HAL
open science

A Calculus for Generating Ground Explanations

Mnacho Echenim, Nicolas Peltier

To cite this version:

Mnacho Echenim, Nicolas Peltier. A Calculus for Generating Ground Explanations. IJCAR 2012 - International Joint Conference on Automated Reasoning, Jun 2012, Manchester, United Kingdom. pp.194-209, 10.1007/978-3-642-31365-3_17 . hal-00933272

HAL Id: hal-00933272

https://hal.science/hal-00933272v1

Submitted on 20 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Calculus for Generating Ground Explanations*

Mnacho Echenim and Nicolas Peltier

University of Grenoble** (LIG, Grenoble INP/CNRS)

Abstract. We present a modification of the superposition calculus that is meant to generate explanations why a set of clauses is satisfiable. This process is related to abductive reasoning, and the explanations generated are clauses constructed over so-called abductive constants. We prove the correctness and completeness of the calculus in the presence of redundancy elimination rules, and develop a sufficient condition guaranteeing its termination; this sufficient condition is then used to prove that all possible explanations can be generated in finite time for several classes of clause sets, including many of interest to the SMT community. We propose a procedure that generates a set of explanations that should be useful to a human user and conclude by suggesting several extensions to this novel approach.

1 Introduction

The verification of complex systems is generally based on proving the validity, or, dually, the satisfiability of a logical formula. The standard practice consists in translating the behavior of the system to be verified into a logical formula, and proving that the negation of the formula is unsatisfiable. These formulas may be domain-specific, so that it is only necessary to test the satisfiability of the formula modulo some background theory, whence the name *Satisfiability Modulo Theories problems*, or *SMT problems*. If the formula is actually satisfiable, this means the system is not error-free, and any model can be viewed as a trace that generates an error. The models of a satisfiable formula can therefore help the designers of the system guess the origin of the errors and deduce how they can be corrected. Yet, this still requires some work. Indeed, there are generally many interpretations on different domains that satisfy the formula, and it is necessary to further analyze these models to understand where the error(s) may come from.

We present what is, to the best of our knowledge, a novel approach to this debugging problem: we argue that rather than studying one model of a formula, more valuable information can be extracted from the properties that hold in *all* the models of the formula. For instance, consider the theory of arrays, which is

* This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

** emails: `Mnacho.Echenim@imag.fr`, `Nicolas.Peltier@imag.fr`

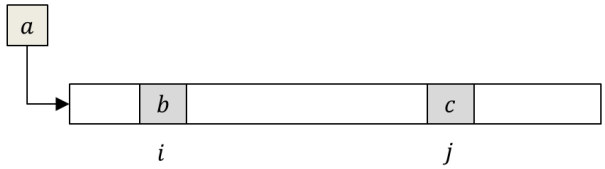

Fig. 1. Insertion into array a of element b at position i and element c at position j.

axiomatized as follows (as introduced by [13]):

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

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

These axioms state that if element v is inserted into array x at position z, then the resulting array contains v at position z, and the same elements as in x elsewhere. Assume that to verify that the order in which elements are inserted into a given array does not matter, the satisfiability of the following formula is tested (see also Figure 1):

$$\mathrm{select}(\mathrm{store}(\mathrm{store}(a, i, b), j, c), k) \not\simeq \mathrm{select}(\mathrm{store}(\mathrm{store}(a, j, c), i, b), k).$$

This formula asserts that there is a position k that holds different values in the array obtained from a by first inserting element b at position i and then element c at position j, and in the array obtained from a by first inserting element c at position j and then element b at position i. It turns out that this formula is actually satisfiable, which in this case means that some hypotheses are missing. State of the art SMT solvers such as Yices [16] can help find out what hypotheses are missing by outputting a model of the formula. In this case, Yices outputs `(= b 1) (= c 3) (= i 2) (= k 2) (= j 2)`, and for this simple example, such a model may be sufficient to quickly understand where the error comes from. However, a simpler and more natural way to determine what hypotheses are missing would be to have a tool that, when fed the formula above, outputs $i \simeq j \wedge b \not\simeq c$, stating that the formula can only be true when elements b and c are distinct, and are inserted at the *same* position in a. This information permits to know immediately what additional hypotheses must be made for the formula to be unsatisfiable. In this example, there are two possible hypotheses that can be added: $i \not\simeq j$ or $b \simeq c$.

In this paper, we investigate what information should be provided to the user and how it can be obtained, by distinguishing a set of constants on which additional hypotheses are allowed to be made. These constants are called *abducible constants* or simply *abducibles*, and the problem boils down to determining what ground clauses containing only abducibles are logically entailed by the formula under consideration, since the negation of any of these clauses can be viewed as a set of additional hypotheses that make the formula unsatisfiable.

Outline. This paper begins by summarizing all necessary background, and then a calculus specially designed for abductive reasoning is defined. This calculus is closely related to the *superposition calculus* $\mathcal{SP}$, and we rely on completeness and termination results for $\mathcal{SP}$ to prove similar results for the new calculus. We also propose a method for generating clauses containing only abducible constants, that can help a user quickly detect where an error comes from, and decide what additional hypotheses should be added to fix the faulty formula.

Due to the space restrictions, several intermediate results and proofs are omitted. A full version of this work containing all proofs is available in [8].

2 Preliminaries

The general framework of this paper is first-order logic with equality. Most of the presentation in this section is standard, and we refer the reader to [14] for details. Given a finite signature Σ and an integer $i \geq 0$, Σ^i stands for the set of function symbols in Σ of arity i. In particular, Σ^0 denotes the set of constants in Σ. We assume the standard definitions of terms, predicates, literals and clauses, all of which are constructed over a set of variables $\mathcal{X}$. Interpretations are defined as usual, $\models$ stands for logical entailment and $\equiv$ stands for logical equivalence. We also consider the standard definitions of positions in terms, predicates, literals or clauses. A term, predicate, literal or clause containing no variable is *ground*. As usual, clauses are assumed to be variable-disjoint. The symbol $\simeq$ stands for unordered equality, $\bowtie$ is either $\simeq$ or $\not\simeq$. If L is a literal, then L^c denotes the complementary literal of L, i.e., $(t \simeq s)^c \stackrel{\text{def}}{=} (t \not\simeq s)$ and $(t \not\simeq s)^c \stackrel{\text{def}}{=} (t \simeq s)$. A literal is *flat* if it only contains constants or variables[1], and a clause is *flat* if it only contains flat literals. The letters l, r, s, u, v and t denote terms, w, x, y, z variables, and all other lower-case letters denote constants or function symbols.

Definition 1. *Given a ground clause C, we denote by $\neg C$ the following set of literals:* $\neg C \stackrel{def}{=} \{L^c \mid L \in C\}$.

