Automatic Decidability for Theories Modulo Integer Offsets

Elena Tushkanova, Christophe Ringeissen, Alain Giorgetti, Olga Kouchnarenko

Automatic Decidability for Theories Modulo Integer Offsets

Elena Tushkanova, Christophe Ringeissen, Alain Giorgetti, Olga Kouchnarenko

RESEARCH REPORT
N° 8139
November 2012
Project-Team Cassis

ISSN 0249-6399   ISRN INRIA/RR--8139--FR+ENG

Automatic Decidability for Theories Modulo Integer Offsets

Elena Tushkanova*, Christophe Ringeissen†, Alain Giorgetti*,
Olga Kouchnarenko*

Project-Team Cassis

Research Report n° 8139 — November 2012 — 20 pages

Abstract: Many verification problems can be reduced to a satisfiability problem modulo theories. For building satisfiability procedures the rewriting-based approach uses a general calculus for equational reasoning named superposition. Schematic superposition, in turn, provides a mean to reason on the derivations computed by superposition. Until now, schematic superposition was only studied for standard superposition. We present a schematic superposition calculus modulo a fragment of arithmetics, namely the theory of Integer Offsets. This new schematic calculus is used to prove the decidability of the satisfiability problem for some theories extending Integer Offsets. We illustrate our theoretical contribution on theories representing extensions of classical data structures, e.g., lists and records. An implementation in the rewriting-based Maude system constitutes a practical contribution. It enables automatic decidability proofs for theories of practical use.

Key-words: Automated deduction, equational reasoning, superposition calculus, decision procedures

* E-mail: `firstName.lastName@femto-st.fr`
† E-mail: `FirstName.LastName@loria.fr`

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Décidabilité automatique pour les théories modulo de l'arithmétique de comptage

Résumé : De nombreux problèmes de vérification peuvent être réduits à un problème de satisfaisabilité modulo une théorie équationnelle. Pour concevoir des procédures de décision de cette satisfaisabilité, l'approche dite "par réécriture" utilise un calcul général pour le raisonnement équationnel, nommé superposition. Une version abstraite de ce calcul, appelée "superposition schématique", fournit un moyen de raisonner sur les dérivations calculées par superposition. Jusqu'à présent, la superposition schématique n'a été définie que pour la superposition standard. Nous présentons ici un calcul de superposition schématique adapté à la superposition modulo l'arithmétique de comptage. Ce nouveau calcul schématique est utilisé pour démontrer automatiquement la décidabilité du problème de satisfaisabilité pour certaines théories qui étendent l'arithmétique de comptage. Nous illustrons cette contribution théorique avec des théories qui axiomatisent des extensions de structures de données classiques, comme les listes et les enregistrements. Une contribution pratique est une implantation de ces calculs dans le système Maude. Cet outil prouve automatiquement la décidabilité de certaines théories usuelles.

Mots-clés : Déduction automatique, raisonnement équationnel, calcul de superposition, procédures de décision

1 Introduction

Decision procedures for satisfiability modulo background theories of classical datatypes such as lists, arrays and records are at the core of many state-of-the-art verification tools. Designing and implementing these satisfiability procedures remains a very hard task. To help the researcher with this time-consuming task, an important approach based on rewriting has been investigated in the last decade [2, 1].

The rewriting-based approach allows building satisfiability procedures in a flexible way by using a general calculus for automated deduction, namely superposition calculus [13]. The superposition calculus is a refutation-complete inference system at the core of all equational theorem provers. In general this calculus provides a semi-decision procedure that halts on unsatisfiable inputs by generating an empty clause, but may not terminate on satisfiable ones. However, it also terminates on satisfiable inputs for some theories axiomatising standard datatypes such as arrays, lists, etc, and thus provides a decision procedure for these theories. A classical termination proof consists in considering the finitely many cases of inputs made of the (finitely many) axioms and any set of ground flat literals.

This proof can be done by hand, by analysing the finitely many forms of clauses generated by saturation, but the process is tedious and error-prone. To simplify this process, a schematic superposition calculus has been developed [9] to build the schematic form of the saturations. It can be seen as an abstraction of superposition calculus: If it halts on one given abstract input, then the superposition calculus halts for all the corresponding concrete inputs. More generally, schematic superposition is an fundamental tool to check important properties related to decidability and combinability [10].

To ensure efficiency, it is very useful to have built-in axioms in the calculus, and so to design superposition calculi modulo theories. This is particularly important for arithmetic fragments due to the ubiquity of arithmetics in applications of formal methods. For instance, superposition calculi have been developed for Abelian Groups [8, 11] and Integer Offsets [12]. These calculi provide decision procedures for theories of practical interest. New combination methods à la Nelson-Oppen have been developed to consider unions of these theories sharing fragments of arithmetics. This paves the way of using non-disjoint combination methods within SMT solvers. In [12], the termination of superposition modulo Integer Offsets is proved manually. Therefore, there is an obvious need for a method to automatically prove that an input theory admits a decision procedure based on superposition modulo Integer Offsets.

In this paper, we introduce theoretical underpinnings and a tool support that allow us to automatically prove the termination of superposition modulo Integer Offsets. To this aim, we design a new schematic superposition calculus to describe saturations modulo Integer Offsets. Our approach requires a new form of schematization to cope with arithmetic expressions. The interest of schematic superposition relies on a correspondence between a derivation using (concrete) superposition and a derivation using schematic superposition: Roughly speaking, the set of derivations obtained by schematic superposition over-approximates the set of derivations obtained by (concrete) superposition. We show under which conditions the termination of schematic superposition implies the termination of (concrete) superposition. Again, the fact of considering Integer Offsets requires some specific proof arguments. We illustrate our contribution on the examples of theories considered in [12] – the theory of lists with length and the theory of records with increment. Our approach has been developed and validated thanks to a proof system we have implemented in the rewriting logic-based environment Maude [5] by using its reflection mechanism and its equational reasoning engines. This proof system implements both schematic and concrete superposition modulo Integer Offsets. The proofs related to our examples are obtained via this automatic proof system.

Plan of the paper. Section 2 introduces the preliminary notions related to first-order theories, briefly presents the superposition and schematic superposition calculi, and introduces the theory of Integer Offsets and its extensions. Section 3 describes the new schematic superposition calculus and states conditions under which its termination implies the one of the (concrete) superposition calculus. Section 4 describes our implementation of these calculi. Section 5 describes our experimentations with Integer Offsets extensions. Eventually, Section 6 concludes.

2 Background

We consider many-sorted first-order equational logic. A (many-sorted functional) *signature* Σ is a set of declarations of distinct function symbols and their type. A function declaration is of the form $f : s_1 \times \ldots \times s_n \to s$, where f is a function symbol, $n \geq 0$ is its arity, $s_1, \ldots, s_n$ and s are *sorts* from a finite set of sorts S. The sorts $s_1, \ldots, s_n$ are called the *argument sorts* and s is called the *value sort* of f. Each sort is interpreted by a nonempty domain. The only predicates are equalities on sorts, denoted $=_s$ for each sort $s \in S$, whose type is $s \times s$. We simply denote them $=$ when there is no risk of confusion.

Given a signature Σ, we assume the usual first-order syntactic notions of term, position, literal, clause, formula, substitution as defined, e.g., in [7]. When extended to many sorts, these notions additionnally require that all terms have the appropriate sorts. A *rewrite system* is a set of directed equalities, called *rewrite rules*. It can be applied repeatedly along the direction given by the rules to replace equals by equals in any term. A term is in *normal form* if it cannot be rewritten any further. A rewrite system is *convergent* if its application to any term leads to a unique normal form.

