Automatic Decidability: A Schematic Calculus for Theories with Counting Operators

Elena Tushkanova¹,², Christophe Ringeissen¹, Alain Giorgetti¹,², and Olga Kouchnarenko¹,²

1 Inria, Villers-les-Nancy, F-54600, France
Christophe.Ringeissen@loria.fr
2 FEMTO-ST Institute (UMR 6174), University of Franche-Comté, Besançon, F-25030, France
{elena.tushkanova,alain.giorgetti,olga.kouchnarenko}@femto-st.fr

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 paramodulation. Schematic paramodulation, in turn, provides means to reason on the derivations computed by paramodulation. Until now, schematic paramodulation was only studied for standard paramodulation. We present a schematic paramodulation 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 equipped with counting operators. We illustrate our theoretical contribution on theories representing extensions of classical data structures, e.g., lists and records. An implementation within the rewriting-based Maude system constitutes a practical contribution. It enables automatic decidability proofs for theories of practical use.

1998 ACM Subject Classification F.4.1 Mechanical Theorem Proving, I.2.3 Inference Engines

Keywords and phrases decision procedures, superposition, schematic saturation

Digital Object Identifier 10.4230/LIPIcs.RTA.2013.303

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 the paramodulation calculus [15] (also called superposition calculus). The paramodulation 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 paramodulation calculus has been developed [11] to build
the schematic form of the saturations. It can be seen as an abstraction of the paramodulation
calculus: If it halts on one given abstract input, then the paramodulation calculus halts
for all the corresponding concrete inputs. More generally, schematic paramodulation is a
fundamental tool to check important properties related to decidability and combinability [12].

To ensure efficiency, it is very useful to have built-in axioms in the calculus, and so to
design paramodulation calculi modulo theories. This is particularly important for arithmetic
fragments due to the ubiquity of arithmetics in applications of formal methods. For in-
stance, paramodulation calculi have been developed for Abelian Groups [10, 13] and Integer
Offsets [14]. These calculi provide decision procedures for theories of practical interest.

New combination methods à la Nelson-Oppen have been designed for theories sharing
these arithmetic fragments. This paves the way of using non-disjoint combination methods
within SMT solvers. In [14], the termination of paramodulation modulo Integer Offsets is
proved manually. Therefore, there is an obvious need for an automatic method to prove that
an input theory admits a decision procedure based on paramodulation modulo Integer Offsets.

In this paper, we introduce theoretical underpinnings and a tool support that allow us to
automatically prove the termination of paramodulation modulo Integer Offsets. We design
a new schematic paramodulation calculus to describe saturations modulo Integer Offsets.
Our approach requires a new form of schematization to cope with arithmetic expressions.
The interest of schematic paramodulation relies on a correspondence between a derivation
using (concrete) paramodulation and a derivation using schematic paramodulation: Roughly
speaking, the set of derivations obtained by schematic paramodulation over-approximates
the set of derivations obtained by (concrete) paramodulation. We show under which con-
ditions the termination of schematic paramodulation implies the termination of (concrete)
paramodulation. Again, the fact of considering Integer Offsets requires some specific proof
arguments. We illustrate our contribution on the examples of theories considered in [14] –
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 [6] by using its reflection mechanism and its equational
reasoning engines. This proof system implements schematic paramodulation modulo Integer
Offsets. The proofs related to our examples are obtained via this automatic proof system.

Paper outline. Section 2 introduces preliminary notions related to first-order theories, the
paramodulation and schematic paramodulation calculi, and theories with counting operators.
Section 3 describes the new schematic paramodulation calculus and states conditions under
which its termination implies the one of the (concrete) paramodulation calculus. Sections 4
and 5 respectively describe our implementation and experimentations with Integer Offsets
extensions. Eventually, Section 6 concludes.

2 Background

We consider many-sorted first-order equational logic. A (many-sorted functional) signature
$\Sigma$ is a set of declarations of distinct function symbols and their type. A function declaration
is of the form $f : s_1 \times \ldots \times s_n \rightarrow s$, 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 $\Sigma$, we assume the usual first-order syntactic notions of term, position, literal, clause, formula, substitution as defined, e.g., in [8]. When extended to many sorts, these notions additionally 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.

Throughout this document, universally quantified variables are represented by capital letters. Given a term $t$ and a position $p$, $t[p]$ denotes the subterm of $t$ at position $p$, and $tl[l]$ 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 $tl$. Application of a substitution $\sigma$ 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 by $\bot$. 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 variables 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: $\text{depth}(t) = 0$, if $t$ is a constant or a variable, and $\text{depth}(f(t_1, \ldots, t_n)) = 1 + \max\{\text{depth}(t_i) \mid 1 \leq i \leq n\}$. 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 $\otimes$ to denote either $=$ or $\neq$. The depth of a literal $l \otimes r$ is defined as follows: $\text{depth}(l \otimes r) = \text{depth}(l) + \text{depth}(r)$. 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 $< \otimes$ on terms is a simplification ordering [8] if it is stable ($l < r \implies \sigma(l) < \sigma(r)$ for every substitution $\sigma$), monotonic ($l < r \implies tl[l] < tr[r]$ 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 for finite signatures. A term $t$ is maximal in a multiset $S$ of terms if there is no $u \in S$ such that $t < u$. 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 Paramodulation Calculus