A *substitution* is a function mapping variables to terms. Given a substitution σ, the set of variables x such that $x\sigma \neq x$ is called the *domain* of σ and denoted by $dom(\sigma)$. If σ is a substitution and V is a set of variables, then $\sigma_{|V}$ is the substitution with domain $dom(\sigma) \cap V$, that matches σ on this domain. As usual, a substitution can be extended into a homomorphism on terms, atoms, literals and clauses. The image of an expression $\mathcal{E}$ by a substitution σ will be denoted by $\mathcal{E}\sigma$. If E is a set of expressions, then $E\sigma$ denotes the set $\{\mathcal{E}\sigma \mid \mathcal{E} \in E\}$. The composition of two substitutions σ and θ is denoted by $\sigma\theta$. A substitution σ is *more general* than θ if there exists a substitution η such that $\theta = \sigma\eta$. The substitution σ is a *renaming* if it is injective and $\forall x \in dom(\sigma), x\sigma \in \mathcal{X}$; and it is a *unifier* of two terms t, s if $t\sigma = s\sigma$. Any unifiable pair of terms (t, s) has a most general unifier, unique up to a renaming, and denoted by $\text{mgu}(t, s)$. A substitution σ is *ground* if $x\sigma$ is ground, for every variable x in its domain.

[1] Note that we depart from the terminology in [2, 1], where flat positive literals can contain a term of depth 1.

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): $u\sigma \not\prec t\sigma$;
(ii): $\forall L \in D : (u \simeq t)\sigma \not\prec L\sigma$;
(iii): $l[u']\sigma \not\prec r\sigma$;
(iv): $\forall L \in C : (l[u'] \bowtie r)\sigma \not\prec L\sigma$;
(v): $\forall L \in C : (u' \simeq u)\sigma \not\prec L\sigma$;
(vi): $\forall L \in \{u' \simeq t'\} \cup C : (u \simeq t)\sigma \not\prec L\sigma$.

Fig. 2. Inference rules of $\mathcal{SP}$: the clause below the inference line is added to the clause set containing the clauses above the inference line.

A *simplification ordering* $\prec$ is an ordering that is stable under substitutions, monotonic, and contains the subterm ordering: if $s \prec t$, then $c[s]\sigma \prec c[t]\sigma$ for any context c and substitution σ, and if s is a strict subterm of t then $s \prec t$. A *complete simplification ordering*, or CSO, is a simplification ordering that is total on ground terms. Similarly to [7], in the sequel, we shall assume that any CSO under consideration is *good*:

Definition 2. *A CSO $\prec$ is* good *if for all ground compound terms t and constants c, we have $c \prec t$.*

The *superposition calculus*, or $\mathcal{SP}$ (see, e.g., [14]), is a refutationally complete rewrite-based inference system for first-order logic with equality. It consists of the inference rules summarized in Fig. 2: each rule contains *premises* which are above the inference line, and generates a *conclusion*, which is below the inference line. If a clause D is generated from premises C, C', then we write $C, C' \vdash D$. The superposition calculus is based on a CSO on terms, which is extended to literals and clauses in a standard way (see, e.g., [3]), and we may write $\mathcal{SP}_{\prec}$ and $\vdash_{\prec}$ to specify the ordering. A ground clause C is *$\prec$-redundant in S*, or simply redundant, if there exists a set of ground clauses S' such that $S' \models C$, and for every $D \in S'$, D is an instance of a clause in S and $D \prec C$. A non-ground clause C is *$\prec$-redundant in S* if all its instances are $\prec$-redundant in S. In particular, every strictly subsumed clause and every tautological clause is redundant. A set of clauses S is *saturated* if every clause $C \notin S$ generated from premises in S is

redundant in S. A saturated set of clauses that does not contain $\square$ is satisfiable [14]. In practice, it is necessary to use a decidable approximation of this notion of redundancy: for example, a clause is redundant if it can be reduced by some demodulation steps to either a tautology or to a subsumed clause.

In the sequel, it will be necessary to forbid the occurrence of clauses containing maximal literals of the form $x \simeq t$, where $x \not\preceq t$:

Definition 3. *A clause is* variable-eligible w.r.t. $\prec$ *if it contains a maximal literal of the form $x \simeq t$, where $x \not\preceq t$. A set of clauses is* variable-inactive *(see [1]) if no non-redundant clause generated from S is variable-eligible.*

For technical reasons we have chosen to present a slightly relaxed version of the superposition calculus, in which the standard strict maximality conditions have been replaced by non-strict maximality conditions. For instance in Condition (i), $u\sigma \not\preceq t\sigma$ is replaced by $u\sigma \not\prec t\sigma$: it is not forbidden for u and t to be identical in *Paramodulation* and *Superposition* inferences. It is clear that the clauses generated in the case where there is an equality actually turn out to be redundant.

3 A calculus for handling abducible constants

As explained in the Introduction, the aim of this paper is to start with a formula F and a set of axioms A, and generate a formula H which logically entails F modulo A, i.e., such that $H, A \models F$ (where $H \wedge A$ is satisfiable). As usual in abductive reasoning (see for instance [9]), we actually consider the contrapositive: since $H, A \models F$ is equivalent to $\neg F, A \models \neg H$, the original problem can be solved by generating logical consequences of the formula $\neg F \wedge A$. For the sake of simplicity, the formula $\neg F$ is added to the axioms which are assumed to be in clausal form, and we have the following definition:

Definition 4. *A clause C is an* implicate *of a set of clauses S iff $S \models C$.*

It is clear that after its generation, it is necessary to verify that H is satisfiable modulo A. For instance, if a is some constant, then an explanation such as $a \simeq 0 \wedge a \simeq 1$ or even $0 \simeq 1$ does not provide any information since it contradicts the axioms of Presburger arithmetic. Testing this satisfiability can be done using standard decision procedures. There are many possible candidate sets of implicates, which may be more or less informative. For instance, it is possible to take $C \in S$, but this is obviously of no use. Thus it is necessary to provide additional information in order to restrict the class of formulas that are searched for. In (propositional) abductive reasoning, this is usually done by considering clauses built on a given set of literals: the *abducible literals*. A more natural possibility in the context of this paper is to consider clauses built on a given set of ground terms. We may assume with no loss of generality that each of these terms is replaced by a constant symbol, by applying the usual flattening operation, see, e.g., [2, 7]. For example, the term $\mathrm{select}(\mathrm{store}(a, i, b), j)$ may be replaced by a new constant d, along with the axioms: $d \simeq \mathrm{select}(d', j) \wedge d' \simeq \mathrm{store}(a, i, b))$. We thus consider a distinguished set of constants $\mathcal{A} \subseteq \Sigma^0$, called the *set of abducible*

constants, and restrict ourselves to explanations that are conjunctions of literals built upon abducible constants. This is formalized with the following definition of an $\mathcal{A}$-implicate:

