Automatic Decidability: A Schematic Calculus for Theories with Counting Operators

Elena Tushkanova[1,2], Christophe Ringeissen[1], Alain Giorgetti[1,2], and Olga Kouchnarenko[1,2]

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


© Elena Tushkanova, Christophe Ringeissen, Alain Giorgetti, and Olga Kouchnarenko;
licensed under Creative Commons License CC-BY
24th International Conference on Rewriting Techniques and Applications (RTA'13).
Editor: Femke van Raamsdonk; pp. 303–318

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


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 instance, 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 conditions 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* 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 [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 $t[l]_p$ 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 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 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* [8] if it is stable ($l < r$ implies $\sigma(l) < \sigma(r)$ 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 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 $\mathcal{UPC}$ [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 *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 *Subsumption*, a literal is removed when it is an instance of another one. *Simplification* rewrites a literal into a simpler one by using an equality that can be considered as a rewrite rule. *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 $\mathcal{UPC}$ 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 [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 *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 $S_0, S_1, ..., S_i, ...$ 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 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:

$$\begin{array}{lll} (s0) & \forall X. & \mathsf{s}(X) \neq \mathsf{0} \\ (inj) & \forall X, Y. & \mathsf{s}(X) = \mathsf{s}(Y) \Rightarrow X = Y \\ (acy) & \forall X. & X \neq \mathsf{s}^n(X) \quad \text{for all} \quad n \geq 1 \end{array}$$

over the signature $\Sigma_I := \{\mathsf{0} : \textsc{int}, \mathsf{s} : \textsc{int} \to \textsc{int}\}$. 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 INT, whose signature shares symbols with Σ_I, and whose axioms possibly involve the symbols s and $\mathsf{0}$. Following [14, Section 5], we consider two Integer Offsets extensions: the theory of lists with length whose signature is $\Sigma_{LLI} = \{\mathsf{car} : \textsc{lists} \to \textsc{elem}, \mathsf{cdr} : \textsc{lists} \to \textsc{lists}, \mathsf{cons} : \textsc{elem} \times \textsc{lists} \to \textsc{lists}, \mathsf{len} : \textsc{lists} \to \textsc{int}, \mathsf{nil} :\to \textsc{lists}, \mathsf{0} :\to \textsc{int}, \mathsf{s} : \textsc{int} \to \textsc{int}\}$ and whose set of axioms $Ax(LLI)$ consists of

$$\begin{array}{ll} \mathsf{car}(\mathsf{cons}(X, Y)) = X & \mathsf{cons}(X, Y) \neq \mathsf{nil} \\ \mathsf{cdr}(\mathsf{cons}(X, Y)) = Y & \mathsf{len}(\mathsf{nil}) = \mathsf{0} \\ \mathsf{len}(\mathsf{cons}(X, Y)) = \mathsf{s}(\mathsf{len}(Y)) & \end{array}$$

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

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

where X is a universally quantified variable of sort REC, and Y is a universally quantified variable of sort INT.

R1 $\quad \dfrac{S \cup \{\mathsf{s}(u) = \mathsf{s}(v)\}}{S \cup \{u = v\}}$ **if** u and v are ground terms.

R2 $\quad \dfrac{S \cup \{\mathsf{s}(u) = t, \mathsf{s}(v) = t\}}{S \cup \{\mathsf{s}(v) = t, u = v\}}$

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

C1 $\quad \dfrac{S \cup \{\mathsf{s}(t) = 0\}}{S \cup \{\mathsf{s}(t) = 0, \bot\}}$ **if** t is a ground term.

C2 $\quad \dfrac{S \cup \{\mathsf{s}^n(t) = t\}}{S \cup \{\mathsf{s}^n(t) = t, \bot\}}$ **if** t is a ground term and $n \geq 1$.