As in [16] we consider only unitary clauses, i.e. clauses composed of at most one literal. The Unitary Paramodulation Calculus is the inference system $UPC$ [16] consisting of the rules in Figures 1 and 2, where $\cup$ stands for disjoint union. Expansion rules (Figure 1) aim at generating new (deduced) literals. The Superposition rule uses an equality to perform
a replacement of equal by equal into a literal. The \textit{Reflection} rule generates the empty clause when both sides of a disequality are unifiable. Contraction rules (Figure 2) aim at simplifying the set of literals. Using \textit{Subsumption}, a literal is removed when it is an instance of another one. \textit{Simplification} rewrites a literal into a simpler one by using an equality that can be considered as a rewrite rule. \textit{Deletion} removes trivial equalities. We assume the usual definitions of redundancy, saturation, derivation and fairness as defined, e.g., in [16].

A fundamental feature of UPC is the usage of a simplification ordering \(<\) to control the application of \textit{Superposition} and \textit{Simplification} rules by orienting equalities. Hence, the \textit{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 [8].

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. Figure 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 Figure 1) or to delete a clause (by contraction rules in Figure 2). One can remark that an application of the \textit{Simplification} rule corresponds to two steps in the derivation: the first step adds a new literal, whilst the second one deletes a literal. 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 \(S_0, S_1, \ldots, S_i, \ldots\) 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.

\[
\text{Superposition} \quad \frac{l[u'] \triangleright r}{\sigma(l[u']) \triangleright r} \quad u = t
\]

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

\[
\text{Reflection} \quad \frac{u' \neq u}{\bot}
\]

Above, \(u\) and \(u'\) are unifiable and \(\sigma\) is the most general unifier of \(u\) and \(u'\).

\[\text{Figure 1} \text{ Expansion inference rules of UPC.}\]

\[
\text{Subsumption} \quad \frac{S \cup \{L, L'\}}{S \cup \{L\} \text{ if } L' = \sigma(L)}.
\]

\[
\text{Simplification} \quad \frac{S \cup \{C[l'], l = r\}}{S \cup \{C[\sigma(r)], l = r\} \text{ if } i) \ l' = \sigma(l), \text{ ii) } \sigma(l) > \sigma(r), \text{ and iii) } C[l'] > (\sigma(l) = \sigma(r)).
\]

\[
\text{Deletion} \quad \frac{S \cup \{u = u\}}{S}
\]

\[\text{Figure 2} \text{ Contraction inference rules of UPC.}\]
2.2 Paramodulation Calculus for Integer Offsets

The paramodulation calculus defined in [14] adapts the 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. The theory of Integer Offsets is axiomatized by the following set of axioms:

\[
\begin{align*}
(s0) & \quad \forall X. \quad s(X) \neq 0 \\
(inj) & \quad \forall X, Y. \quad s(X) = s(Y) \Rightarrow X = Y \\
(acy) & \quad \forall X. \quad X \neq s^n(X) \quad \text{for all } n \geq 1
\end{align*}
\]

over the signature \( \Sigma_I := \{ 0 : \mathbb{INT}, s : \mathbb{INT} \rightarrow \mathbb{INT} \} \). 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.

A (non-disjoint) Integer Offsets extension is a many-sorted theory whose set of sorts contains \( \mathbb{INT} \), whose signature shares symbols with \( \Sigma_I \), and whose axioms possibly involve the symbols \( s \) and 0. Following [14, Section 5], we consider two Integer Offsets extensions: the theory of lists with length whose signature is \( \Sigma_{LLI} = \{ \text{car} : \text{LISTS} \rightarrow \text{ELEM}, \text{cdr} : \text{LISTS} \rightarrow \text{LISTS}, \text{cons} : \text{ELEM} \times \text{LISTS} \rightarrow \text{LISTS}, \text{len} : \text{LISTS} \rightarrow \text{INT}, 0 : \rightarrow \text{INT}, s : \text{INT} \rightarrow \text{INT} \} \) and whose set of axioms \( Ax(\Sigma_{LLI}) \) consists of

\[
\begin{align*}
\text{car}(\text{cons}(X,Y)) & = X & \text{cons}(X,Y) \neq \text{nil} \\
\text{cdr}(\text{cons}(X,Y)) & = Y & \text{len}(\text{nil}) = 0 \\
\text{len}(\text{cons}(X,Y)) & = \text{len}(Y)
\end{align*}
\]

where \( X \) is a universally quantified variable of sort \( \text{ELEM} \), and \( Y \) is a universally quantified variable of sort \( \text{LISTS} \), and the theory of records of length 3 (without extensionality) with increment whose signature is \( \Sigma_{RII} = \bigcup_{i=1}^{3} \{ \text{rstore}_i : \text{REC} \times \text{INT} \rightarrow \text{REC}, \text{rselect}_i : \text{REC} \rightarrow \text{INT}, \text{incr} : \text{REC} \rightarrow \text{REC}, s : \text{INT} \rightarrow \text{INT} \} \) and whose set of axioms \( Ax(\Sigma_{RII}) \) consists of

\[
\begin{align*}
\text{rselect}_i(\text{rstore}_i(X,Y)) & = Y & \text{for } i \in \{1,2,3\} \\
\text{rselect}_i(\text{rstore}_j(X,Y)) & = \text{rselect}_i(X) & \text{for } i, j \in \{1,2,3\} \text{ with } i \neq j \\
\text{rselect}_i(\text{incr}(X)) & = s(\text{rselect}_i(X)) & \text{for } i \in \{1,2,3\}
\end{align*}
\]

where \( X \) is a universally quantified variable of sort \( \text{REC} \), and \( Y \) is a universally quantified variable of sort \( \text{INT} \).