Definition 5. *Let S be a set of clauses. A clause C is an* $\mathcal{A}$-implicate *of S iff every term occurring in C is also in $\mathcal{A}$ and $S \models C$.*

As in propositional abductive reasoning, the set $\mathcal{A}$ must be provided by the user. Given a set of clauses S containing both the axioms A and the clauses corresponding to the conjunctive normal form of $\neg F$, we investigate how to generate the set of flat ground clauses C built on $\mathcal{A}$, that are logical consequences of S. Since $\mathcal{SP}$ is only *refutationally* complete, this cannot be done directly using this calculus (except in some very particular cases, see for instance [15]). For example, it is clear that $f(a) \not\simeq f(b) \models a \not\simeq b$, but $a \not\simeq b$ cannot be generated from the antecedent clause. In principle, it is possible to enumerate all possible clauses C built on $\mathcal{A}$ and then use the superposition calculus to check whether $S \cup \neg C$ is unsatisfiable, however, this yields a very inefficient procedure. An alternate method consists in replacing the superposition calculus by a less restrictive calculus, such as the Resolution calculus [11] together with the equality axioms. For instance in the previous case, the clause $f(a) \not\simeq f(b)$ and the substitutivity axiom $x \not\simeq y \vee f(x) \simeq f(y)$ permit to generate by the Resolution rule: $a \not\simeq b$. However, again, this calculus is not efficient, and in particular all the termination properties of the superposition calculus on many interesting subclasses of first-order logic [4, 2, 1] are lost In this section, we provide a variant of the superposition calculus which is able to *directly* generate, from a set of clauses S, a set of logical consequences of S that are built on a given set of constant symbols $\mathcal{A}$. The calculus is thus parameterized both by the term ordering $\prec$ and by the set of abducible constants $\mathcal{A}$. We shall show that the calculus is complete, in the sense that if $S \models C$ and if C is an $\mathcal{A}$-implicate of S, then C is a logical consequence of other clauses built on $\mathcal{A}$ that are generated from S. We will also prove that the calculus terminates on many classes of interest in the SMT community.

We will thus consider clauses of a particular form and a slight variation of the superposition calculus in order to be able to reason on abducible constants. The principle behind this calculus is similar to that of [5] for the combination of *hierarchic theories*, with the difference that in this framework, abducible constants can potentially interact with other terms, whereas in the framework of [5], such an interaction is prevented by sortedness. In both settings however, a same *abstraction* principle is used to delay the reasoning on the objects of interest (in this case, the abducible constants).

From now on we assume that the set of variables $\mathcal{X}$ is of the form $\mathcal{X} = \mathcal{V} \uplus \mathcal{V}_{\mathcal{A}}$. The elements in $\mathcal{V}$ are ordinary variables and the elements in $\mathcal{V}_{\mathcal{A}}$ are called *abducible variables*, and they will serve as placeholders for abducible constants in terms and clauses. In the sequel, when we mention *standard* terms, literals or clauses, we assume that all the variables they contain are in $\mathcal{V}$.

Definition 6. *An* $\mathcal{A}$-literal *is a literal of the form $t \bowtie s$, where $t, s \in \mathcal{V}_{\mathcal{A}} \cup \mathcal{A}$. An* $\mathcal{A}$-clause *is a disjunction of $\mathcal{A}$-literals. Given a clause C, we denote by $\Delta(C)$*

the disjunction of $\mathcal{A}$-literals in C and by $\overline{\Delta}(C)$ the disjunction of non-$\mathcal{A}$-literals in C. We define $\mathrm{Var}_{\mathcal{A}}(C) \stackrel{def}{=} \mathrm{Var}(C) \cap \mathcal{V}_{\mathcal{A}}$.

A first step towards reasoning on abducible constants will consist in extracting them from the terms in which they occur, and replacing them by abducible variables. Then, to ensure that such a property is preserved by inferences, every substitution mapping an abducible variable to anything other than an abducible variable will be discarded. More formally:

Definition 7. *A term is* abstracted *if it contains no abducible constant. A literal $t \bowtie s$ is* abstracted *if t and s are both abstracted. A clause is* abstracted *if all non-abstracted literals in C are $\mathcal{A}$-literals.*

If t is an abstracted term, then not every instance of t is also abstracted. We define a condition on substitutions that guarantees such a stability result.

Definition 8. *A substitution σ is* $\mathcal{A}$-compliant *if for all $x \in dom(\sigma)$, $x\sigma$ is abstracted, and for all $x \in dom(\sigma) \cap \mathcal{V}_{\mathcal{A}}$, $x\sigma \in \mathcal{V}_{\mathcal{A}}$. Two abstracted terms are* $\mathcal{A}$-unifiable *if they are unifiable and admit an $\mathcal{A}$-compliant mgu.*

In the sequel, every time abstracted terms are $\mathcal{A}$-unifiable, we will assume the corresponding mgu is $\mathcal{A}$-compliant.

Definition 9. *Let $<_{\mathcal{A}}$ be a total ordering on $\mathcal{A}$ and a_0 denote the smallest abducible in $\mathcal{A}$. Given a term t, we denote by $t_{\downarrow\mathcal{A}}$ the term obtained by replacing every abducible constant occurring in t by a_0. The term t is* $\mathcal{A}$-reduced *if $t_{\downarrow\mathcal{A}} = t$. The previous notation and this definition extend to literals, clauses and sets of clauses.*

Example 1. Let $C = f(b,c) \simeq g(d) \vee x \not\simeq b \vee f(a,b) \not\simeq f(c,d)$, where $\mathcal{A} = \{a,b,c\}$ and $a \prec b \prec c$. Then $C_{\downarrow\mathcal{A}} = f(a,a) \simeq g(d) \vee x \not\simeq a \vee f(a,a) \not\simeq f(a,d)$, and this clause is an $\mathcal{A}$-reduced clause.

It is clear that if all abducible constants are replaced by abducible variables in a standard clause, then the resulting abstracted clause is not equivalent to the former one. However, equivalence can be regained by adding so-called $\mathcal{V}_{\mathcal{A}}$-*constraint literals* to the resulting abstracted clause.