We use the following notations: l, r, u, t are terms, v, w, x, y, z are variables, all other lower case letters are constant or function symbols. Given a term t and a position p, $t_{|p}$ denotes the subterm of t at position p, and $t[l]_p$ denotes the term t in which l appears as the subterm at position p. When the position p is clear from the context, we may simply write $t[l]$. Application of a substitution σ to a term t is written $\sigma(t)$. The notations $C[l]$ and $\sigma(C)$ are also used for any clause C. The *empty* clause, i.e. the clause with no disjunct, corresponding to an unsatisfiable formula, is denoted $\bot$. The clause obtained from a clause C by replacing the terms occurring in C with their normal forms w.r.t. a convergent rewrite system R is denoted $C\downarrow_R$.

Terms, literals and clauses are *ground* whenever no variable appears in them. Given a function symbol f, an *f-rooted* term is a term whose top-symbol is f. A *compound* term is an f-rooted term for a function symbol f of positive arity. The *depth* of a term is defined inductively as follows: $depth(t) = 0$, if t is a constant or a variable, and $depth(f(t_1, \ldots, t_n)) = 1 + max\{depth(t_i) \mid 1 \leq i \leq n\}$. A term is *flat* if its depth is 0 or 1. A *positive literal* is an equality $l = r$ and a *negative literal* is a disequality $l \neq r$. We use the symbol $\bowtie$ to denote either $=$ or $\neq$. The depth of a literal $l \bowtie r$ is defined as follows: $depth(l \bowtie r) = depth(l) + depth(r)$. A positive literal is *flat* if its depth is 0 or 1. A negative literal is *flat* if its depth is 0.

We also assume the usual first-order notions of model, satisfiability, validity, logical consequence. A *first-order theory* (over a finite signature) is a set of first-order formulae with no free variables. When T is a finitely axiomatized theory, $Ax(T)$ denotes the set of axioms of T. In this paper we consider first-order theories *with equality*, for which the equality symbol $=$ is always interpreted as the equality relation. A formula is *satisfiable in a theory* T if it is satisfiable in a model of T. The *satisfiability problem* modulo a theory T amounts to establishing whether any given finite conjunction of literals (or equivalently, any given finite set of literals) is T-satisfiable or not.

We consider inference systems using well-founded orderings on terms (resp. literals) that are total on ground terms (resp. literals). An ordering $<$ on terms is a *simplification ordering* [7] if it is stable ($l < r$ implies $l\sigma < r\sigma$ for every substitution σ), monotonic ($l < r$ implies $t[l]_p < t[r]_p$ for every term t and position p), and has the subterm property (i.e., it contains the subterm ordering: if l is a strict subterm of r, then $l < r$). Simplification orderings are well-founded. A term t is *maximal* in a multiset S of terms if there is no $u \in S$ such that $t < u$, equivalently $t \not< u$ for every $u \in S$. Hence, if $t \not\leq u$, then t and u are different terms and t is maximal in $\{t, u\}$. An ordering on terms is extended to literals by using its multiset extension on literals viewed as multisets of terms. Any positive literal $l = r$ (resp. negative literal $l \neq r$) is viewed as the multiset $\{l, r\}$ (resp. $\{l, l, r, r\}$). Also, a term is *maximal* in a literal whenever it is maximal in the corresponding multiset.

2.1 Superposition Calculus

The *superposition calculus* is the inference system $\mathcal{UPC}$ [14] consisting of the rules in Figs. 1 and 2. The reader is refered to [14] for a detailed explanation of these rules. As in [14] we consider only unitary clauses, i.e. clauses composed of at most one literal.

A fundamental feature of $\mathcal{UPC}$ is the usage of a simplification ordering $<$ to control the application of *Superposition* and *Simplification* rules by orienting equalities. Hence, the *Superposition* rule is applied by using terms that are maximal in their literals with respect to $<$. This ordering is total on ground terms. We use a lexicographic path ordering [7] such that terms of positive depth are greater than constants.

Let us recall the usual definitions of redundancy, saturation, derivation and fairness. A clause C is *redundant* with respect to a set S of clauses if either $C \in S$ or S can be obtained from $S \cup \{C\}$ by a sequence of applications of contraction rules (cf. Fig. 2). An inference is *redundant* with respect to a set S of clauses if its conclusion is redundant with respect to S. A set S of clauses is *saturated* if every inference with a premise in S is redundant with respect to S. A *derivation* is a sequence $S_0, S_1, \ldots, S_i, \ldots$ of sets of clauses where each S_{i+1} is obtained from S_i by applying an inference to add a clause (by expansion rules in Fig. 1) or to delete a clause (by contraction rules in Fig. 2). A derivation is characterized by its *limit*, defined as the set of persistent clauses $\bigcup_{j \geq 0} \bigcap_{i > j} S_i$, that is, the union for each $j \geq 0$ of the set of clauses occurring in all future steps starting from S_j. A derivation $S_0, S_1, ..., S_i, ...$ is *fair* if for every inference with premises in the limit, there is some $j \geq 0$ such that the inference is redundant with respect to S_j. The set of persistent literals obtained by a fair derivation is called the *saturation* of the derivation.

Superposition $\dfrac{l[u'] \bowtie r \qquad u = t}{\sigma(l[t] \bowtie r)}$

if i) $\sigma(u) \not\leq \sigma(t)$, ii) $\sigma(l[u']) \not\leq \sigma(r)$, and iii) u' is not a variable.

Reflection $\dfrac{u' \neq u}{\bot}$

Above, u and u' are unifiable and σ is the most general unifier of u and u'.

Figure 1: Expansion inference rules of $\mathcal{UPC}$

Subsumption $\dfrac{S \cup \{L, L'\}}{S \cup \{L\}}$ **if** $L' = \sigma(L)$.

Simplification $\dfrac{S \cup \{C[l'], l = r\}}{S \cup \{C[\sigma(r)], l = r\}}$

if i) $l' = \sigma(l)$, ii) $\sigma(l) > \sigma(r)$, and iii) $C[l'] > (\sigma(l) = \sigma(r))$.

Deletion $\dfrac{S \cup \{u = u\}}{S}$

Figure 2: Contraction inference rules of $\mathcal{UPC}$

2.2 Schematic Superposition

The *Schematic Superposition Calculus* $\mathcal{SUPC}$ is an abstraction of $\mathcal{UPC}$. Indeed, any concrete saturation computed by $\mathcal{UPC}$ can be viewed as an instance of an abstract saturation computed by $\mathcal{SUPC}$ [10, Theorem 2]. Hence, if $\mathcal{SUPC}$ halts on one given abstract input, then $\mathcal{UPC}$ halts for all the corresponding concrete inputs. More generally, $\mathcal{SUPC}$ is an automated tool to check properties of $\mathcal{UPC}$ such as termination, stable infiniteness and deduction completeness [10].

$\mathcal{SUPC}$ is almost identical to $\mathcal{UPC}$, except that literals are constrained by conjunctions of atomic constraints of the form $const(x)$ where x is a variable. An implementation of *Superposition* and *Schematic Superposition* calculi $\mathcal{UPC}$ and $\mathcal{SUPC}$ is presented in [14].

2.3 Theory of Integer Offsets

The theory of Integer Offsets is axiomatized by the following axioms over the signature $\Sigma_I := \{0 : \textsc{int}, \mathsf{s} : \textsc{int} \to \textsc{int}\}$:

$$\begin{array}{l} \forall x\ \mathsf{s}(x) \neq 0 \\ \forall x, y\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y \\ \forall x\ x \neq \mathsf{s}^n(x) \quad \text{for all} \quad n \geq 1 \end{array}$$

The second axiom specifies that the successor function s is injective. The third axiom is in fact an axiom scheme, which specifies that this function is acyclic.