\[
\begin{align*}
R1 & \quad \frac{S \cup \{s(u) = s(v)\}}{S \cup \{u = v\}} \quad \text{if } u \text{ and } v \text{ are ground terms.} \\
R2 & \quad \frac{S \cup \{s(u) = t, s(v) = t\}}{S \cup \{s(v) = t, u = v\}} \\
& \quad \text{if } u, v \text{ and } t \text{ are ground terms, } s(u) > t, s(v) > t \text{ and } u > v. \\
C1 & \quad \frac{S \cup \{s(t) = 0\}}{S \cup \{t = 0, \bot\}} \quad \text{if } t \text{ is a ground term.} \\
C2 & \quad \frac{S \cup \{s^n(t) = t\}}{S \cup \{s^n(t) = t, \bot\}} \quad \text{if } t \text{ is a ground term and } n \geq 1.
\end{align*}
\]

\[\text{Figure 3} \quad \text{Ground reduction rules for Integer Offsets.}\]

The paramodulation calculus \( \mathcal{UPC}_I \) consists of the expansion rules of \( \mathcal{UPC} \) (Figure 1), the contraction rules of \( \mathcal{UPC} \) (Figure 2) plus some additional reduction rules corresponding
to the axioms of the theory of Integer Offsets (Figure 3). We use a $T_i = good$ ordering $<$ to control $UPC_i$; $<$ 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 $t_2$, $s^{n_1}(t_1) > s^{n_2}(t_2)$ iff either $t_1 = t_2$ and $n_1$ is bigger than $n_2$, or $t_1 > t_2$. The refutation completeness of $UPC_i$ is studied in [14].

2.3 Schematic Paramodulation

The motivation of schematic paramodulation is to derive a schematic form of saturations computed by paramodulation. In the following, we consider the Schematic Unitary Paramodulation Calculus, denoted by $SUPC$, as an abstraction of $UPC$. Indeed, any concrete saturation computed by $UPC$ can be viewed as an instance of an abstract saturation computed by $SUPC$ [12, Theorem 2]. Hence, if $SUPC$ halts on one given abstract input, then $UPC$ halts for all the corresponding concrete inputs. More generally, $SUPC$ is an automated tool to check properties of $UPC$ such as termination, stable infiniteness and deduction completeness [12].

$SUPC$ is almost identical to $UPC$, except that literals are constrained by conjunctions of atomic constraints of the form $\text{const}(x)$ where $x$ is a variable. An implementation of Paramodulation and Schematic Paramodulation calculi $UPC$ and $SUPC$ is presented in [16].

Let us recall the notions of constrained clause and instance of constrained clause used in [12] for $SUPC$. An atomic constraint is of the form $\text{const}(t)$, where $t$ is a term. 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, $\text{const}(t_1, \ldots, t_n)$ denotes the conjunction $\text{const}(t_1) \land \cdots \land \text{const}(t_n)$. A constrained clause is of the form $C \parallel \varphi$, where $C$ is a clause and $\varphi$ is a constraint. A variable $x$ is constrained in a constrained clause $C \parallel \varphi$ if $\text{const}(x)$ is in $\varphi$; otherwise it is unconstrained. We say that $\sigma(C)$ is a constraint instance of $C \parallel \varphi$ if the domain of $\sigma$ contains all the constrained variables in $C \parallel \varphi$, the range of $\sigma$ contains only constants and $\sigma(\varphi)$ is satisfiable. By a slight abuse of notation, we say that a term $u$ is ground with respect to a constraint $\varphi$ if all the variables in $u$ are constrained (are in $\varphi$).

3 Schematic Paramodulation Calculus for Integer Offsets

This section introduces a new schematic calculus denoted by $SUPC_i$. It is a schematization of $UPC_i$ taking into account the axioms of the theory of Integer Offsets within the framework based on schematic paramodulation introduced in Section 2.3.

The theory of Integer Offsets allows us to build arithmetic expressions of the form $s^n(t)$ 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 $s^+: \text{INT} \to \text{INT}$ such that $s^+(t)$ denotes the infinite set of terms $\{s^n(t) \mid n \geq 1\}$. A schematic term is a term containing $s^+$.

The calculus $SUPC_i$ handles schematic clauses that extend constrained clauses of $SUPC$.

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 $n > 0$.