Definition 10. *A* $\mathcal{V}_{\mathcal{A}}$-constraint literal *is a literal of the form $x \not\simeq a$, where $x \in \mathcal{V}_{\mathcal{A}}$ and $a \in \mathcal{A}$. For all clauses C, we denote by $\Gamma(C)$ the disjunction of $\mathcal{V}_{\mathcal{A}}$-constraint literals in C. A* $\mathcal{V}_{\mathcal{A}}$-constraint clause *is a disjunction of $\mathcal{V}_{\mathcal{A}}$-constraint literals. Given a $\mathcal{V}_{\mathcal{A}}$-constraint clause $A = \bigvee_{i=1}^{k} x_i \not\simeq a_i$, the* substitution associated to A *is denoted by ν_A and defined as follows: $dom(\nu_A) = \{x_1, \ldots, x_k\}$, and for all $x \in dom(\nu_A)$, $x\nu_A = \min_{<_{\mathcal{A}}} \{a_i \mid x_i = x\}$.*

For readability, if B is a clause then we will write ν_B instead of $\nu_{\Gamma(B)}$. If S is a set of abstracted clauses, then S_ν is the set $S_\nu = \{C\nu_C \mid C \in S\}$.

Example 2. Assume $\mathcal{A} = \{a,b,c\}$, where $a <_{\mathcal{A}} b <_{\mathcal{A}} c$, and let $A = x \not\simeq a \vee x \not\simeq c \vee y \not\simeq b \vee z \not\simeq a \vee y \not\simeq c$. Then $\nu_A = \{x \mapsto a, y \mapsto b, z \mapsto a\}$.

Note that by definition, $C \equiv C\nu_C$ and $S \equiv S_\nu$. As mentioned earlier, abducible variables are meant to be placeholders for abducible constants. In general, it will be necessary to keep some information permitting to know what abducible constants an abducible variable could be replaced by. Such a requirement is satisfied by imposing that every abducible variable occurs in at least one $\mathcal{V}_\mathcal{A}$-constraint literal, which intuitively specifies its value.

Definition 11. *A clause C is* $\mathcal{V}_\mathcal{A}$-stable *if* $\mathrm{Var}_\mathcal{A}(C) \subseteq \mathrm{Var}_\mathcal{A}(\Gamma(C))$. *A set of clauses is* $\mathcal{V}_\mathcal{A}$-stable *if every clause it contains is $\mathcal{V}_\mathcal{A}$-stable.*

Given a set of standard clauses, it is easy to construct an equivalent set of abstracted and $\mathcal{V}_\mathcal{A}$-stable clauses. It suffices to replace every abducible a occurring in a non-$\mathcal{A}$-literal by a fresh variable $x \in \mathcal{V}_\mathcal{A}$, and to add the literal $x \not\simeq a$ to the clause. For instance, if $\mathcal{A} = \{a, b\}$ then the clause $a \simeq b \vee a \simeq c \vee f(b, d, x) \not\simeq g(b, y)$ is replaced by $x_1 \not\simeq a \vee x_2 \not\simeq b \vee x_3 \not\simeq b \vee a \simeq b \vee x_1 \simeq c \vee f(x_2, d, x) \not\simeq g(x_3, y)$.

Definition of the calculus. We introduce a calculus for generating $\mathcal{A}$-implicates. It is a modified version of the superposition calculus, and consists of inference rules that are meant to be applied to abstracted clauses. In particular, it is based on orderings that are suitable for abstracted terms, literals and clauses: the order between two terms t and s should not depend on the abducible constants occurring in t and s, and maximal terms and literals in abstracted clauses should be related to maximal terms and literals in standard clauses, in a sense that will be made precise later. We thus define particular orderings for standard clauses, from which we define suitable orderings for abstracted clauses.

Definition 12. *We consider a good CSO $\prec$ such that*[2]*:*

1. *for all $a, b \in \mathcal{A}$, $a \prec b$ if and only if $a <_\mathcal{A} b$;*
2. *for all $a \in \mathcal{A}$ and for all non-variable terms $t \notin \mathcal{A}$, $a \prec t$;*
3. *for all ground terms t, s not in $\mathcal{A}$, if $t \prec s$ then $t_{\downarrow\mathcal{A}} \preceq s_{\downarrow\mathcal{A}}$, and if $t_{\downarrow\mathcal{A}} \prec s_{\downarrow\mathcal{A}}$ then $t \prec s$.*

We let γ_0 denote the ground substitution of domain $\mathcal{V}_\mathcal{A}$ such that for all $x \in \mathcal{V}_\mathcal{A}$, $x\gamma_0 = a_0$. Given abstracted terms t, s, we define $\prec_\mathcal{A}$ as follows: $t \prec_\mathcal{A} s$ iff $t\gamma_0 \prec s\gamma_0$. This definition extends to literals and clauses in a standard way. A term is $\mathcal{A}$-maximal *if it is maximal for $\prec_\mathcal{A}$; this definition also extends to literals and clauses.*

Definition 13. *We denote by $\mathcal{SP}_\mathcal{A}$ the calculus such that for all clause sets S, we have $S \vdash^\mathcal{A} D$ if $S \vdash_{\prec_\mathcal{A}} D$ and the mgu involved in the $\mathcal{SP}$-inference is $\mathcal{A}$-compliant.*

By construction, $\mathcal{SP}$ and $\mathcal{SP}_\mathcal{A}$ coincide on ground $\mathcal{A}$-clauses. We define a particular notion of redundancy for abstracted clauses, that is related to redundancy for standard clauses. The main difference with the standard definition is that the redundancy test is performed modulo the substitution ν_C that replaces the abstracted variables in C by the abducible constants they denote.

[2] It is not difficult to see that there exist orderings fulfilling these properties (see [8]).

Definition 14. *Consider a set of abstracted clauses S and an abstracted clause C such that* $\mathrm{Var}(C) \subseteq \mathcal{V}_{\mathcal{A}}$. *The clause C is* $\mathcal{A}$-redundant in S *if:*

- *C is an $\mathcal{A}$-clause, $\nu_C \neq \mathrm{id}$ and $C\nu_C$ either occurs or is $\mathcal{A}$-redundant in S,*
- *or there exists a set of ground clauses S' such that $S' \models C$, every $D \in S'$ is an instance of a clause in S_ν and $D \prec C\nu_C$.*

If C is an abstracted clause such that $\mathrm{Var}(C) \not\subseteq \mathcal{V}_{\mathcal{A}}$, *then C is* $\mathcal{A}$-redundant in S *if for all ground substitutions σ with a domain in $\mathcal{V}$, $C\sigma$ is $\mathcal{A}$-redundant in S. The set S is* $\mathcal{A}$-saturated *if every clause $C \notin S$ generated by an $\mathcal{SP}_{\mathcal{A}}$-inference with premises in S is $\mathcal{A}$-redundant in S.*

This notion of redundancy permits to add the standard contraction rules of the superposition calculus to $\mathcal{SP}_{\mathcal{A}}$ (subsumption, simplification, elimination of tautologies, etc). The following contraction inference rule is also added to $\mathcal{SP}_{\mathcal{A}}$:

$$\mathcal{A}\text{-reduction}: \quad \frac{C}{\overline{C\nu_C}} \quad \text{if } C \text{ is an } \mathcal{A}\text{-clause and } \nu_C \neq \mathrm{id}.$$

After any application of the $\mathcal{A}$-reduction rule, the premise becomes $\mathcal{A}$-redundant and can be deleted.