2.4 Integer Offsets Extensions

A (non-disjoint) *Integer Offsets extension* is a many-sorted theory whose set of sorts contains $\textsc{int}$, whose signature shares symbols with Σ_I, and whose axioms possibly involve the symbols s and 0. Following [12, Section 5], we consider two Integer Offsets extensions: the theory of lists with length whose signature is $\Sigma_{LLI} = \{\mathsf{car} : \textsc{lists} \to \textsc{elem}, \mathsf{cdr} : \textsc{lists} \to \textsc{lists}, \mathsf{cons} : \textsc{elem} \times \textsc{lists} \to \textsc{lists}, \mathsf{s} : \textsc{int} \to \textsc{int}, \mathsf{len} : \textsc{lists} \to \textsc{int}\}$ and whose set of axioms $Ax(LLI)$

consists of

$$\begin{array}{rcl}\mathsf{car}(\mathsf{cons}(X,Y)) & = & X\\ \mathsf{cdr}(\mathsf{cons}(X,Y)) & = & Y\\ \mathsf{len}(\mathsf{cons}(X,Y)) & = & \mathsf{s}(\mathsf{len}(Y))\end{array}$$

and the theory of records with increment whose signature is $\Sigma_{RII} = \bigcup_{i=1}^{3}\{\mathsf{rstore}_i : \text{REC} \times \text{INT} \to \text{REC}, \mathsf{rselect}_i : \text{REC} \to \text{INT}, \mathsf{s} : \text{INT} \to \text{INT}, \mathsf{incr} : \text{REC} \to \text{REC}\}$ whose set of axioms $Ax(RII)$ consists of

$$\begin{array}{rcl}\mathsf{rselect}_i(\mathsf{rstore}_i(X,Y)) & = & Y \text{ for } i \in \{1,2,3\}\\ \mathsf{rselect}_i(\mathsf{rstore}_j(X,Y)) & = & \mathsf{rselect}_i(X,Y) \text{ for } i,j \in \{1,2,3\} \text{ with } i \neq j\\ \mathsf{rselect}_i(\mathsf{incr}(X)) & = & \mathsf{s}(\mathsf{rselect}_i(X)) \text{ for } i \in \{1,2,3\}\end{array}$$

The superposition-based calculus $\mathcal{UPC}_I$ defined in [12] adapts the superposition calculus $\mathcal{UPC}$ to the theory of Integer Offsets, so that it can serve as a basis for the design of decision procedures for Integer Offsets extensions.

Technically, the axioms of the theory of Integer Offsets are directly integrated in the simplification rules of $\mathcal{UPC}_I$.

We are interested in extending the schematic superposition calculus $\mathcal{SUPC}$ – developed for the standard superposition $\mathcal{UPC}$ – to get a schematic calculus for superposition modulo Integer Offsets.

3 Schematic Superposition Calculus for Integer Offsets

This section introduces a new schematic calculus taking into account the axioms of the theory of Integer Offsets within a framework based on schematic superposition [10, 14]. The theory of Integer Offsets allows us to build arithmetic expressions of the form $\mathsf{s}^n(t)$ for $n > 0$. The idea investigated here is to represent all terms of this form in a unique way. To this end, we consider a new operator $\mathsf{s}^+ : \text{INT} \to \text{INT}$ such that $\mathsf{s}^+(t)$ denotes the infinite set of terms $\{s^n(t) \mid n > 0\}$. The rewrite system

$$R\mathsf{s}^+ = \{\ \mathsf{s}^+(\mathsf{s}(x)) \to \mathsf{s}^+(x),\ \mathsf{s}(\mathsf{s}^+(x)) \to \mathsf{s}^+(x),\ \mathsf{s}^+(\mathsf{s}^+(x)) \to \mathsf{s}^+(x)\ \}$$

can be used to simplify terms containing s^+. For each of these rules, one can easily check that the set of terms denoted by the left-hand side is included in the set of terms denoted by the right-hand side.

3.1 Schematic Terms and Clauses

The schematic superposition calculus handles schematic clauses. This calculus takes as input a set of schematic literals, G_0, that represents all possible sets of ground literals given as inputs of the superposition calculus. Let us introduce the notions of schematic clause, instance of schematic clause and schematic input G_0 we use for the schematization of $\mathcal{UPC}_I$. These notions extend the ones used in [10] for the schematization of standard superposition via constrained clauses. An *atomic constraint* is of the form $const(t)$, and it is true iff t is a constant. A *constraint* is a conjunction of atomic constraints which is true if each atomic constraint in the conjunction is true. For sake of brevity, $const(x_1,\ldots,x_n)$ denotes the conjunction $const(x_1) \wedge \cdots \wedge const(x_n)$. A *constrained clause* is of the form $C \parallel \varphi$, where C is a clause and φ is a constraint. A variable x is *constrained* in a constrained clause $C \parallel \varphi$ if $const(x)$ is in φ; otherwise it is *unconstrained.* We say that $\sigma(C)$ is a *constraint instance* of $C \parallel \varphi$ if the domain of σ contains all the constrained variables in $C \parallel \varphi$, the range of σ contains only constants and $\sigma(\varphi)$ is satisfiable.

Definition 1 (Schematic Clause)**.** *A* schematic clause *is a constrained clause built over the signature extended with* s^+*. An* instance of a schematic clause *is a constraint instance where each occurrence of* s^+ *is replaced by some* s^n *with* $n > 0$*.*

For a given theory T with the signature Σ_T, we define G_0 as follows:

$$\begin{aligned} G_0 = \quad & \{\bot, x \bowtie y \parallel \mathit{const}(x,y), u = \mathsf{s}^+(v) \| \mathit{const}(u,v)\} \\ & \cup \bigcup_{f \in \Sigma_T} \{f(x_1,\ldots,x_n) = x_0 \parallel \mathit{const}(x_0,x_1,\ldots,x_n)\} \end{aligned}$$

where u, v are constrained variables of sort INT, and x, y are of the same sort.

Compared to the standard definition of G_0 introduced in [10], our G_0 contains in addition the schematic literal $u = \mathsf{s}^+(v) \| \mathit{const}(u,v)$.

3.2 Schematic Calculus

To design a schematic calculus for Integer Offsets, we re-use the rules of $\mathcal{SUPC}$ – recalled in Figs. 3 and 4 – and complete them with two reduction rules – presented in Fig. 5 – which are simplification rules for Integer Offsets.

Superposition $\dfrac{l[u'] \bowtie r \| \varphi \qquad u = t \| \psi}{\sigma(l[t] \bowtie r \| \varphi \wedge \psi)}$

if i) $\sigma(u) \not\leq \sigma(t)$, ii) $\sigma(l[u']) \not\leq \sigma(r)$, and iii) u' is not an unconstrained variable.

Reflection $\dfrac{u' \neq u \| \psi}{\bot}$ **if** $\sigma(\psi)$ is satisfiable.

Above, u and u' are unifiable and σ is the most general unifier of u and u'.

Figure 3: Schematic expansion rules

Let us denote $\mathcal{SUPC}_I$ the calculus depicted in Figs. 3, 4 and 5. Let us notice that two simplification rules $C1$ and $C2$ described in [12] that represent two remaining axioms of the theory of Integer Offsets do not appear in the Schematic Superposition calculus modulo Integer Offsets. This is due to the fact that these rules produce only the empty clause $\bot$ which is already in the initial set G_0.