The calculus $SUPC_i$ takes as input a set of schematic literals, $G_0$, that represents all possible sets of ground literals given as inputs to $UPC_i$:

\[
G_0 = \{ \bot, x = y \parallel \text{const}(x, y), x \neq y \parallel \text{const}(x, y), u = s^+(v) \parallel \varphi \} \\
\cup \bigcup_{f \in \Sigma_x} \{ f(x_1, \ldots, x_n) = x_0 \parallel \text{const}(x_0, x_1, \ldots, x_n) \}
\]

where $u, v$ are flat terms of sort $\text{INT}$ whose variables are all constrained, and $x, y$ are constrained variables of the same sort.
3.1 Schematic Calculus

The calculus SUPC₁ is depicted in Figures 4, 5 and 6. It re-uses most of the rules of SUPC (Figures 4 and 5) and completes them with two rules (Figure 6) coming from the reduction rules for Integer Offsets (Figure 3).

In [14] the definition of derivation has been adapted to the paramodulation calculus for Integer Offsets. Similarly, we adapt the standard definition of derivation to the schematic paramodulation calculus modulo Integer Offsets.

- Definition 2. A derivation with respect to SUPC₁ 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 Figures 4, 5 and 6 to some literals in $S_i$;
2. $S_{i+1}$ is obtained from $S_i$ by removing a literal according to one of the rules in Figures 5 and 6.

\[
\text{Superposition} \quad \frac{l[u'] \Rightarrow r \mid \varphi}{\sigma(l[l] \Rightarrow r \mid \varphi \land \psi)} \quad \text{if } \begin{cases} i) \sigma(u) \not\leq \sigma(t), & \text{ii) } \sigma(l[l']) \not\leq \sigma(r), \text{ and } \iii) \text{ } u' \text{ is not an unconstrained variable.} \end{cases}
\]

\[
\text{Reflection} \quad \frac{u' \not= u \mid \psi}{\perp} \quad \text{if } \sigma(\psi) \text{ is satisfiable.}
\]

Above, $u$ and $u'$ are unifiable and $\sigma$ is the most general unifier of $u$ and $u'$.

Figure 4 Schematic expansion rules.

We use a specific term rewrite system to simplify schematic terms. Hence, the rewrite system $Rs^+ = \{ s^+(s(x)) \rightarrow s^+(x), s(s^+(x)) \rightarrow s^+(x), s^+(s^+(x)) \rightarrow s^+(x) \}$ is applied eagerly whenever a new literal is generated by superposition or simplification. 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.

Let us notice that the two rules $C1$ and $C2$ from UPC₁ (Figure 3) do not appear in SUPC₁. This is due to the fact that these rules produce only the empty clause $\perp$ which occurs already in the input set $G_0$. Note that the Reflection rule could be omitted as well, but it is kept because it cannot be removed in the non-unitary case.

Similarly to [12], we introduce a specific Schematic Deletion rule to avoid the divergence of SUPC₁. Let us motivate this rule by considering the theory of lists with length. In fact, the calculus generates a schematic clause $\text{len}(a) = s(\text{len}(b))) || \text{const}(a, b)$ which will superpose with a renamed copy of itself, i.e. with $\text{len}(a') = s(\text{len}(b')) || \text{const}(a', b')$ to generate a schematic clause of a new form $\text{len}(a) = s(s(\text{len}(b'))) || \text{const}(a, b')$. This process continues to generate deeper and deeper schematic clauses so that the Schematic Saturation will diverge. To avoid this divergence problem, the additional Schematic Deletion rule aims at checking whether instances of a schematic clause $C'$ are instances of another schematic clause $C$ containing an occurrence of $s^+$. To implement this rule, we apply a morphism $\pi$ replacing all the occurrences of $s$ in $C'$ by $s^+$ ($\pi(s(t)) = s^+(\pi(t))$ for any $t$, $\pi(x) = x$ if $x$ is a variable), and then the rewrite system $Rs^+$ to replace a chain of $s^+$ occurrences in $\pi(C')$ by only one occurrence of $s^+$.
310 Automatic Decidability for Theories with Counting Operators

**Subsumption**

\[
S \cup \{L\parallel \psi, L'\parallel \psi'\} \\
S \cup \{L\parallel \psi\}
\]

If either a) \(L \in \text{Ax}(T)\), \(\psi\) is empty and for some substitution \(\sigma\), \(L' = \sigma(L)\); or b) \(L' = \sigma(L)\) and \(\psi' = \sigma(\psi)\), where \(\sigma\) is a renaming or a mapping from constrained variables to constrained variables.

**Simplification**

\[
S \cup \{C[l']\parallel \varphi, l = r\} \\
S \cup \{C[\sigma(r)]\parallel \varphi, l = r\}
\]

\(\because\) l = r \(\in \text{Ax}(T)\), ii) \(l' = \sigma(l)\), iii) \(\sigma(l) > \sigma(r)\), and iv) \(C[l'] > (\sigma(l) = \sigma(r))\).

**Tautology**

\[
S \cup \{u = u\parallel \varphi\}
\]

**Deletion**

\[
S \cup \{L\parallel \varphi\} \\
S
\]

If \(\varphi\) is unsatisfiable.

**Schematic Del.**

\[
S \cup \{C'\parallel \varphi, C[s^+(t)]\parallel \psi\} \\
S \cup \{C[s^+(t)\parallel \psi]\}
\]

\(\because\) \(\sigma(\pi(C') \downarrow_{R_{\text{ax}}} = C[s^+(t)], \sigma(\varphi) = \psi\), for a renaming \(\sigma\).