Theorem 1. *If S is a variable-inactive (w.r.t. $\prec_{\mathcal{A}}$) set of abstracted clauses that are $\mathcal{V}_{\mathcal{A}}$-stable, then every non-redundant clause generated from S by $\mathcal{SP}_{\mathcal{A}}$ is abstracted and $\mathcal{V}_{\mathcal{A}}$-stable. Also, if one of the premises of a binary $\mathcal{SP}_{\mathcal{A}}$-inference is an $\mathcal{A}$-clause, then the other premise is also an $\mathcal{A}$-clause.*

The variable-inactive condition, which ensures that all generated clauses are variable-eligible, prevents non-abstracted clauses from being generated from abstracted ones. For example, if S contains the unit clauses $\{a \simeq b, x \simeq y\}$ with $\{a, b\} \in \mathcal{A}$ and $\{x, y\} \in \mathcal{V}$, then $S \vdash^{\mathcal{A}} y \simeq b$, and the latter is *not* abstracted. In what follows, we will prove completeness and termination results for $\mathcal{SP}_{\mathcal{A}}$. The completeness result guarantees that $\mathcal{SP}_{\mathcal{A}}$ generates the required information about existing abducibles for any abstracted set of clauses, while the termination result relies on termination results for $\mathcal{SP}$, and will be used to verify without any additional effort that our technique can be used as a decision procedure for reasoning about abducibles in SMT problems with several theories of interest.

4 Completeness of the calculus

This section is devoted to showing that if S is an unsatisfiable set of abstracted clauses that is $\mathcal{A}$-saturated, then $\square \in S$ (due to space restrictions, we only provide a sketch of the proof, see [8] for details). Note that this result does *not* follow from the refutational completeness of the superposition calculus: indeed, the ordering $\prec_{\mathcal{A}}$ is not a simplification ordering (it is not stable by substitution), and all inferences in which non-$\mathcal{A}$-compliant unifiers are involved are ignored. However, the proof is based on the refutational completeness of $\mathcal{SP}$, and requires determining relationships between $\mathcal{SP}$-inferences and $\mathcal{SP}_{\mathcal{A}}$-inferences.

Let S be a $\mathcal{V}_{\mathcal{A}}$-stable and $\mathcal{A}$-saturated set of clauses, with no variable-eligible clause. We will show that S is satisfiable by constructing a set of standard clauses whose satisfiability will entail that of S. The set we construct will be saturated under $\mathcal{SP}_{\prec}$-inferences, and it will not contain the empty clause; we will conclude that it must be satisfiable, and hence that so must S.

Let T be the set of $\mathcal{A}$-clauses in S. Since S is $\mathcal{V}_{\mathcal{A}}$-stable and $\mathcal{A}$-saturated by hypothesis, T can only contain ground $\mathcal{A}$-clauses, because if a non-ground clause occurs in T then $\mathcal{A}$-reduction applies. Since $\mathcal{SP}$ and $\mathcal{SP}_{\mathcal{A}}$ coincide on ground $\mathcal{A}$-clauses, T must also be saturated under $\mathcal{SP}_{\prec}$-inferences and cannot contain $\square$; this set is therefore satisfiable. We consider a fixed interpretation I that is a model of T.

Definition 15. *We define the ground set* $U_I = \{a \simeq b \mid a, b \in \mathcal{A}, a^I = b^I\} \cup \{a \not\simeq b \mid a, b \in \mathcal{A}, a^I \neq b^I\}$. *We inductively define the notion of an* I-reduction*:*

- *For all* $a \in \mathcal{A}$, $a_{\|I} = \min_{\prec} \{b \in \mathcal{A} \mid b^I = a^I\}$.
- $f(t_1, \ldots, t_n)_{\|I} = f(t_{1\|I}, \ldots, t_{n\|I})$.

This definition extends to standard literals and clauses.

The I-reduction procedure is used to define a set whose satisfiability entails that of S, and that turns out to be saturated:

Definition 16. *Let* $S_I = U_I \cup \{\overline{\Delta}(C_{\|I}) \mid C \in S_\nu \wedge U_I \models \neg\Delta(C)\}$.

Proposition 1. *If* S_I *is satisfiable then so is* S_ν*, and therefore so is* S*.*

Lemma 1. S_I *is saturated for* $\mathcal{SP}_{\prec}$*.*

Since S_I is saturated for the standard superposition calculus $\mathcal{SP}_{\prec}$ and contains no occurrence of the empty clause, we deduce that it is satisfiable.

Theorem 2. *Let* S *be a set of abstracted clauses that is* $\mathcal{V}_{\mathcal{A}}$*-stable and contains no variable-eligible clause. If* S *is* $\mathcal{A}$*-saturated and does not contain the empty clause, then* S *is satisfiable.*

This theorem proves the refutational completeness of $\mathcal{SP}_{\mathcal{A}}$ together with contraction rules that eliminate $\mathcal{A}$-redundant clauses, for those sets of abstracted clauses S whose saturation is guaranteed to meet the requirements of the theorem. The first two requirements are not restrictive: the abstraction of a set of standard clauses described right before Section 3 produces a set of abstracted and $\mathcal{V}_{\mathcal{A}}$-stable clauses, and the saturation of this set is guaranteed to only contain abstracted and $\mathcal{V}_{\mathcal{A}}$-stable clauses by Theorem 1. The fact that S contains no variable-eligible clause cannot be imposed that easily, but such a condition is guaranteed if S is variable-inactive, which is the case for many classes of clause sets of interest [2, 1].

Note that this completeness result is not – by itself – sufficient for our purpose, since our goal is not merely to test the satisfiability of clause sets but rather to generate flat consequences they logically entail. The next section shows how the calculus $\mathcal{SP}_{\mathcal{A}}$ can be employed to reach this goal.

5 A generation of explanations

We return to the problem of explaining why a set of clauses is satisfiable, and show how $\mathcal{SP}_{\mathcal{A}}$ can be used to generate explanations relating abducibles to one another. Given a satisfiable set of clauses S', we denote by $\mathrm{I}_{\mathcal{A}}(S')$ the set of all $\mathcal{A}$-implicates of S': $\mathrm{I}_{\mathcal{A}}(S') \stackrel{\text{def}}{=} \{C \text{ an } \mathcal{A}\text{-clause} \mid C \text{ is ground and } S' \models C\}$.

