned}$$

and the theory $\mathcal{RDS}_2$ is the theory of non-empty acyclic lists.

Consider the following axiom, denoted by (**ext**):

$$\forall x, y.\ x \simeq y \ \vee \left(\bigvee_{i=1}^{k} (\mathsf{sel}_i(x) \not\simeq \mathsf{sel}_i(y)) \right).$$

It was proved in [9] that a $\mathcal{T}$-satisfiability problem in $\mathcal{RDS}_k$ can be reduced to a $\mathcal{T}$-satisfiability problem in the theory defined by $\{(\mathbf{ext})\} \cup Ac(n)$, where n is computed by considering the number of constructors and selectors in the original set of ground literals (see [9, Definition 3.2 and Corollary 4.9] for details). We also have the following result:

Lemma 5.3 $\langle \emptyset, \{(\mathbf{ext})\}, Ac(n) \rangle$ *is subterm-inactive.*

Proof. Let Γ be a function in Ω_Σ such that for every $f \in \Sigma_{sel}$, $\Gamma(f) = 1$. We show that $\{(\mathbf{ext})\}$ is Γ-immune from $Ac(n)$. This set is trivially Γ-externally closed from $Ac(n)$ since every positive literal in (**ext**) is strictly flat. Also, (**ext**) is weakly flat and conditions *(vip.1)* and *(vip.2.a)* hold, so that $\{(\mathbf{ext})\}$ is Γ-variable-inactive preserving. Since $\{(\mathbf{ext})\} \cup Ac(n)$ only contains positive literals that are strictly flat, $\{(\mathbf{ext})\}$ is trivially Γ-interaction free from $Ac(n)$.

We now show that $Ac(n)$ is saturation-closed. It is simple to check that this set is saturated and subsymbol-free from Σ^0. Since it only contains unit clauses, all these clauses are trivially ordered-flat. The clauses in $Ac(n)$ are all of the form $t[x] \not\simeq x$, so that condition *(icn.1)* holds. Since these clauses are all negative, condition *(icn.2)* trivially holds. □

Theorem 5.4 *The superposition calculus yields $\mathcal{T}$-decision procedures for the following theories:*

- *Equality with Uninterpreted Functions (EUF).*
- *Arrays with or without extensionality, possibly augmented with an injectivity predicate, a swap predicate or both.*
- *Finite sets with or without extensionality.*
- *Recursive data structures.*

Proof. The result is obvious for the theory of equality with uninterpreted functions, which is presented by the empty set, and was shown for the variations of the theory of arrays in the previous section. Lemmas 5.1 and 5.3 prove the result for finite sets with or without extensionality and recursive data structures, respectively. □

6 Discussion

In this paper, we introduced the notion of subterm-inactive theory, that guarantees that $\mathcal{SP}$ yields $\mathcal{T}$-decision procedures. Almost all the conditions for subterm-inactivity are static, which means they can be tested automatically and only once. We showed that several theories, including most of those considered in [3,1], and two extensions of the theory of arrays, satisfy these conditions, which indicates that they are not too strong. Still, some of the theories of [1] are not subterm-inactive. They are the theory of possibly empty lists, the theory of records and the theory of integer offsets modulo. We intend to investigate how to weaken the subterm-inactivity conditions to obtain a larger class of subterm-inactive theories.

The subterm-inactivity condition guarantees the termination of any fair $\mathcal{SP}_{\succ}$-strategy, but the efficiency of such an approach to $\mathcal{T}$-decision problems of practical interest still has to be tested. It would be especially relevant to investigate how well such a generic approach can manage $\mathcal{T}$-decision problems with a large boolean part.

Another important issue is how to combine subterm-inactive theories with Presburger arithmetic, especially for the theory of arrays. Indeed, using Presburger arithmetic on indices allows one to work on more complex properties about arrays, such as testing whether subarrays are identical.

Acknowledgement

The authors wish to thank Alessandro Armando and Silvio Ranise for bringing injective arrays to their attention.

References

[1] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. On a rewriting approach to satisfiability procedures: Extension, combination of theories and an experimental appraisal. In Bernhard Gramlich, editor, *Proc. 5th FroCoS*, volume 3717 of *LNAI*, pages 65–80. Springer, 2005. The full version is available at http://profs.sci.univr.it/~bonacina/rewsat.html.

[2] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Online Proc. SAT-7*, 2004.

[3] A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Info. and Comp.*, 183(2):140–164, June 2003.

[4] Clark W. Barrett and Sergey Berezin. CVC lite: A new implementation of the Cooperating Validity Checker. In Rajeev Alur and Doron Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, pages 515–518. Springer, 2004.

[5] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. MathSAT: Tight integration of SAT and mathematical decision procedures. *J. of Autom. Reason.*, 35(1–3):265–293, Oct. 2005.

[6] Clark W. Barrett, David L. Dill, and Aaron Stump. A framework for cooperating decision procedures. In David A. McAllester, editor, *Proc. CADE-17*, volume 1831 of *LNAI*, pages 79–98. Springer, 2000.

[7] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Kim G. Larsen and Ed Brinksma, editors, *Proc. CAV-14*, volume 2404 of *LNCS*, pages 236–249. Springer, 2002.

[8] M. P. Bonacina and M. Echenim. Generic theorem proving for decision procedures. Technical Report RR 41/2006, Università degli studi di Verona, 2006. Full version available at http://profs.sci.univr.it/~echenim/.

[9] Maria Paola Bonacina and Mnacho Echenim. Rewrite-based satisfiability procedures for recursive data structures. In Byron Cook and Roberto Sebastiani, editors, *Proceedings of the Fourth Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR), Third International Joint Conference on Automated Reasoning (IJCAR) and Fourth Federated Logic Conference (FLoC)*, Electronic Notes in Theoretical Computer Science. Elsevier, August 2006. To appear.

[10] Leonardo de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. In Andrei Voronkov, editor, *Proc. CADE-18*, volume 2392 of *LNAI*, pages 438–455. Springer, 2002.

[11] Nachum Dershowitz and David A. Plaisted. Rewriting. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 535–610. Elsevier Science Publishers, 2001.

[12] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In Rajeev Alur and Doron A. Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.

[13] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.

[14] Alexandre Riazanov and Andrei Voronkov. Splitting without backtracking. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 611–617. Morgan Kaufmann, 2001.

[15] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A Cooperating Validity Checker. In Rajeev Alur and Doron Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, pages 500–504. Springer, 2004.