Figure 3 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 $T_I - \mathit{good}$ ordering $<$ to control $\mathcal{UPC}_I$: $<$ is a simplification ordering which is total on ground terms, such that $\mathsf{0}$ 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 $t_1 = t_2$ and n_1 is bigger than n_2, or $t_1 > t_2$. The refutation completeness of $\mathcal{UPC}_I$ 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 $\mathcal{SUPC}$, as 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 $\mathcal{SUPC}$ [12, 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 [12].

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

Let us recall the notions of constrained clause and instance of constrained clause used in [12] for $\mathcal{SUPC}$. An *atomic constraint* is of the form $\mathit{const}(t)$, 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, $\mathit{const}(t_1, \ldots, t_n)$ denotes the conjunction $\mathit{const}(t_1) \wedge \cdots \wedge \mathit{const}(t_n)$. 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 $\mathit{const}(x)$ 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. By a slight abuse of notation, we say that a term u is *ground* with respect to a constraint φ if all the variables in u are constrained (are in φ).

3 Schematic Paramodulation Calculus for Integer Offsets

This section introduces a new schematic calculus denoted by $\mathcal{SUPC}_I$. It is a schematization of $\mathcal{UPC}_I$ 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 $\mathsf{s}^n(t)$ 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 $\{\mathsf{s}^n(t) \mid n \geq 1\}$. A *schematic term* is a term containing s^+.

The calculus $\mathcal{SUPC}_I$ handles schematic clauses that extend constrained clauses of $\mathcal{SUPC}$.

▶ 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 $\mathcal{SUPC}_I$ takes as input a set of schematic literals, G_0, that represents all possible sets of ground literals given as inputs to $\mathcal{UPC}_I$:

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

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

3.1 Schematic Calculus

The calculus $\mathcal{SUPC}_I$ is depicted in Figures 4, 5 and 6. It re-uses most of the rules of $\mathcal{SUPC}$ (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 $\mathcal{SUPC}_I$ 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 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.

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 4 Schematic expansion rules.

We use a specific term rewrite system to simplify schematic terms. Hence, the rewrite system $R\mathsf{s}^+ = \{\ \mathsf{s}^+(\mathsf{s}(x)) \to \mathsf{s}^+(x),\ \mathsf{s}(\mathsf{s}^+(x)) \to \mathsf{s}^+(x),\ \mathsf{s}^+(\mathsf{s}^+(x)) \to \mathsf{s}^+(x)\ \}$ 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 $\mathcal{UPC}_I$ (Figure 3) do not appear in $\mathcal{SUPC}_I$. This is due to the fact that these rules produce only the empty clause $\bot$ 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 $\mathcal{SUPC}_I$. Let us motivate this rule by considering the theory of lists with length. In fact, the calculus generates a schematic clause $\mathsf{len}(a) = \mathsf{s}(\mathsf{len}(b)) \| const(a, b)$ which will superpose with a renamed copy of itself, i.e. with $\mathsf{len}(a') = \mathsf{s}(\mathsf{len}(b'))\ \| const(a', b')$ to generate a schematic clause of a new form $\mathsf{len}(a) = \mathsf{s}(\mathsf{s}(\mathsf{len}(b')))\ \| const(a, b')$. This process continues to generate deeper and deeper schematic clauses so that the Schematic Saturation will diverge. To 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 π replacing all the occurrences of s in C' by s^+ ($\pi(\mathsf{s}(t)) = \mathsf{s}^+(\pi(t))$ for any t, $\pi(x) = x$ if x is a variable), and then the rewrite system $R\mathsf{s}^+$ to replace a chain of s^+ occurrences in $\pi(C')$ by only one occurrence of

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

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

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 $\dfrac{S \cup \{u = u\|\varphi\}}{S}$

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

Schematic Del. $\dfrac{S \cup \{C'\|\varphi, C[\mathsf{s}^+(t)]\|\psi\}}{S \cup \{C[\mathsf{s}^+(t)]\|\psi\}}$

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

Figure 5 Schematic contraction rules.