It is clear that all the information about abducible constants that is entailed by S' is contained in $\mathrm{I}_{\mathcal{A}}('S)$. However this set can be very large and it contains a lot of non-pertinent information, for example all logical tautologies, or all instances of the equality axioms. It therefore does not seem reasonable to return this entire set to a user. Another solution could be to return a subset $T \subseteq \mathrm{I}_{\mathcal{A}}(S')$ such that $T \vdash \mathrm{I}_{\mathcal{A}}(S')$, but again, such a set might be large and contain unnecessary information. The solution we choose is to return a minimal subset $T' \subseteq \mathrm{I}_{\mathcal{A}}(S')$ satisfying the following property: for all $C \in \mathrm{I}_{\mathcal{A}}(S')$ that is not a tautology, there exists a clause $C' \in T'$ such that $C' \models C$. The clauses in T' are the *prime implicates* of S'. The notion of prime implicates plays a central rôle in many applications of computer science and artificial intelligence, and several approaches have been proposed for computing the prime implicates of a given propositional formula (see, e.g., [10]). Some extensions to first-order logic have also been considered, such as, e.g., [12]. In what follows, we define an algorithm that computes prime implicates for sets of flat equational clauses.

It turns out that $\mathcal{SP}_{\mathcal{A}}$ cannot be used to determine the set T'. For instance, if $S' = \{a \simeq b, c \not\simeq d\}$, then the clause $a \not\simeq c \vee b \not\simeq d$ must be in $\mathrm{I}_{\mathcal{A}}(S')$. Since it is subsumed by no clause in $\mathrm{I}_{\mathcal{A}}(S')$ but itself, it must also be in T', but no $\mathcal{SP}_{\mathcal{A}}$-inference rule (or $\mathcal{SP}$-inference rule for that matter) can be applied to S' to generate such a clause. In the sequel, we will show how, starting with a set of $\mathcal{A}$-clauses that logically entails $\mathrm{I}_{\mathcal{A}}(S')$, it is possible to generate a set T' using the *Resolution calculus*, denoted by $\mathcal{R}$ (we refer the reader to [11] for details on the Resolution calculus). From now on, S' denotes a satisfiable set of standard clauses, and S is a set of abstracted clauses such that $S_\nu = S'$. Thus, S and S' are equivalent. The first step towards this construction is the definition of a set of $\mathcal{A}$-clauses that logically entails $\mathrm{I}_{\mathcal{A}}(S')$. The (finite) set of all $\mathcal{A}$-clauses in the saturated set generated from S using $\mathcal{SP}_{\mathcal{A}}$ will satisfy this requirement.

Definition 17. *We denote by T_∞ the set of $\mathcal{A}$-clauses in the $\mathcal{A}$-saturated set generated from S by $\mathcal{SP}_{\mathcal{A}}$.*

The key result that makes the generation of $\mathcal{A}$-implicates possible is that all the $\mathcal{A}$-clauses that are entailed by S are actually logical consequences of T_∞:

Proposition 2. $T_\infty \models \mathrm{I}_{\mathcal{A}}(S')$.

Let Eq be the set of axioms stating that $\simeq$ is an equivalence relation[3]: $Eq = \{x \simeq x,\ x \not\simeq y \vee y \simeq x,\ x \not\simeq y \vee y \not\simeq z \vee x \simeq z\}$, and let $Eq_{\mathcal{A}}$ be the set consisting of all instantiations of the axioms in Eq by the elements in $\mathcal{A}$. The result we

[3] There will be no need to consider the congruence axiom, since all the clauses in T_∞ only contain constants.

$\textsc{Explain}(S', \mathcal{A}) =$
$\quad S := \textsc{Abstract}(S')$
$\quad S := \mathcal{SP}_{\mathcal{A}}\text{-saturation}(S)$
$\quad T_\infty := \{C \in S \mid C \text{ is an } \mathcal{A}\text{-clause}\}$
$\quad$ **return** $\mathcal{R}\text{-saturation}(T_\infty \cup Eq_{\mathcal{A}})$

Fig. 3. Generation of a set of explanations

show is that the $\mathcal{R}$-closure of the set $T_\infty \cup Eq_{\mathcal{A}}$ satisfies the requirements for the set of $\mathcal{A}$-clauses that is searched for.

Theorem 3. *Let $T = T_\infty \cup Eq_{\mathcal{A}}$, and let C be a non-tautological ground clause in $\mathrm{I}_{\mathcal{A}}(S)$. Then there is a derivation from T of a clause C' such that $C' \models C$.*

To summarize, given a set of clauses S' that is satisfiable and a set of abducible constants $\mathcal{A}$, the simple algorithm in pseudo-code described in Figure 3 returns a set of clauses constructed over $\mathcal{A}$ that can be viewed as explanations why S' is satisfiable. Note that $\mathcal{R}$-saturation can be performed on the fly: it is clear that it is not necessary to wait until $\mathcal{SP}_A$-saturation(S) is computed to start generating the clauses in $\mathcal{R}$-saturation$(T_\infty \cup Eq_{\mathcal{A}})$. Thus even in case of non-termination, all the prime implicates can eventually be generated. After the set $\mathcal{R}$-saturation$(T_\infty \cup Eq_{\mathcal{A}})$ is computed, it is possible to remove from this set all the clauses that can be inferred from other prime implicates. This solution yields a more compact representation. However, this is possible only in case of termination, since the deleted clauses may be involved in the generation of other prime implicates. A termination result for $\mathcal{SP}_{\mathcal{A}}$ will be presented in the following section. By putting all the previous results together, we obtain the following theorem, stating the soundness and completeness of the procedure Explain.