**Figure 5** Schematic contraction rules.

**Figure 6** Ground reduction rules.

\(s^+.\) For instance, **Schematic Deletion** rule applies for the theory of lists with length since \(G_0\) already contains the non-flat schematic literal \(\text{len}(a) = s^+ (\text{len}(b)) \parallel \text{const}(a, b)\), and so it subsumes a schematic clause like \(\text{len}(a) = s(s(\text{len}(b))) \parallel \text{const}(a, b')\). Similarly, the schematic saturation of the theory of records with increment diverges. But thanks to the **Schematic Deletion** rule, it terminates since the initial set of schematic literals additionally contains the schematic literal \(\text{rselect}(a) = s^+ (\text{rselect}(b)) \parallel \text{const}(a, b)\). A generalization of this condition to any theory extending Integer Offsets consists in adding to the standard definition of \(G_0\) (introduced in [12]) a finite set of the schematic literals of the form \(u = s^+(v)\parallel \varphi\), where \(u\) and \(v\) are two flat terms of sort \text{INT} whose variables are constrained by \(\varphi\).
3.2 Adequation Result

We show that any clause in a saturation obtained by $UPC_I$ is an instance of a schematic clause in a schematic saturation obtained by $SUPC_I$, under the following assumption.

- **Assumption 1.** Let $SC$ be any set of schematic clauses generated by $SUPC_I$. 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 $SC$.

Without Assumption 1 we would need a specific unification algorithm to handle literals involving $s$ and $s^+$. Thanks to it we can continue applying the superposition rule with syntactic unification.

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

**Proof.** The proof is an adaptation of the one of [12, Theorem 2]. The proof is by induction on the length of derivations of $UPC_I$. 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 $G_\infty$ of $Ax(T) \cup G_0$ by $SUPC_I$, and
2. if a clause is deleted by Subsumption, Tautology or Deletion from (or simplified by Simplification/reduced by Reduction in) $G_\infty$, then all instances of the latter will also be deleted from (or simplified/reduced in) the saturation of $Ax(T) \cup S$ by $UPC_I$.

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 $UPC_I$. By induction hypothesis $l[u'] \bowtie r$ and $u = t$ are instances of schematic clauses in $G_\infty$, i.e. there is some schematic clause $\hat{D}$ in $G_\infty$ such that $l[u'] \bowtie r$ is an instance of $\hat{D}$, and a schematic clause $\hat{E}$ in $G_\infty$ such that $u = t$ is an instance of $\hat{E}$. Two cases can be distinguished:

(*) If there is no occurrence of $s$ in $u$ or $u'$, then there exists a Superposition inference of $SUPC_I$ in $G_\infty$ whose premises are $\hat{D}$ and $\hat{E}$, and whose conclusion is a schematic clause $\hat{C}$ such that $\sigma(l[l] \bowtie r)$ is an instance of $\hat{C}$, where $\sigma$ denotes the most general unifier of $u$ and $u'$.

(**) If there are occurrences of $s$ in both $u$ 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 $\text{UPC}_1$ can be handled in a way similar to Superposition and is therefore omitted.

**Proof of (2).** Let us consider Subsumption of $\text{UPC}_1$. For the case (a), let us assume that there are a schematic clause $A$ deleted from $G_\infty$ and a clause $B$ in the saturation of $Ax(T) \cup S$ by $\text{UPC}_1$, which is an instance of the schematic clause $A$. Then there must exist a clause $C \in Ax(T)$ and some substitution $\theta$ such that $\theta(C) \subseteq A$. Since all the clauses in $Ax(T)$ persist, there must be a substitution $\theta'$ such that $\theta'(C) \subseteq B$. Thereby $B$ must also be deleted from the saturation of $Ax(T) \cup S$ by $\text{UPC}_1$, 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 $\text{UPC}_1$. Assume that there is a schematic clause $C[l'] \parallel \varphi$ in $G_\infty$ simplified by an equality $l = r$ ($l = r \in Ax(T)$) into $C[\theta(r)] \parallel \varphi$. Let $\sigma$ 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 $\text{UPC}_1$, 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 $\text{UPC}_1$, it is easy to see that a constraint instance of a tautology is also a tautology. For the Deletion rule of $\text{UPC}_1$, notice that clauses with an unsatisfiable constraint have no instances.

For the reduction rule $R1$ of $\text{UPC}_1$, it is easy to see that an instance of a schematic clause $s(u) = s(v)$ will also reduce a root symbol $s$. For the reduction rule $R2$ of $\text{UPC}_1$, a similar argument can be given.

**Proof of (3).** Let $C \downarrow_{R^+}$ be the clause obtained from $C$ by replacing the terms occurring in $C$ with their normal forms w.r.t. $R^+$. The set of (concrete) clauses schematized by a schematic clause $C$ is included in the set of (concrete) clauses schematized by $C \downarrow_{R^+}$, because a similar inclusion holds for all the terms in $C$ and all the rules in $R^+$.

### 3.3 Application to the Termination of Paramodulation

Contrary to the standard case, a schematized saturation may represent an infinite set of clauses since the term $s^+(t)$ represents all the terms $s^n(t)$ with $n \geq 1$. The difficulty is then to prove the termination in this case. In [14], the termination proofs do not only rely on the fact that there are finitely many forms of clauses generated by the paramodulation 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 paramodulation allows computing the different forms of clauses generated by paramodulation, we still need an additional analysis to conclude that the paramodulation 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.