R1 $\dfrac{S \cup \{\mathsf{s}(u) = \mathsf{s}(v)\|\varphi\}}{S \cup \{u = v\|\varphi\}}$ **if** u and v are ground terms.

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

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

Figure 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 $\mathsf{len}(a) = \mathsf{s}^+(\mathsf{len}(b))\|const(a, b)$, and so it subsumes a schematic clause like $\mathsf{len}(a) = \mathsf{s}(\mathsf{s}(\mathsf{len}(b'))) \ \|const(a, b')$. 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 $\mathsf{rselect}_i(a) = \mathsf{s}^+(\mathsf{rselect}_i(b))\|const(a, b)$. 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 $u = \mathsf{s}^+(v)\|\varphi$, where u and v are two flat terms of sort INT whose variables are constrained by φ.

3.2 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 $\mathcal{SUPC}_I$, under the following assumption.

▶ 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.

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 $\mathcal{UPC}_I$. 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 [12, 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* 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 or u', then there exists a *Superposition* inference of $\mathcal{SUPC}_I$ in G_∞ whose premises are $\hat{D}$ and $\hat{E}$, and whose conclusion is a schematic clause $\hat{C}$ such that $\sigma(l[t] \bowtie r)$ is an instance of $\hat{C}$, where σ 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 $\mathcal{UPC}_I$ 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 us 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). Let $C \downarrow_{R\mathsf{s}^+}$ be the clause obtained from C by replacing the terms occurring in C with their normal forms w.r.t. $R\mathsf{s}^+$. 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 $R\mathsf{s}^+$. ◀

3.3 Application to the Termination of Paramodulation

Contrary to the standard case, a schematized saturation may represent an infinite set of clauses since the term $\mathsf{s}^+(t)$ represents all the terms $s^n(t)$ with $n \geq 1$. The difficulty is then to prove the termination in this case. In [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 $i = \mathsf{s}(j)$ and $i = \mathsf{s}^2(j)$ in the saturation of a satisfiable input due to the acyclicity axiom.

With respect to disequalities, we restrict us to the case where $\mathcal{UPC}_I$ 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 $\mathcal{UPC}_I$.

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 $\mathcal{UPC}_I$ does not generate new disequalities having an occurrence of* s*. If $\mathcal{SUPC}_I$ generates a finite schematic saturation such that all its schematic equalities satisfy Assumption 2, then $\mathcal{UPC}_I$ 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 $\mathcal{UPC}_I$, but not with $\mathcal{SUPC}_I$. 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 $\mathcal{SUPC}_I$, by using the Maude system [7] and its support of rewriting logic. It extends an implementation of $\mathcal{SUPC}$ 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 $\mathcal{SUPC}_I$ 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.

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 literal 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.

Normalization of Schematic Terms. The rewrite system $R\mathsf{s}^+$ is convergent, i.e. it computes a unique normal form. The `nf` function (`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.

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:

```
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 `'succ` (`'succ` stands for `s`) and the variables of their subterms `U` and `U'` are constrained (are in `Phi`). This condition is checked by the function `isVarInConstraint`.

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)
```

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

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

The expression `sup(`C_1`,` C_2`,` u`,` $l[u']$`,` Ctx`)` `gives` C_3 means that the constrained clause $C_3 = \sigma(l[t] \bowtie r \| \varphi \wedge \psi)$ is derived from the constrained clauses $C_1 = (l[u'] \bowtie r \| \varphi)$ 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) $\mathsf{car}(\mathsf{cons}(X, Y)) = X$
   b) $\mathsf{cdr}(\mathsf{cons}(X, Y)) = Y$
   c) $\mathsf{cons}(X, Y) \neq \mathsf{nil}$
2. Axioms for the length
   a) $\mathsf{len}(\mathsf{cons}(X, Y)) = \mathsf{s}(\mathsf{len}(Y))$
   b) $\mathsf{len}(\mathsf{nil}) = 0$
3. Schematic literals of sort ELEM
   a) $\mathsf{car}(a) = e \parallel const(a, e)$
   b) $e_1 = e_2 \parallel const(e_1, e_2)$
   c) $e_1 \neq e_2 \parallel const(e_1, e_2)$
4. Schematic literals of sort LISTS
   a) $\mathsf{cons}(e, a) = b \parallel const(e, a, b)$
   b) $\mathsf{cdr}(a) = b \parallel const(a, b)$
   c) $a = b \parallel const(a, b)$
   d) $a \neq b \parallel const(a, b)$