We assume that the ordering $>$ used in $\mathcal{SUPC}_I$ is $T_I - \mathit{good}$ [12]: $>$ is a simplification ordering which is total on ground terms, such that 0 is minimal and, for any non s-rooted terms t_1 and t_2, $\mathsf{s}^{n_1}(t_1) > \mathsf{s}^{n_2}(t_2)$ iff either $t_1 > t_2$ or ($t_1 \equiv t_2$ and n_1 is bigger than n_2). In [12] the definition of *derivation* has been adapted to the superposition calculus for Integer Offsets. Similarly, we adapt the standard definition of derivation to the schematic superposition calculus modulo Integer Offsets.

Definition 2. *A derivation with respect to* $\mathcal{SPC}_I$ *is a (finite or infinite) sequence of sets of literals* $S_1, S_2, S_3, \ldots, S_i, \ldots$ *such that, for every i, it holds that:*

1. S_{i+1} *is obtained from* S_i *by adding a literal obtained by the application of one of the rules in Figs. 3, 4 and 5 to some literals in* S_i*;*

Subsumption $$\frac{S \cup \{L\|\psi, L'\|\psi'\}}{S \cup \{L\|\psi\}}$$

if either a) $L \in Ax(T)$, ψ is empty and for some substitution σ, $L' = \sigma(L)$; or b) $L' = \sigma(L)$ and $\psi' = \sigma(\psi)$, where σ is a renaming or a mapping from constrained variables to constrained variables.

Simplification $$\frac{S \cup \{C[l']\|\varphi, l = r\}}{S \cup \{C[\sigma(r)]\|\varphi, l = r\}}$$

if i) $l = r \in Ax(T)$, ii) $l' = \sigma(l)$, iii) $\sigma(l) > \sigma(r)$, and iv) $C[l'] > (\sigma(l) = \sigma(r))$.

Tautology $$\frac{S \cup \{u = u\|\varphi\}}{S}$$

Deletion $$\frac{S \cup \{L\|\varphi\}}{S}$$ **if** φ is unsatisfiable.

Figure 4: Schematic contraction rules

R1 $$\frac{S \cup \{\mathsf{s}(u) = \mathsf{s}(v)\|\varphi\}}{S \cup \{u = v\|\varphi\}}$$

if u and v are ground terms

R2 $$\frac{S \cup \{\mathsf{s}(u) = t\|\varphi, \mathsf{s}(v) = t\|\psi\}}{S \cup \{\mathsf{s}(v) = t\|\psi, u = v\|\psi \wedge \varphi\}}$$

if u, v and t are ground terms, $\mathsf{s}(u) > t$, $\mathsf{s}(v) > t$ and $u > v$

Figure 5: Ground reduction rules

2. *S_{i+1} is obtained from S_i by removing a literal according to one of the rules in Figs. 4 and 5.*

3.2.1 Schematic deletion

Unfortunately, the schematic saturation calculus diverges. To illustrate this point, let us take a look at the theory of lists with length. In fact, the calculus generates a schematic clause $\mathsf{len}(a) = s(\mathsf{len}(b))\|const(a, b)$ which will superpose with a renamed copy of itself, i.e. with $\mathsf{len}(a') = s(\mathsf{len}(b'))$ $\|const(a', b')$ to generate a schematic clause of a new form $\mathsf{len}(a) = s(s(\mathsf{len}(b')))$ $\|const(a, b')$. This process continues to generate deeper and deeper schematic clauses so that the Schematic Saturation will diverge. To cope with this kind of clauses, we add the following *Schematic Deletion* rule in order to delete constrained clauses that are not relevant for simulating inferences of $\mathcal{UPC}_I$. Since a term $\mathsf{s}^+(t)$ represents all the terms $s(t), s(s(t)), \ldots, s^n(t), \ldots,$ the idea is to replace all these terms by $\mathsf{s}^+(t)$ in the clauses containing them.

Schematic Deletion $$\frac{C'\|\varphi \qquad C[\mathsf{s}^+(t)]\|\psi}{C[\mathsf{s}^+(t)]\|\psi}$$

if there is a renaming σ s.t.
$\sigma(\pi(C') \downarrow_{R\mathsf{s}^+}) = C[\mathsf{s}^+(t)]$ and $\sigma(\varphi) = \psi$

where π is a morphism replacing all the occurences of s by s^+ ($\pi(s(t)) = \mathsf{s}^+(t)$ for any t).

This rule removes a schematic literal $C'\|\varphi$ from a set of schematic literals that contains $C[\mathsf{s}^+(t)]\|\psi$ if $C'\|\varphi$ is an instance of $C[\mathsf{s}^+(t)]\|\psi$, modulo some renaming.

The *Schematic Deletion* rule can be applied if and only if the initial set of schemas of ground flat literals G_0 for the theory of lists with length is extended with the non-flat schematic literal $\mathsf{len}(a) = \mathsf{s}^+(\mathsf{len}(b))\|const(a, b)$. Thanks to these two changes, the schematic saturation terminates for the theory of lists with length. Similarly, the schematic saturation of the theory of records with increment diverges. But thanks to the *Schematic Deletion* rule, it terminates if the initial set of schematic literals additionally contains the schematic literal $\mathsf{rselect}_i(a) = \mathsf{s}^+(\mathsf{rselect}_i(b))\|const(a, b)$. More generally, we propose to extend G_0 with the non-flat schematic literal $u = \mathsf{s}^+(v)\|\varphi$ where u and v are two flat terms whose variables are all constrained. Finally, the set of ground schematic literals G_0 is defined as follows:

$$\begin{aligned} G_0 = \ & \{\bot, x \bowtie y \parallel const(x, y), u = \mathsf{s}^+(v)\|const(u, v)\} \\ & \cup \textstyle\bigcup_{f\in\Sigma_T}\{f(x_1, \ldots, x_n) = x_0 \parallel const(x_0, x_1, \ldots, x_n)\} \end{aligned}$$

where u, v are either constrained variables of sort INT or flat terms whose variables are all constrained, and x, y are constrained variables of the same sort.

3.3 Adequation Result

We show that any clause in a saturation obtained by $\mathcal{UPC}_I$ is an instance of a schematic clause in a schematic saturation obtained by $\mathcal{SUPC}_I$.

Assumption 1. *Let SC be any set of schematic clauses generated by $\mathcal{SUPC}_I$. If an s^+-rooted term (resp. s-rooted term) occurs in a term u which is maximal in an equality $u = t$ in SC, then there is no s-rooted term (resp. s^+-rooted term) occurring in a term $l[u']$ which is maximal in a clause $l[u'] \bowtie r$ in SC.*

Theorem 1. *Let T be a theory axiomatized by a finite set $Ax(T)$ of literals, which is saturated with respect to $\mathcal{UPC}_I$. Let G_∞ be the set of all schematic clauses in a saturation of $Ax(T) \cup G_0$ by $\mathcal{SUPC}_I$. Then for every set S of ground flat literals, every clause in a saturation of $Ax(T) \cup S$ by $\mathcal{UPC}_I$ is an instance of a schematic clause in G_∞.*

Proof. The proof is an adaptation of the one of [10, Theorem 2]. The proof is by induction on the length of derivations of $\mathcal{UPC}_I$. The base case is obvious. For the inductive case, we need to show two facts:

(1) each clause added in the process of saturation of $Ax(T) \cup S$ is an instance of a schematic clause in the saturation G_∞ of $Ax(T) \cup G_0$ by $\mathcal{SUPC}_I$, and

(2) if a clause is deleted by Subsumption, Tautology Deletion or Deletion from (or simplified by Simplification/reduced by Reduction in) G_∞, then all instances of the latter will also be deleted from (or simplified/reduced in) the saturation of $Ax(T) \cup S$ by $\mathcal{UPC}_I$.

Moreover, because of additional rewriting rules for terms containing s^+, we have to check another fact:

(3) Any such rule preserves the set of instances of any schematic clause.