Theorem 4. *Let S be a set of clauses. Every clause $C \in \textsc{Explain}(S', \mathcal{A})$ is an $\mathcal{A}$-implicate of S, and for every $\mathcal{A}$-implicate C of S that is not a tautology, there exists a clause $C' \in \textsc{Explain}(S', \mathcal{A})$ such that $C' \models C$.*

Example 3. We return to the problem mentioned in the Introduction. After flattening, we get the following set of clauses:

1	$\mathrm{select}(\mathrm{store}(x, z, v), z) \simeq v$	4	$d_2 \simeq \mathrm{store}(d_1, j, c)$
2	$z \simeq w \vee \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w)$	5	$d_3 \simeq \mathrm{store}(a, j, c)$
3	$d_1 \simeq \mathrm{store}(a, i, b)$	6	$d_4 \simeq \mathrm{store}(d_3, i, b)$
7	$\mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_4, k)$		

Assume that $\mathcal{A} = \{i, j, b, c\}$. Then Clauses $3, 4, 5, 6$ are abstracted as follows:

$3'$	$x' \not\simeq i \vee y' \not\simeq b \vee d_1 \simeq \mathrm{store}(a, x', y')$
$4'$	$x'' \not\simeq j \vee y'' \not\simeq c \vee d_2 \simeq \mathrm{store}(d_1, x'', y'')$
$5'$	$x'' \not\simeq j \vee y'' \not\simeq c \vee d_3 \simeq \mathrm{store}(a, x'', y'')$
$6'$	$x' \not\simeq i \vee y' \not\simeq b \vee d_4 \simeq \mathrm{store}(d_3, x', y')$

$\mathcal{SP}_{\mathcal{A}}$ generates the following clauses[4]:

[4] For readability we simply drop irrelevant disequations, i.e. $x \not\simeq a \vee C$ is replaced by C if x does not occur in C and $x \not\simeq a \vee x' \not\simeq a \vee C$ is replaced by $x \not\simeq a \vee C\{x' \mapsto x\}$.

8	$x' \not\simeq i \vee w \simeq x' \vee \mathrm{select}(d_1, w) \simeq \mathrm{select}(a, w)$	(3′,2)
9	$x'' \not\simeq j \vee w \simeq x'' \vee \mathrm{select}(d_2, w) \simeq \mathrm{select}(d_1, w)$	(4′,2)
10	$x'' \not\simeq j \vee w \simeq x'' \vee \mathrm{select}(d_3, w) \simeq \mathrm{select}(a, w)$	(5′,2)
11	$x' \not\simeq i \vee w \simeq x' \vee \mathrm{select}(d_4, w) \simeq \mathrm{select}(d_3, w)$	(6′,2)
12	$x' \not\simeq i \vee y' \not\simeq b \vee \mathrm{select}(d_1, x') \simeq y'$	(3′,1)
13	$x'' \not\simeq j \vee y'' \not\simeq c \vee \mathrm{select}(d_2, x'') \simeq y''$	(4′,1)
14	$x'' \not\simeq j \vee y'' \not\simeq c \vee \mathrm{select}(d_3, x'') \simeq y''$	(5′,1)
16	$x' \not\simeq i \vee y' \not\simeq b \vee \mathrm{select}(d_4, x') \simeq y'$	(6′,1)
17	$x' \not\simeq i \vee k \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_3, k)$	(11, 7)
18	$x' \not\simeq i \vee k \simeq x' \vee x'' \not\simeq j \vee k \simeq x'' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(a, k)$	(10, 17)
19	$x' \not\simeq i \vee k \simeq x' \vee x'' \not\simeq j \vee k \simeq x'' \vee \mathrm{select}(d_1, k) \not\simeq \mathrm{select}(a, k)$	(9, 18)
20	$x' \not\simeq i \vee x'' \not\simeq j \vee k \simeq x' \vee k \simeq x''$	(8,19)
21	$x' \not\simeq i \vee x'' \not\simeq j \vee k \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_4, x'')$	(20,7)
22	$x' \not\simeq i \vee x'' \not\simeq j \vee k \simeq x' \vee x'' \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_3, x'')$	(11,21)
23	$x' \not\simeq i \vee x'' \not\simeq j \vee y'' \not\simeq c \vee k \simeq x' \vee x'' \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq y''$	(14,22)
24	$x' \not\simeq i \vee x'' \not\simeq j \vee y'' \not\simeq c \vee k \simeq x' \vee x'' \simeq x' \vee \mathrm{select}(d_2, x'') \not\simeq y''$	(20,23)
25	$x' \not\simeq i \vee x'' \not\simeq j \vee k \simeq x' \vee x'' \simeq x'$	(13,24)
26	$x' \not\simeq i \vee x'' \not\simeq j \vee x'' \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_4, x')$	(25,7)
27	$x' \not\simeq i \vee x'' \not\simeq j \vee y' \not\simeq b \vee x'' \simeq x' \vee \mathrm{select}(d_2, k) \not\simeq y'$	(16,26)
28	$x' \not\simeq i \vee x'' \not\simeq j \vee y' \not\simeq b \vee x'' \simeq x' \vee \mathrm{select}(d_2, x') \not\simeq y'$	(25,27)
29	$x' \not\simeq i \vee x'' \not\simeq j \vee y' \not\simeq b \vee x'' \simeq x' \vee \mathrm{select}(d_1, x') \not\simeq y'$	(9,28)
30	$i \simeq j$	(12,29)
31	$x' \not\simeq i \vee x'' \not\simeq j \vee x' \not\simeq x'' \vee k \simeq x'$	(20)
33	$x' \not\simeq i \vee x'' \not\simeq j \vee x' \not\simeq x'' \vee \mathrm{select}(d_2, k) \not\simeq \mathrm{select}(d_4, x')$	(31,7)
34	$x' \not\simeq i \vee x'' \not\simeq j \vee x' \not\simeq x'' \vee y' \not\simeq b \vee \mathrm{select}(d_2, k) \not\simeq y'$	(16,34)
35	$x' \not\simeq i \vee x'' \not\simeq j \vee x' \not\simeq x'' \vee y' \not\simeq b \vee \mathrm{select}(d_2, x') \not\simeq y'$	(31,34)
36	$i \not\simeq j \vee b \not\simeq c$	(13,35)

By Resolution, from 30 and 36, we get $c \not\simeq b$, which subsumes 36. We obtain the A-implicates $\{i \simeq j, b \not\simeq c\}$, yielding the explanation $i \not\simeq j \vee b \simeq c$.

6 A termination result for $\mathcal{SP}_{\mathcal{A}}$

We now prove a result that relates the termination of $\mathcal{SP}$ on a set of standard clauses S to the termination of $\mathcal{SP}_{\mathcal{A}}$ on an abstracted version of S. This shows that many existing results about the termination of the superposition calculus for subclasses of first-order logic carry over to $\mathcal{SP}_{\mathcal{A}}$. We relate standard and abstracted terms by defining a so-called relation of $\mathcal{A}$-relaxation. This relation will be used afterwards to relate the forms of the clauses generated by $\mathcal{SP}$-inferences and those generated by $\mathcal{SP}_{\mathcal{A}}$-inferences in a more precise manner.