Thanks to Assumption 2, there are only finitely many possible instances in the saturation of a satisfiable input. For instance, we cannot have both $i = s(j)$ and $i = s^2(j)$ in the saturation of a satisfiable input due to the acyclicity axiom.

With respect to disequalities, we restrict us to the case where $\text{UPC}_1$ does not generate new disequalities having an occurrence of $s$: the simplification of an input disequality is the only way to have a disequality with an occurrence of $s$ introduced in a derivation with $\text{UPC}_1$.  


This restriction is satisfied in the following cases:

- The set of axioms of the theory contains only equalities.
- The set of axioms of the theory contains some disequalities of a given sort, say $d$, such that it is not possible to build terms of sort $d$ containing $s$.

Hence, this restriction is satisfied for the theories we are interested in.

**Theorem 2.** Assume that $\text{UPC}_I$ does not generate new disequalities having an occurrence of $s$. If $\text{SUPC}_I$ generates a finite schematic saturation such that all its schematic equalities satisfy Assumption 2, then $\text{UPC}_I$ terminates on any input set of ground literals.

**Proof.** Consider a satisfiable input. Let $n_d$ (resp. $n_e$) be the number of disequalities (resp. equalities) obtained from the schematic disequalities (resp. equalities) in the finite schematic saturation by considering all possible instantiations of constrained variables by the finitely many constants in the input. The number of clauses occurring in the saturation of the input can be bounded as follows:

1. By hypothesis, the number of disequalities cannot be greater than $n_d + i_d$, where $i_d$ denotes the number of disequalities in the input set.
2. Consider the equalities. According to Assumption 2, the number of equalities is bounded by $n_e$.

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

One can remark that our restriction on the generation of new disequalities in Theorem 2 is expressed with $\text{UPC}_I$, but not with $\text{SUPC}_I$. We adopt this solution because this restriction is easy to satisfy in practice and it is sufficient for our need. Of course, an interesting problem would be to find a way at the schematic level to ensure the boundedness of the generated disequalities. A possible solution could be envisioned when all the generated schematic literals are ground.

## 4 Implementation

This section presents an implementation of the schematic paramodulation calculus modulo Integer Offsets $\text{SUPC}_I$, by using the Maude system [7] and its support of rewriting logic. It extends an implementation of $\text{SUPC}$ described in [16]. We reuse the expansion and contraction rules implemented in [16]. The new implementation supports many-sorted theories. We discuss the normalization of schematic terms containing $s^*$, the implementation of the new reduction rules and the Schematic Deletion rule. We briefly introduce our support for derivation traces before showing our experimental results.

**Sorts.** The underlying logic of Maude is order-sorted, admitting a subsort ordering, whereas the underlying logic of our calculus $\text{SUPC}_I$ is many-sorted, i.e. there is no subsort relation between sorts in the addressed theories. Let $\Sigma$ be a many-sorted signature and $S$ be its set of sorts. When implementing $\Sigma$ 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.
```
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 $S$.

**Schematic Literals.** We take profit of the powerful reflection mechanism of Maude. Maude terms are reflected as “meta-terms” with sort $\text{Term}$. The base cases in the metarepresentation of terms are given by the subsorts $\text{Constant}$ and $\text{Variable}$ of the sort $\text{Term}$.

Most of our implementation works at the meta-level, i.e. its functions operate on meta-terms with sort $\text{Term}$. Literals are defined by

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

The attribute [comm] declares that the infix binary symbols equals and $\neq$ for equality and disequality are commutative. Then, the sort $\text{SLiteral}$ of schematic literal is declared by

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

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

**Normalization of Schematic Terms.** The rewrite system $Rs^+$ is convergent, i.e. it computes a unique normal form. The $\text{nf}$ function ($\text{nf} : \text{Term} \to \text{Term}$) 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 $\text{nfLit}$ function ($\text{nfLit} : \text{Literal} \to \text{Literal}$) that normalizes both sides of a given literal.

**Ground Reduction Rules.** Let us now present the encoding of the reduction rules of $\text{SUCP}_1$. We translate them into rewrite rules.

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

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

This rule removes the root symbol in both sides of a literal if this root symbol is $\text{'succ}$ ($\text{'succ}$ stands for $s$) and the variables of their subterms $U$ and $U'$ are constrained (are in $\text{Phi}$). This condition is checked by the function $\text{isVarInConstraint}$.

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