Proof of (1). Consider the *Superposition* rule of $\mathcal{UPC}_I$. By induction hypothesis $l[u'] \bowtie r$ and $u = t$ are instances of schematic clauses in G_∞, i.e. there is some schematic clause $\hat{D}$ in G_∞ such that $l[u'] \bowtie r$ is an instance of $\hat{D}$, and a schematic clause $\hat{E}$ in G_∞ such that $u = t$ is an instance of $\hat{E}$. Two cases can be distinguished:

($*$) If there is no occurrence of s in u and u', then there must exist a *Superposition* inference of $\mathcal{SUPC}_I$ in G_∞, whose premises are $\hat{D}$ and $\hat{E}$ with conclusion $\hat{C}$ such that $\sigma(l[t] \bowtie r)$ is an instance of the schematic clause $\hat{C}$, where σ denotes the most general unifier of u and u'.

($**$) If there are occurrences of s in u and u', two additional subcases can be considered. Assume that $\hat{u}$ and $\hat{u'}$ denote the schematic terms of u and u':

1. If $\hat{u}$ and $\hat{u'}$ contain only s^+-rooted terms (resp. s-rooted terms), then we proceed as in ($*$).
2. if $\hat{u}$ contains an s^+-rooted term (resp. s-rooted term) and $\hat{u'}$ contains an s-rooted term (resp. s^+-rooted term), then $\hat{u}$ may not unify with $\hat{u'}$ since we use syntactic unification, while u and u' may unify. This subcase is avoided by Assumption 1 and the side conditions of the *Superposition* rule.

Reflection of $\mathcal{UPC}_I$ can be handled in a way similar to *Superposition* and is therefore omitted.

Proof of (2). Let us consider *Subsumption* of $\mathcal{SUPC}_I$. For the case (a), let assume that there are a schematic clause A deleted from G_∞ and a clause B in the saturation of $Ax(T) \cup S$ by $\mathcal{UPC}_I$, which is an instance of the schematic clause A. Then there must exist a clause $C \in Ax(T)$ and some substitution θ such that $\theta(C) \subseteq A$. Since all the clauses in $Ax(T)$ persist, there must be a substitution θ' such that $\theta'(C) \subseteq B$. Thereby B must also be deleted from the saturation of $Ax(T) \cup S$ by $\mathcal{UPC}_I$, and we are done. The case (b) of *Subsumption* is just a matter of deleting duplicates and leaving only more general constrained literals.

Since axioms do not contain the s^+ symbol, a similar argument can be used for *Simplification* of $\mathcal{SUPC}_I$. Assume that there is a schematic clause $C[l'] \parallel \varphi$ in G_∞ simplified by an equality $l = r$ ($l = r \in Ax(T)$) into $C[\theta(r)] \parallel \varphi$. Let σ be a substitution such that $\sigma(C[l'])$ is an instance of $C[l'] \parallel \varphi$. Since $l = r$ persists in the saturation of $Ax(T) \cup S$ by $\mathcal{UPC}_I$, there must be a simplification of $\sigma(C[l']) = \sigma(C)[\sigma(\theta(l))]$ by $l = r$ into $\sigma(C)[\sigma(\theta(r))] = \sigma(C[\theta(r)])$, which is an instance of $C[\theta(r)] \parallel \varphi$.

For the *Tautology Deletion* rule of $\mathcal{SUPC}_I$, it is easy to see that a constraint instance of a tautology is also a tautology. For the *Deletion* rule of $\mathcal{SUPC}_I$, notice that clauses with an unsatisfiable constraint have no instances.

For the reduction rule $R1$ of $\mathcal{SUPC}_I$, it is easy to see that an instance of a schematic clause $\mathsf{s}(u) = \mathsf{s}(v)$ will also reduce a root symbol s. For the reduction rule $R2$ of $\mathcal{SUPC}_I$, a similar argument can be given.

Proof of (3). The set of (concrete) clauses schematized by a schematic clause C is included in the set of (concrete) clauses schematized by $C \downarrow_{R\mathsf{s}^+}$, because a similar inclusion holds for all the terms in C and all the rules in $R\mathsf{s}^+$. □

3.4 Application to the Analysis of Superposition

Contrary to the standard case, a schematized saturation may represent an infinite set of clauses since the term $\mathsf{s}^+(t)$ represents all the terms $s^n(t)$ with $n \geq 1$. The difficulty is then to prove the termination in this case. In [12], the termination proofs do not only rely on the fact that there are finitely many forms of clauses generated by the superposition calculus. In addition, the following proof argument is used: any new ground literal is strictly smaller than the biggest ground literal in the input set. Similarly, whereas the schematic superposition allows computing the different forms of clauses generated by superposition, we still need an additional analysis to conclude that the superposition calculus terminates. Fortunately, this analysis can be easily performed for some cases. We investigate hereafter a new solution where the analysis is restricted to the (few) schematic equalities containing s^+ that occur in the (finite) schematic saturation.

Assumption 2. *A schematic equality containing* s^+ *cannot be instantiated with different values of the exponent of* s *in a saturation of a satisfiable input.*

For instance, we cannot have both $i = \mathsf{s}(j)$ and $i = \mathsf{s}^2(j)$ in the saturation of a satisfiable input due to the acyclicity axiom.

Theorem 2. *If the schematic superposition calculus does not increase the number of disequalities and generates a finite schematic saturation such that all its schematic equalities satisfy Assumption 2, then the superposition calculus terminates on any input set of ground literals.*

Proof. Consider a satisfiable input. The number of clauses occurring in the saturation can be bounded as follows:

1. By hypothesis, the number of disequalities cannot be greater than the number of disequalities occurring in the input set.
2. Consider the equalities. The instantiation of constrained variables by constants is bounded by the number of constants. According to Assumption 2, a schematic literal containing an s^+-rooted term can be instantiated by only one instance. Consequently, if the schematic saturation is finite, then the number of equalities occurring in the saturation is also bounded.

Consequently, the superposition calculus computes a finite saturated set of clauses and terminates. □

4 Implementation

This section presents an implementation of the schematic superposition calculus modulo Integer Offsets $\mathcal{SUPC}_I$, with the Maude system [6] supporting rewriting logic and membership equational logic. This implementation extends an implementation of $\mathcal{SUPC}$ described in [14]. We reuse the expansion and contraction rules implemented in [14]. The new implementation supports now many-sorted theories. We have implemented a normalization of schematic terms containing s^+ (Section 4.3), an implementation of the two additional reduction rules *R1* and *R2* (Section 4.4) and of the *Schematic Deletion* rule (Section 4.5), and a support for derivation traces (Section 4.6).

4.1 Schematic literals

We take profit of the powerful reflection mechanism of Maude. Maude terms are reflected as "meta-terms" with sort `Term`. The base cases in the metarepresentation of terms are given by the subsorts `Constant` and `Variable` of the sort `Term`.

Most of our implementation works at the meta-level, i.e. its functions operate on meta-terms with sort `Term`.

Literals are defined by

```
sort Literal .
op _equals_ : Term Term -> Literal [comm] .
op _!=_     : Term Term -> Literal [comm] .
```

The attribute `[comm]` declares that the infix binary symbols `equals` and `!=` for equality and disequality are commutative. Then, the sort `SLiteral` of schematic literal is declared by

```
sorts SLiteral .
op emptyClause : -> SLiteral .
op ax : Literal -> SLiteral .
op _ || _ : Literal Constraint -> SLiteral .
```

where the infix operator `||` constructs a constrained literal from a literals and a constraint of sort `Constraint`. A constraint is implemented as a set of atomic constraints of the form `const(`t`)` where t is a term. An atomic constraint is satisfiable iff t is of subsort `Variable`, but unification sometimes produces `const(`t`)` where t is not a variable. Such a constraint is afterwards detected as unsatisfiable.