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

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

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

Indeed, our tool computes the following trace:

$\mathbf{sup}(label(5.d), label(5.d),\ i_1,\ i_1,\ [])$ **gives** $\mathsf{s}^+(i_1) = \mathsf{s}^+(i_2) \parallel const(i_1, i_2)$
$\mathbf{sup}(label(5.g), label(5.d),\ i_1,\ i_1,\ [])$ **gives** $i_1 \neq \mathsf{s}^+(i_2) \parallel const(i_1, i_2)$
$\mathbf{sup}(label(2), label(5.d),\ i_1,\ i_1,\ [])$ **gives** $\mathsf{s}^+(i_1) \neq \mathsf{s}^+(i_2) \parallel const(i_1, i_2)$
$\mathbf{sup}(label(5.a), label(5.b),\ \mathsf{len}(a),\ \mathsf{len}(a),\ [])$ **gives** $\mathsf{s}^+(i_1) = \mathsf{s}^+(\mathsf{len}(a)) \parallel const(i_1, a)$
$\mathbf{sup}(label(5.b), label(5.b),\ \mathsf{s}^+(\mathsf{len}(b)),\ \mathsf{s}^+(\mathsf{len}(b)),\ [])$ **gives** $\mathsf{len}(a) = \mathsf{len}(b) \parallel const(a, b)$
$\mathbf{sup}(label(5.b), label(5.b),\ \mathsf{len}(a),\ \mathsf{len}(a),\ [])$ **gives** $\mathsf{s}^+(\mathsf{len}(a)) = \mathsf{s}^+(\mathsf{len}(b)) \parallel const(a, b)$

5.2 Example 2

Consider $Ax(RII) \cup G_0$ 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) $\mathsf{rselect}_i(\mathsf{rstore}_i(X, Y)) = Y$
   b) $\mathsf{rselect}_j(\mathsf{rstore}_i(X, Y)) = \mathsf{rselect}_j(X, Y)$
2. Axiom for the increment
   a) $\mathsf{rselect}_i(\mathsf{incr}(X)) = \mathsf{s}(\mathsf{rselect}_i(X))$
3. Schematic literals of sort REC
   a) $b = \mathsf{rstore}_i(a, e) \parallel const(a, b, e)$
   b) $b = \mathsf{incr}(a) \parallel const(a, b)$
   c) $a = b \parallel const(a, b)$
   d) $a \neq b \parallel const(a, b)$
4. Schematic literals of sort INT
   a) $e = \mathsf{rselect}_i(a) \parallel const(a, e)$
   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) $e = \mathsf{s}^+(\mathsf{rselect}_i(a)) \| const(a, e)$
   e) $e_1 = \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
   f) $e_1 = e_2 \parallel const(e_1, e_2)$
   g) $e_1 \neq e_2 \parallel const(e_1, e_2)$

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

1. $\mathsf{s}^+(e_1) = \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
2. $e_1 \neq \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
3. $\mathsf{s}^+(e_1) \neq \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
4. $\mathsf{rselect}_i(a) = \mathsf{rselect}_i(b) \parallel const(a, b)$
5. $\mathsf{s}^+(\mathsf{rselect}_i(a)) = \mathsf{s}^+(\mathsf{rselect}_i(b)) \parallel const(a, b)$
6. $\mathsf{s}^+(e_1) = \mathsf{s}^+(\mathsf{rselect}_i(a)) \parallel const(a, e_1)$
7. $\mathsf{rstore}_i(a, \mathsf{s}^+(e)) = b \parallel const(a, b, e)$