```markdown
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)
```

---

1 An $s$ is added to the Maude sort names for integers and lists because Maude sorts named $\text{Int}$ and $\text{List}$ already exist.
if isVarInConstraint(U, Phi1) and isVarInConstraint(V, Phi2) and
isVarInConstraint(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, Phi, t) = true iff u > t. We add the additional parameter Phi that collects the constrained variables, since constrained variables of different sorts are comparable. The function cleanConstraint aims at removing the constrained variables that do no occur in u = v.

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)

The function conditionDel implements the side conditions of the Schematic Deletion rule presented in Figure 5.

Traces. An additional and important feature of our tool consists in providing a trace indicating the name of the applied rule, the schematic clauses it is applied to at each derivation step and the position at which the rule is applied. This trace helps understanding the origin of each new schematic clause.

Let us show the syntax of traces for the superposition rule:

\[
\frac{\sigma[l[u'] \triangleright r || \varphi \quad u = t || \psi]}{\sigma[l[t] \triangleright r || \varphi \land \psi}}
\]

The expression \( \text{sup}(C_1, C_2, u, l[u'], Ctx) \) gives \( C_3 \) means that the constrained clause \( C_3 = \sigma[l[t] \triangleright r || \varphi \land \psi] \) is derived from the constrained clauses \( C_1 = (l[u'] \triangleright r || \varphi) \) and \( C_2 = (u = t || \psi) \) by superposing term \( u \) from \( C_2 \) in term \( l[u'] \) from \( C_1 \) at the context \( Ctx = l[] \), where the rewriting has taken place.

5 Experimental Results

The implementation has been used to compute the schematic saturations for the theories LLI and RII introduced in Section 2.2. These computations generate the expected results, as shown in [17].

5.1 Example 1

Consider \( Ax(LLI) \cup G_0 \) that consists of the empty clause and the following literals:

1. Axioms for lists
   a) \( \text{car} (\text{cons}(X, Y)) = X \)
   b) \( \text{cdr} (\text{cons}(X, Y)) = Y \)
   c) \( \text{cons}(X, Y) \neq \text{nil} \)

2. Axioms for the length
   a) \( \text{len} (\text{cons}(X, Y)) = s (\text{len}(Y)) \)
   b) \( \text{len}(\text{nil}) = 0 \)

3. Schematic literals of sort ELEM
   a) \( \text{car}(a) = e \parallel \text{cons}(a, e) \)
   b) \( e_1 = e_2 \parallel \text{cons}(e_1, e_2) \)
   c) \( e_1 \neq e_2 \parallel \text{cons}(e_1, e_2) \)

4. Schematic literals of sort LISTS
   a) \( \text{cons}(e, a) = b \parallel \text{cons}(e, a, b) \)
   b) \( \text{cdr}(a) = b \parallel \text{cons}(a, b) \)
   c) \( a = b \parallel \text{cons}(a, b) \)
   d) \( a \neq b \parallel \text{cons}(a, b) \)
5. Schematic literals of sort INT
   a) \( \text{len}(a) = s^+(i) \parallel \text{const}(a, i) \)
   b) \( \text{len}(a) = s^+(\text{len}(b)) \parallel \text{const}(a, b) \)
   c) \( i = s^+(\text{len}(a)) \parallel \text{const}(a, i) \)
   d) \( i_1 = s^+(i_2) \parallel \text{const}(i_1, i_2) \)
   e) \( i = \text{len}(a) \parallel \text{const}(a, i) \)
   f) \( i_1 = i_2 \parallel \text{const}(i_1, i_2) \)
   g) \( i_1 \neq i_2 \parallel \text{const}(i_1, i_2) \)

The saturation of \( \text{Ax}(\text{LLI}) \cup G_0 \) by \( \text{SUPC}_I \) consists of \( \text{Ax}(\text{LLI}) \cup G_0 \) and the following schematic literals:

1. \( s^+(i_1) = s^+(i_2) \parallel \text{const}(i_1, i_2) \)
2. \( i_1 \neq s^+(i_2) \parallel \text{const}(i_1, i_2) \)
3. \( s^+(i_1) \neq s^+(i_2) \parallel \text{const}(i_1, i_2) \)
4. \( s^+(i_1) = s^+(\text{len}(a)) \parallel \text{const}(i_1, a) \)
5. \( \text{len}(a) = \text{len}(b) \parallel \text{const}(a, b) \)
6. \( s^+(\text{len}(a)) = s^+(\text{len}(b)) \parallel \text{const}(a, b) \)

Indeed, our tool computes the following trace:

\[
\begin{align*}
\text{sup}(\text{label}(5.d), \text{label}(5.d), i_1, i_1, []) & \text{ gives } s^+(i_1) = s^+(i_2) \parallel \text{const}(i_1, i_2) \\
\text{sup}(\text{label}(5.g), \text{label}(5.d), i_1, i_1, []) & \text{ gives } i_1 \neq s^+(i_2) \parallel \text{const}(i_1, i_2) \\
\text{sup}(\text{label}(2), \text{label}(5.d), i_1, i_1, []) & \text{ gives } s^+(i_1) \neq s^+(i_2) \parallel \text{const}(i_1, i_2) \\
\text{sup}(\text{label}(5.a), \text{label}(5.b), \text{len}(a), \text{len}(a), []) & \text{ gives } s^+(i_1) = s^+(\text{len}(a)) \parallel \text{const}(i_1, a) \\
\text{sup}(\text{label}(5.b), \text{label}(5.b), s^+(\text{len}(b)), s^+(\text{len}(b)), []) & \text{ gives } \text{len}(a) = \text{len}(b) \parallel \text{const}(a, b) \\
\text{sup}(\text{label}(5.b), \text{label}(5.b), \text{len}(a), \text{len}(a), []) & \text{ gives } s^+(\text{len}(a)) = s^+(\text{len}(b)) \parallel \text{const}(a, b)