4.2 Sorts

The underlying logic of Maude is order-sorted, admitting a subsort ordering, whereas the underlying logic of our calculus $\mathcal{SUPC}_I$ is many-sorted, i.e. there is no subsort relation between sorts in the addressed theories. Let Σ be a many-sorted signature and S be its set of sorts. When implementing Σ in Maude, each sort in S is implemented as a Maude sort. For the theory of lists with length, the sorts LISTS, ELEM and INT are implemented by the declaration

```
sorts Lists Elem Ints .
```

in Maude.[1] Moreover, no subsort relation is declared between these Maude sorts. This condition guarantees that the order-sorted features of Maude (pattern-matching, unification, etc) behave as many-sorted ones on the set of Maude sorts associated to S.

4.3 Normalization of Schematic Terms

The rewrite system Rs^+ is convergent, i.e. the repeated application of rules leads to a unique normal form. The `nf` function (`op nf : Term -> Term`) computes this normal form. This function is applied eagerly whenever a new literal is generated. The normalization of terms is extended to literals by the `nfLit` function (`nfLit : Literal -> Literal`) that normalizes both sides of a given literal.

4.4 Ground Reduction Rules

Let us now present the encoding of the reduction rules of $\mathcal{SUPC}_I$. We translate them into rewrite rules.

The *R1* reduction rule is encoded by the following conditional rewrite rule:

[1] An `s` is added to the Maude sort names for integers and lists because Maude sorts named `Int` and `List` already exist.

```
crl [red1] :
 'succ[U] equals 'succ[U'] || Phi => U equals U' || Phi
   if isGround(U, Phi) and isGround(U', Phi) .
```

This rule removes the root symbol in both sides of a literal if this root symbol is `'succ` (`'succ` stands for `s`) and their subterms `U` and `U'` are ground terms, i.e. they have no variables. This condition is checked by the function `isGround` defined by

```
op isGround : Term Constraint -> Bool .
eq isGround(T, Phi) = vars(T) inTL varsOfSC(Phi) .
```

The function checks whether all variables of term `T` are in the list of variables of constraint `Phi`. This inclusion is checked by the `inTL` function.

The following Maude conditional rewrite rule encodes the *R2* reduction rule.

```
crl [red2] :
 'succ[U] equals T || Phi1, 'succ[V] equals T || Phi2 =>
 'succ[V] equals T || Phi2,
 U equals V || cleanConstraint(U, V, Phi1, Phi2)
   if isGround(U, Phi1) and isGround(V, Phi2) and
      isGround(T, Phi1) and gtLPO('succ[U], Phi1, T) and
      gtLPO('succ[V], Phi2, T) and gtLPO(U, (Phi1, Phi2), V) .
```

The ordering $>$ on terms is implemented as a Boolean function `gtLPO` such that `gtLPO`$(u, \mathtt{SC}, t) =$ `true` iff $u > t$ where the constrained variables collected in the additional parameter `SC` are viewed as constants. The function `cleanConstraint` aims at removing the constrained variables that do no occur in $u = v$.

4.5 Schematic Deletion

The *Schematic Deletion* rule is encoded by the following Maude conditional rewrite rule

```
crl [sdel] : L || Phi1, L' || Phi2 => L' || Phi2
  if conditionDel(L || Phi1, L' || Phi2) .
```

It leaves the second schematic unitary clause if the following three conditions checked by the `conditionDel` function are satisfied:

1. one of the terms of the literal `L` contains an `s` function symbol

   This is checked by the `isSuccLit` function

   ```
   op isSuccLit : Literal -> Bool .
   eq isSuccLit(U equals U') = isSucc(U) or isSucc(U') .
   ```

   that invokes the `isSucc` function to determine whether a given term contains `'succ` symbol or not

   ```
   op isSucc : Term -> Bool .
   eq isSucc('succ[T]) = true .
   ceq isSucc(F[TL]) = true if isSucc$(TL) .
   eq isSucc(T) = false [owise] .
   ```

The function `isSucc$` checks the same property in the list of terms.

```
op isSucc$ : TermList -> Bool .
eq isSucc$(empty) = false .
eq isSucc$((T, TL)) = if isSucc(T) then true
  else isSucc$(TL) fi .
```

2. one of the terms of the literal `L'` contains an s^+ function symbol

   This is similarly implemented by the `isSucc+Lit` function

```
op isSucc+Lit : Literal -> Bool .
eq isSucc+Lit(U equals U') = isSucc+(U) or isSucc+(U') .
```

that calls up the boolean function `isSucc+` to check whether a given term contains `'succ+` symbol

```
op isSucc+ : Term -> Bool .
eq isSucc+('succ+[T]) = true .
ceq isSucc+(F[TL]) = true if isSucc+$(TL) .
eq isSucc+(T) = false [owise] .
```

This function invokes the function `isSucc+$` that determines whether the list of terms contains `'succ+`

```
op isSucc+$ : TermList -> Bool .
eq isSucc+$(empty) = false .
eq isSucc+$((T, TL)) = if isSucc+(T) then true
  else isSucc+$(TL) fi .
```

3. after replacing `s` with s^+ in `L` by the `replaceSbyS+` function

```
op replaceSbyS+ : Term -> Term .
eq replaceSbyS+('succ[T]) = 'succ+[replaceSbyS+(T)] .
eq replaceSbyS+('succ+[T]) = 'succ+[replaceSbyS+(T)] .
eq replaceSbyS+(F[T]) = F[replaceSbyS+(T)] .
eq replaceSbyS+(T) = T [owise] .
```

and normalizing the result (by function `nfLit` described in Section 4.3), literals `L` and `L'` are renamings of each other. The renaming property is checked by the `isRename` function that determines if there is a substitution mapping the first literal into the second one and the constraint of the first literal into the constraint of the second one. This substitution should replace variables by variables and the correspondence between the replaced and the replacing variables is one to one.

4.6 Traces

An additional and important feature of our tool consists in providing a trace indicating the name of the applied rule and the schematic clauses it is applied to at each derivation step. This trace helps understanding the origin of each new schematic clause. With this information, we can adapt the schematic calculus and/or its initial set of clauses in order to entail termination of the schematic saturation. The following example of a traced schematic clause generated by our tool is composed of the trace fragment, the word "gives" and the schematic clause obtained on this trace.

$\mathbf{sup}(i_1 = \mathsf{s}^+(i_2) \parallel \mathit{const}(i_1, i_2),\ i_1 = \mathsf{s}^+(i_2) \parallel \mathit{const}(i_1, i_2))$
$\quad\mathbf{gives}\ \mathsf{s}^+(i_1) = \mathsf{s}^+(i_2) \parallel \mathit{const}(i_1, i_2)$

5 Experimentation with Extensions of Integer Offsets

We experiment our implementation on two examples of Integer Offsets extensions introduced in [12], namely the theory of lists with length and the theory of records with increment. Both of them share symbols with the theory of Integer Offsets in a specific way, in axioms of the form $g(f(\ldots, x, \ldots)) = \mathsf{s}(g(x))$, where symbols f and g are not in the signature of the theory of Integer Offsets. We report here the results generated by our implementation for these two theories of practical interest.

5.1 Theory of Lists with Length

The many-sorted signature Σ_{LLI} of the theory of lists with length is the set $\{\mathsf{car} : \textsc{lists} \to \textsc{elem}, \mathsf{cdr} : \textsc{lists} \to \textsc{lists}, \mathsf{cons} : \textsc{elem} \times \textsc{lists} \to \textsc{lists}, \mathsf{len} : \textsc{lists} \to \textsc{int}, \mathsf{s} : \textsc{int} \to \textsc{int}\}$.