Indeed, our tool computes the following trace:

$\mathbf{sup}(label(4.e), label(4.e),\ e_1,\ e_1,\ [])\ \mathbf{gives}\ \mathsf{s}^+(e_1) = \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
$\mathbf{sup}(label(4.g), label(4.e),\ e_1,\ e_1,\ [])\ \mathbf{gives}\ e_1 \neq \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
$\mathbf{sup}(label(2), label(4.e),\ e_1,\ e_1,\ [])\ \mathbf{gives}\ \mathsf{s}^+(e_1) \neq \mathsf{s}^+(e_2) \parallel const(e_1, e_2)$
$\mathbf{sup}(label(4.c), label(4.c),\ \mathsf{s}^+(\mathsf{rselect}_i(b)),\ \mathsf{s}^+(\mathsf{rselect}_i(b)),\ [])\ \mathbf{gives}$
$\mathsf{rselect}_i(a) = \mathsf{rselect}_i(b) \parallel const(a, b)$
$\mathbf{sup}(label(4.c), label(4.c),\ \mathsf{rselect}_i(a),\ \mathsf{rselect}_i(a),\ [])\ \mathbf{gives}$
$\mathsf{s}^+(\mathsf{rselect}_i(a)) = \mathsf{s}^+(\mathsf{rselect}_i(b)) \parallel const(a, b)$
$\mathbf{sup}(label(4.c), label(4.b),\ \mathsf{rselect}_i(a),\ \mathsf{rselect}_i(a),\ [])\ \mathbf{gives}\ \mathsf{s}^+(e) = \mathsf{s}^+(\mathsf{rselect}_i(a)) \parallel const(a, e)$
$\mathbf{sup}(label(3.a), label(4.e),\ e_1,\ \mathsf{rstore}_i(a, e),\ \mathsf{rstore}_i(a, []),)\ \mathbf{gives}$
$\mathsf{rstore}_i(a, \mathsf{s}^+(e)) = b \parallel const(a, b, e)$

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 *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, $\mathsf{s}^m(j)$ and $\mathsf{s}^n(j)$ denote different values, and so we cannot have both $i = \mathsf{s}^m(j)$ and $i = \mathsf{s}^n(j)$ in a saturation of a satisfiable input. Moreover, $\mathsf{rstore}_i(a, \mathsf{s}^m(e))$ and $\mathsf{rstore}_i(a, \mathsf{s}^n(e))$ denote different records, and so we cannot have both $b = \mathsf{rstore}_i(a, \mathsf{s}^m(e))$ and $b = \mathsf{rstore}_i(a, \mathsf{s}^n(e))$ in a saturation of a satisfiable input. Consequently, according to Theorem 2, we can conclude that the paramodulation calculus terminates for both examples.

6 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

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

2 A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *J. 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 *LNCS*, 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 *LNCS*, pages 153–167. Springer, 2006.

5 A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In E. A. Emerson and K. S. Namjoshi, editors, *VMCAI*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.

6 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 R. Treinen, editor, *RTA*, volume 5595 of *LNCS*, pages 380–390. Springer, 2009.

7 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, *RTA*, volume 2706 of *LNCS*, pages 76–87. Springer, 2003.

8 N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier and MIT Press, 1990.

9 S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision procedures for extensions of the theory of arrays. *Ann. Math. Artif. Intell.*, 50(3-4):231–254, 2007.

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

11 C. Lynch and B. Morawska. Automatic decidability. In *LICS*, pages 7–16, Copenhagen, Denmark, July 2002. IEEE Computer Society.

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

13 E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of Abelian groups. In R. Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 51–66. Springer, 2009.

14 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.

15 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.

16 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 221–239. Springer, 2012.

17 E. Tushkanova, C. Ringeissen, A. Giorgetti, and O. Kouchnarenko. Automatic Decidability for Theories Modulo Integer Offsets. Research Report RR-8139, INRIA, November 2012.