Definition 18. *The relation of* $\mathcal{A}$-relaxation *relates an abstracted term t to a standard one t' and is defined as follows: $t \trianglelefteq_{\mathcal{A}} t'$ if and only if $t\gamma_0 = t'_{\downarrow\mathcal{A}}$.*

Given an abstracted clause C and a standard clause C', we write $C \trianglelefteq_{\mathcal{A}} C'$ if and only if $\overline{\Delta}(C\gamma_0) = \overline{\Delta}(C'_{\downarrow\mathcal{A}})$. This relation is extended to sets of clauses in a straightforward manner.

Example 4. Assume $\mathcal{A} = \{a, b\}$, let $C = x \not\simeq a \vee a \simeq b \vee f(x, x, d) \simeq g(y) \vee g(y) \simeq d$ and $C' = a \not\simeq b \vee f(a, b, d) \simeq g(b) \vee g(a) \simeq d$. Then $C \trianglelefteq_{\mathcal{A}} C'$.

We define a notion of redundancy that is meant to hold no matter what abducible constants occur in the clause under consideration.

Definition 19. *An $\mathcal{A}$-reduced clause C' is* P-redundant *in an $\mathcal{A}$-reduced set of clauses S' if for all sets of abstracted clauses S such that $(S_\nu)_{\downarrow\mathcal{A}} \equiv S'$ and for every abstracted clause D such that $(D\nu_D)_{\downarrow\mathcal{A}} \equiv C'$, clause D is $\mathcal{A}$-redundant in S. An $\mathcal{A}$-reduced set of clauses S' is* P-saturated *if every clause generated with premises in S' either occurs in S' or is* P*-redundant in S'.*

This notion permits to eliminate clauses that are redundant in the usual sense and do not contain any abducible constant. Notice, however, that P-redundant clauses can possibly contain abducible constants. For example if $\mathcal{A} = \{a, b\}$ and $S' = \{f(c) \not\simeq f(d)\}$, then $C' = g(a, c) \simeq h(a) \vee f(c) \not\simeq f(d)$ is P-redundant in S'.

Theorem 5. *Let S' be a set of $\mathcal{A}$-reduced clauses, and let T be the* P*-saturated set of clauses generated from S'. If T is finite and S is a set of abstracted clauses that is $\mathcal{V}_\mathcal{A}$-stable, variable-inactive and such that $S \trianglelefteq_\mathcal{A} S'$, then the set of non-redundant clauses generated from S is finite.*

Theorem 5 guarantees that $\mathcal{SP}_\mathcal{A}$ (and thus Explain) terminates on several classes of clause sets, in particular for clause sets related to SMT problems. The authors of [2] and [1] prove that sets of the form $\mathcal{T} \cup U$, where $\mathcal{T}$ is a theory and U a set of ground unit clauses, generate finite saturated sets. This result is extended to clause sets of the form $\mathcal{T} \cup U'$, where U' is an arbitrary set of ground clauses, in [6]. An inspection of the finiteness results of [2, 1, 6] shows that they hold not only for saturated sets but also for P-saturated sets, since the redundant clauses that are deleted are actually P-redundant: they do not contain any constant at all. Thus, $\mathcal{SP}_\mathcal{A}$ terminates for clause sets of the form $\mathcal{T} \cup U'$, where U' is the abstraction of a set of ground clauses, and $\mathcal{T}$ is the axiomatization of any of the following theories: records, integer offsets, possibly empty lists, arrays...

7 Discussion

We have presented a calculus that permits to reason on the relations involving abducible constants, that are logical consequences of a satisfiable set of clauses. These relations can be viewed as explanations of why the set is satisfiable, since any of their negations, when added to the original clause set, renders the latter unsatisfiable. We proved a completeness result for the calculus, along with a sufficient condition guaranteeing its termination on classes of clause sets, among which SMT problems in several theories of interest. To the best of our knowledge, this approach is novel and there are many interesting directions to explore. One first direction is to investigate what set of clauses can be considered as a *good* set of explanations, and determine what a good trade-off might be between a small set of explanations that may hide too many details, and a large set of explanations that may carry too much unnecessary information. Another line of work that is currently under investigation is the search for a more efficient way to generate explanations. Indeed, the saturation with the Resolution calculus in the presence of the equality axioms is not entirely satisfactory as far as efficiency is concerned, and it would be interesting to see how the calculus $\mathcal{SP}_\mathcal{A}$ can be

enhanced to directly produce the required set of explanations. As far as other extensions are concerned, we plan to investigate how to extend these results to *abducible terms* and not only abducible constants, by allowing the occurrence of function symbols in $\mathcal{A}$. This would allow the derivation of non-ground explanations. Another possibility is to consider mixed literals, containing both abducible and non-abducible symbols. It would then be possible to generate explanations of the form $a \simeq 0$ without having to declare 0 as an abducible constant. We also plan on devising a calculus capable of efficiently generating explanations with abducibles interpreted in a particular theory, such as, e.g., arithmetic.

References

1. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 10(1):129–179, January 2009.
2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Computational Logic and Proof Theory, KGC 93*, pages 83–96. Springer, LNCS 713, 1993.
5. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierachic first-order theories. *Appl. Algebra Eng. Commun. Comput.*, 5:193–212, 1994.
6. M. P. Bonacina and M. Echenim. On variable-inactivity and polynomial T-satisfiability procedures. *Journal of Logic and Computation*, 18(1):77–96, 2008.
7. M. P. Bonacina and M. Echenim. Theory decision by decomposition. *Journal of Symbolic Computation*, 45(2):229–260, 2010.
8. M. Echenim and N. Peltier. A calculus for generating ground explanations (technical report). *CoRR*, abs/1201.5954, 2012. URL: `http://arxiv.org/1201.5954`.
9. T. Eiter and G. Gottlob. The complexity of logic-based abduction. *J. ACM*, 42(1):3–42, 1995.
10. P. Jackson. Computing prime implicates. In *ACM Conference on Computer Science*, pages 65–72, 1992.
11. A. Leitsch. *The resolution calculus*. Springer. Texts in Theoretical Computer Science, 1997.
12. P. Marquis. Extending abduction from propositional to first-order logic. In P. Jorrand and J. Kelemen, editors, *Fundamentals of Artificial Intelligence Research*, volume 535 of *LNCS*, pages 141–155. Springer Berlin, 1991.
13. J. McCarthy. Computer programs for checking mathematical proofs. In *Recursive Function Theory*, pages 219–228, Providence, Rhode Island, 1962. Proc. of Symposia in Pure Mathematics, Volume 5, American Mathematical Society.
14. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.
15. D.-K. Tran, C. Ringeissen, S. Ranise, and H. Kirchner. Combination of convex theories: Modularity, deduction completeness, and explanation. *J. Symb. Comput.*, 45(2):261–286, 2010.
16. Yices. `http://yices.csl.sri.com`.