This theory is axiomatized by the following set of axioms $Ax(LLI)$:

1. Axioms for lists
   a) $\mathsf{car}(\mathsf{cons}(X, Y)) = X$
   b) $\mathsf{cdr}(\mathsf{cons}(X, Y)) = Y$

2. Axiom for the length
   a) $\mathsf{len}(\mathsf{cons}(X, Y)) = \mathsf{s}(\mathsf{len}(Y))$

where X is a universally quantified variable of sort ELEM and Y is a universally quantified variable of sort LISTS. The set G_0 consists of the empty clause $\perp$ and the following schemas of clauses:

3. Schematic clauses of sort ELEM
   a) $\mathsf{car}(a) = e \parallel \mathit{const}(a, e)$
   b) $e_1 \bowtie e_2 \parallel \mathit{const}(e_1, e_2)$

4. Schematic clauses of sort LISTS
   a) $\mathsf{cons}(e, a) = b \parallel \mathit{const}(e, a, b)$
   b) $\mathsf{cdr}(a) = b \parallel \mathit{const}(a, b)$
   c) $a \bowtie b \parallel \mathit{const}(a, b)$

5. Schematic clauses of sort INT
   a) $\mathsf{len}(a) = \mathsf{s}^+(i) \parallel \mathit{const}(a, i)$
   b) $\mathsf{len}(a) = \mathsf{s}^+(\mathsf{len}(b)) \parallel \mathit{const}(a, b)$
   c) $\mathsf{s}^+(i_1) = i_2 \parallel \mathit{const}(i_1, i_2)$
   d) $i_1 \bowtie i_2 \parallel \mathit{const}(i_1, i_2)$

where e, e_1, e_2 are constrained variables of sort ELEM, a, b are constrained variables of sort LISTS and i, i_1, i_2 are constrained variables of sort INT. We consider an LPO ordering $>$ over the symbols of the signature Σ_{LLI} respecting the following requirement: $\mathsf{cons} > \mathsf{cdr} > \mathsf{car} > c > e > \mathsf{len} > i > \mathsf{s} > \mathsf{s}^+$ for every constant c of sort LISTS, every constant e of sort ELEM and every constant i of sort INT. These precedence requirements guarantee that every compound term of sort LISTS or ELEM is bigger than any constant, and that $>$ is a $T_I - good$ ordering.

Lemma 1. *The saturation of $Ax(LLI) \cup G_0$ by $\mathcal{SUPC}_I$ consists of $Ax(LLI)$, G_0 and the following schematic clauses:*

$$\mathsf{s}^+(i_1) = \mathsf{s}^+(i_2) \quad \| \quad const(i_1, i_2) \tag{1}$$

$$\mathsf{s}^+(i) = \mathsf{s}^+(\mathsf{len}(a)) \quad \| \quad const(i, a) \tag{2}$$

$$\mathsf{len}(a) = \mathsf{len}(b) \quad \| \quad const(a, b) \tag{3}$$

$$\mathsf{s}^+(\mathsf{len}(a)) = \mathsf{s}^+(\mathsf{len}(b)) \quad \| \quad const(a, b) \tag{4}$$

Proof. The four new schematic clauses are generated by applications of the *Superposition* rule between two schematic clauses in the initial set G_0, as follows: *Superposition* between (5.c) and a renamed copy of itself yields the new schematic clause (1). *Superposition* between (5.a) and (5.b) yields the new schematic clause (2). *Superposition* between (5.b) and a renamed copy of itself yields two new schematic clauses (3) and (4).

Let us now consider all the applications of the *Superposition* rule between an axiom in $Ax(LLI)$ and a schematic clause in G_0. *Superposition* between (1.a) (resp. (1.b)) and (4.a) yields a renaming of (3.a) (resp. (4.b)) which is immediately removed by the *Subsumption* rule. *Superposition* between (2.a) and (4.a) yields the new schematic clause

$$\mathsf{len}(a) = \mathsf{s}(\mathsf{len}(b)) \quad \| \quad const(a, b)$$

which is immediately removed by applying the *Schematic Deletion* rule between it and (5.b)).

The set of axioms $Ax(LLI)$ is saturated. Moreover, any other application of the *Superposition* rule between two schematic clauses or between an axiom and a schematic clause yields a schematic clause that is redundant with respect to $G_0 \cup Ax(LLI) \cup \{(1), (2), (3), (4)\}$. Therefore, this set of schematic clauses is saturated. □

From an encoding of $G_0 \cup Ax(LLI)$ our tool generates the schematic saturation given in Lemma 1. Moreover, its trace

$$\begin{array}{lll}
\mathbf{sup}(label(5.c), label(5.c)) & \mathbf{gives} & \mathsf{s}^+(i_1) = \mathsf{s}^+(i_2) \parallel const(i_1, i_2) \\
\mathbf{sup}(label(5.a), label(5.b)) & \mathbf{gives} & \mathsf{s}^+(i) = \mathsf{s}^+(\mathsf{len}(a)) \parallel const(i, a) \\
\mathbf{sup}(label(5.b), label(5.b)) & \mathbf{gives} & \mathsf{len}(a) = \mathsf{len}(b) \parallel const(a, b) \\
\mathbf{sup}(label(5.b), label(5.b)) & \mathbf{gives} & \mathsf{s}^+(\mathsf{len}(a)) = \mathsf{s}^+(\mathsf{len}(b)) \parallel const(a, b)
\end{array}$$

shows that the new schematic clauses are generated as described in the lemma proof.

5.2 Theory of Records with Increment

We consider now the theory of records of length 3 with increment defined by the many-sorted signature $\Sigma_{RII} = \bigcup_{i=1}^{3}\{\mathsf{rstore}_i : \text{REC} \times \text{INT} \to \text{REC}, \mathsf{rselect}_i : \text{REC} \to \text{INT}, \mathsf{incr} : \text{REC} \to \text{REC}, \mathsf{s} : \text{INT} \to \text{INT}\}$ and the following set of axioms $Ax(RII)$:

1. Axioms for records
   a) $\mathsf{rselect}_i(\mathsf{rstore}_i(X, Y)) = Y$ for all $i \in \{1, 2, 3\}$
   b) $\mathsf{rselect}_j(\mathsf{rstore}_i(X, Y)) = \mathsf{rselect}_j(X, Y)$ for all $i, j \in \{1, 2, 3\}$, $i \neq j$
2. Axiom for the increment

a) $\mathsf{rselect}_i(\mathsf{incr}(X)) = \mathsf{s}(\mathsf{rselect}_i(X))$ for all $i \in \{1, 2, 3\}$

where X is a universally quantified variable of sort REC and Y is a universally quantified variable of sort INT. The set G_0 consists of the empty clause $\perp$ and the following schemas of clauses:

3. Schematic clauses of sort REC

   a) $\mathsf{rstore}_i(a, e) = b \parallel const(a, b, e)$

   b) $\mathsf{incr}(a) = b \parallel const(a, b)$

   c) $a \bowtie b \parallel const(a, b)$

4. Schematic clauses of sort INT

   a) $\mathsf{rselect}_i(a) = e \parallel const(a, e)$

   b) $\mathsf{rselect}_i(a) = \mathsf{s}^+(e) \parallel const(a, e)$

   c) $\mathsf{rselect}_i(a) = \mathsf{s}^+(\mathsf{rselect}_i(b)) \parallel const(a, b)$

   d) $e_1 = \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$

   e) $e_1 \bowtie e_2 \parallel const(e_1, e_2)$

where a, b are constrained variables of sort REC, e, e_1, e_2 are constrained variables of sort INT, and $i \in \{1, 2, 3\}$. We consider an LPO ordering $>$ satisfying the following requirements: for all i, j in $\{1, .., n\}$ $\mathsf{incr} > \mathsf{rstore}_i$, $\mathsf{rstore}_i > \mathsf{rselect}_j$, $\mathsf{rselect}_i > c$ for every constant c, and every constant c is such that $c > \mathsf{s}$.