\end{align*}
\]

5.2 Example 2

Consider \( \text{Ax}(\text{RII}) \cup G_0 \) that consists of the empty clause and the following literals, where \( i, j \) are any integers in \( \{1, 2, 3\} \) such that \( i \neq j \):

1. Axioms for records
   a) \( \text{rselect},(\text{rstore}(X, Y)) = Y \)
   b) \( \text{rselect},(\text{rstore}(X, Y)) = \text{rselect}_i(X, Y) \)
2. Axiom for the increment
   a) \( \text{rselect},(\text{incr}(X)) = s(\text{rselect}(X)) \)
3. Schematic literals of sort REC
   a) \( b = \text{rstore}(a, e) \parallel \text{const}(a, b, e) \)
   b) \( b = \text{incr}(a) \parallel \text{const}(a, b) \)
   c) \( a = b \parallel \text{const}(a, b) \)
   d) \( a \neq b \parallel \text{const}(a, b) \)
4. Schematic literals of sort INT
   a) \( e = \text{rselect}(a) \parallel \text{const}(a, e) \)
   b) \( \text{rselect}_i(a) = s^+(e) \parallel \text{const}(a, e) \)
   c) \( \text{rselect}_i(a) = s^+(\text{rselect}_i(b)) \parallel \text{const}(a, b) \)
   d) \( e = s^+(\text{rselect}(a)) \parallel \text{const}(a, e) \)
   e) \( e_1 = s^+(e_2) \parallel \text{const}(e_1, e_2) \)
   f) \( e_1 = e_2 \parallel \text{const}(e_1, e_2) \)
   g) \( e_1 \neq e_2 \parallel \text{const}(e_1, e_2) \)

The saturation of \( \text{Ax}(\text{RII}) \cup G_0 \) by \( \text{SUPC}_I \) consists of \( \text{Ax}(\text{RII}) \cup G_0 \) and the following schematic literals, where \( i \) is any integer in \( \{1, 2, 3\} \):

1. \( s^+(e_1) = s^+(e_2) \parallel \text{const}(e_1, e_2) \)
2. \( e_1 \neq s^+(e_2) \parallel \text{const}(e_1, e_2) \)
3. \( s^+(e_1) \neq s^+(e_2) \parallel \text{const}(e_1, e_2) \)
4. \( \text{rselect}_i(a) = \text{rselect}_i(b) \parallel \text{const}(a, b) \)
5. \( s^+(\text{rselect}_i(a)) = s^+(\text{rselect}_i(b)) \parallel \text{const}(a, b) \)
6. \( s^+(e_1) = s^+(\text{rselect}_i(a)) \parallel \text{const}(a, e_1) \)
7. \( \text{rstore}(a, s^+(e)) = b \parallel \text{const}(a, b, e) \)
Indeed, our tool computes the following trace:

\[
\sup([\text{label}(4,c), \text{label}(4,e), e_1, e_1, []]) \text{ gives } s^+(e_1) = s^+(e_2) \ || \ const(e_1, e_2)
\]

\[
\sup([\text{label}(4,g), \text{label}(4,c), e_1, e_1, []]) \text{ gives } e_1 \neq s^+(e_2) \ || \ const(e_1, e_2)
\]

Both examples satisfy Assumption 1 given in Section 3.2. In fact, \(s\)-rooted terms occur only in the set of axioms of theories \(LLI\) and \(RII\). In both cases the \(s\)-rooted term is not the maximal one, therefore, the \textit{Superposition} rule cannot be applied between this axiom and any other literal containing an \(s^+\)-rooted term. Let us check that Assumption 2 given in Section 3.3 is also satisfied. Assume \(m\) and \(n\) are distinct strictly positive integers. Clearly, \(s^m(j)\) and \(s^n(j)\) denote different values, and so we cannot have both \(i = s^m(j)\) and \(i = s^n(j)\) in a saturation of a satisfiable input. Moreover, \(rstore(a, s^m(e))\) and \(rstore(a, s^n(e))\) denote different records, and so we cannot have both \(b = rstore(a, s^n(e))\) and \(b = rstore(a, s^m(e))\) in a saturation of a satisfiable input. Consequently, according to Theorem 2, we can conclude that the paramodulation calculus terminates for both examples.

\section{Conclusion}

This paper has introduced a new schematic calculus integrating the axioms of the Integer Offsets theory into a framework based on schematic paramodulation. In this context, introducing the \(s^+\) operator together with 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 automatically prove the termination of paramodulation modulo Integer Offsets for data structures equipped with counting operators. Our schematic calculus is described as a rule-based system and implemented and validated in the Maude environment.

This paper is the first extension of the notion of schematic paramodulation dedicated to a paramodulation calculus modulo a built-in theory. This study has led to new automatic proof techniques that are different from those performed manually in [14]. 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 order to allow for instance arrays with counting operators. Decision procedures for some extensions of the theory of arrays already exist (see, e.g. [5, 9]) but our approach would additionally provide automated proofs of decidability. In this direction, we would have to find a less restrictive assumption to guarantee termination, possibly via a criterion involving the simplification ordering. As in [12], we plan to study some other combinability conditions, and to work on the possibility of determining an upper bound on the number of clauses generated in saturation. Another research direction is to consider a schematic calculus for a more expressive built-in theory of arithmetic, like the theory of Abelian Groups [10, 13].
References