Lemma 2. *The saturation of $G_0 \cup Ax(RII)$ by $\mathcal{SUPC}_I$ consists of G_0, $Ax(RII)$ and the following schematic clauses, where $i \in \{1, 2, 3\}$:*

$$\mathsf{s}^+(e_1) = \mathsf{s}^+(e_2) \quad \parallel \quad const(e_1, e_2) \tag{5}$$

$$e = \mathsf{s}^+(\mathsf{rselect}_i(a)) \quad \parallel \quad const(a, e) \tag{6}$$

$$\mathsf{rselect}_i(a) = \mathsf{rselect}_i(b) \quad \parallel \quad const(a, b) \tag{7}$$

$$\mathsf{s}^+(\mathsf{rselect}_i(a)) = \mathsf{s}^+(\mathsf{rselect}_i(b)) \quad \parallel \quad const(a, b) \tag{8}$$

$$\mathsf{s}^+(e_1) = \mathsf{s}^+(\mathsf{rselect}_i(a)) \quad \parallel \quad const(a, e_1) \tag{9}$$

$$\mathsf{rstore}_i(a, \mathsf{s}^+(e)) = b \quad \parallel \quad const(a, b, e) \tag{10}$$

Proof. The six new schematic clauses are generated by applications of the *Superposition* rule between two schematic clauses in the initial set G_0, as follows: *Superposition* between (4.d) and the renamed copy of itself yields the new schematic clause (5). For $i \in \{1, 2, 3\}$ *Superposition* between (4.a) and (4.c) yields the new schematic clause (6). *Superposition* between (4.c) and its renamed copy yields two new schematic clauses (7) and (8) for $i \in \{1, 2, 3\}$. *Superposition* between (4.c) and (4.b) yields the new schematic clause (9) for $i \in \{1, 2, 3\}$. *Superposition* between (3.a) and (4.d) yields the new schematic clause (10) for $i \in \{1, 2, 3\}$.

Let us now consider all the applications of the *Superposition* rule between an axiom in $Ax(RII)$ and a schematic clause in G_0. For $i \in \{1, 2, 3\}$ *Superposition* between (1.a) and (3.a) yields a renaming of (4.a), which is immediately removed by the *Subsumption* rule. For $i, j \in \{1, 2, 3\}$ with $i \neq j$ *Superposition* between (1.b) and (3.a) yields a renaming of (7), which is immediately removed by the *Subsumption* rule.

The set of axioms $Ax(RII)$ is saturated. Moreover, any other application of *Superposition* rule between an axiom and a schematic clause or between two schematic clauses yields a schematic clause that is redundant with respect to $G_0 \cup Ax(RII) \cup \{(5), (6), (7), (8), (9), (10)\}$. Therefore, this set of schematic clauses is saturated. □

From an encoding of $G_0 \cup Ax(RII)$ our tool generates the schematic saturation given in Lemma 2 and provides the following trace in conformity with the proof of this lemma:

$$\begin{array}{lll}
\mathbf{sup}(label(4.d), label(4.d)) & \mathbf{gives} & \mathsf{s}^+(e_1) = \mathsf{s}^+(e_2) \parallel const(e_1, e_2) \\
\mathbf{sup}(label(4.a), label(4.c)) & \mathbf{gives} & e = \mathsf{s}^+(\mathsf{rselect}_i(a)) \parallel const(a, e) \\
\mathbf{sup}(label(4.c), label(4.c)) & \mathbf{gives} & \mathsf{rselect}_i(a) = \mathsf{rselect}_i(b) \parallel const(a, b) \\
\mathbf{sup}(label(4.c), label(4.c)) & \mathbf{gives} & \\
\quad \mathsf{s}^+(\mathsf{rselect}_i(a)) = \mathsf{s}^+(\mathsf{rselect}_i(b)) \parallel const(a, b) & & \\
\mathbf{sup}(label(4.c), label(4.b)) & \mathbf{gives} & \mathsf{s}^+(e_1) = \mathsf{s}^+(\mathsf{rselect}_i(a))) \parallel const(a, e_1) \\
\mathbf{sup}(label(3.a), label(4.d)) & \mathbf{gives} & \mathsf{rstore}_i(a, \mathsf{s}^+(e)) = b \parallel const(a, b, e)
\end{array}$$

6 Conclusion

This paper has introduced a new schematic calculus integrating the axioms of the Integer Offsets theory into a framework based on schematic superposition. In this context, introducing the s^+ operator together with the rewriting rules for terms containing s^+ fits well with automatic verification needs. Indeed, similar abstractions have been successfully used to verify cryptographic protocols with algebraic properties [4], and to prove properties of Java Bytecode programs [3]. Moreover, like in [3], our schematization can be used for fine-tuning the precision of the analysis.

In the present paper the calculus with a new form of schematization for arithmetic expressions has been used to show the termination of superposition modulo Integer Offsets. Our approach has been developed and validated thanks to a proof system we have implemented in the Maude environment.

This paper is the first extension of the notion of schematic superposition dedicated to a superposition calculus modulo a built-in theory. This study has led to new automatic proof techniques that are different from those performed manually in [12]. The assumptions we use to apply our proof techniques are easy to satisfy for equational theories of practical interest. As future work, we plan to extend this current framework to theories defined by arbitrary clauses. In this direction, we would have to find a less restrictive assumption to guarantee termination, for instance via a criterion involving the simplification ordering on terms extended to clauses.

References

[1] A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Logic*, 10:4:1–4:51, January 2009.

[2] A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140 – 164, 2003.

[3] Y. Boichut, T. Genet, T. P. Jensen, and L. Le Roux. Rewriting approximations for fast prototyping of static analyzers. In F. Baader, editor, *RTA*, volume 4533 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2007.

[4] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Handling algebraic properties in automatic analysis of security protocols. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC*, volume 4281 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2006.

[5] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. Unification and narrowing in Maude 2.4. In *Rewriting Techniques and*

Applications, 20th International Conference, RTA 2009, Brasília, Brazil, Proceedings, pages 380–390, 2009.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In R. Nieuwenhuis, editor, *Int. Conf. RTA 2003*, volume 2706 of *LNCS*, pages 76–87. Springer, 2003.

[7] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Formal Models and Semantics*, volume B, pages 243–320. MIT Press, 1990.

[8] G. Godoy and R. Nieuwenhuis. Superposition with completely built-in abelian groups. *Journal of Symbolic Computation*, 37(1):1–33, 2004.

[9] C. Lynch and B. Morawska. Automatic decidability. In *LICS*, pages 7–16. IEEE Computer Society, 2002.

[10] C. Lynch, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic decidability and combinability. *Inf. Comput.*, 209(7):1026–1047, 2011.

[11] E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of abelian groups. In R. Schmidt, editor, *Int. Conf. CADE'09*, volume 5663 of *LNAI*, pages 51–66. Springer, 2009.

[12] E. Nicolini, C. Ringeissen, and M. Rusinowitch. Satisfiability procedures for combination of theories sharing integer offsets. In S. Kowalewski and A. Philippou, editors, *TACAS*, volume 5505 of *LNCS*, pages 428–442. Springer, 2009.

[13] 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.

[14] E. Tushkanova, A. Giorgetti, C. Ringeissen, and O. Kouchnarenko. A rule-based framework for building superposition-based decision procedures. In F. Durán, editor, *WRLA*, volume 7571 of *LNCS*, pages 164–182. Springer, 2012.

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399