Science of Computer Programming 99 (2015) 3–23

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A rule-based system for automatic decidability and combinability

E. Tushkanova [a,b], A. Giorgetti [a,b,*], C. Ringeissen [a], O. Kouchnarenko [a,b]

[a] Inria, Villers-les-Nancy, F-54600, France
[b] FEMTO-ST Institute (UMR 6174), University of Franche-Comté, Besançon, F-25030, France

HIGHLIGHTS

- We present a many-sorted schematic superposition calculus for non-unit theories.
- The calculus is presented as a rule-based system.
- The implementation automatically checks termination for some theories of interest.
- Combinability of signature-disjoint theories can also be checked.

ARTICLE INFO

Article history:
Received 14 January 2013
Received in revised form 31 January 2014
Accepted 6 February 2014
Available online 27 February 2014

Keywords:
Decision procedures
Superposition
Schematic saturation

ABSTRACT

This paper deals with decision procedures specified by using a superposition calculus which is an inference system at the core of all equational theorem provers. This calculus is refutation complete: it provides a semi-decision procedure that halts on unsatisfiable inputs but may diverge on satisfiable ones. Fortunately, it may also terminate for some theories of interest in verification, and thus it becomes a decision procedure. To reason on the superposition calculus, a schematic superposition calculus has been developed to build the schematic form of the saturations allowing to automatically prove decidability of single theories and of their combinations.

This paper presents a rule-based logical framework and a tool implementing a complete many-sorted schematic superposition calculus for arbitrary theories. By providing results for unit theories, arbitrary theories, and also for theories with counting operators, we show that this tool is very useful to derive decidability and combinability of theories of practical interest in verification.

1. Introduction

Satisfiability procedures modulo background theories such as classical data structures (e.g., lists, records, arrays, ...) are at the core of many state-of-the-art verification tools. Designing and implementing satisfiability procedures is a very complex task, where one of the main difficulties consists in proving their soundness.

To overcome this problem, the rewriting approach [1] allows building satisfiability procedures in a flexible way, by using a superposition calculus [2] (also called *Paramodulation Calculus* in [3]). In general, a fair and exhaustive application of the rules of this calculus leads to a semi-decision procedure that halts on unsatisfiable inputs (the empty clause is generated)

* Corresponding author.
E-mail addresses: elena.tushkanova@femto-st.fr (E. Tushkanova), alain.giorgetti@femto-st.fr (A. Giorgetti), Christophe.Ringeissen@loria.fr (C. Ringeissen), olga.kouchnarenko@femto-st.fr (O. Kouchnarenko).

http://dx.doi.org/10.1016/j.scico.2014.02.005
0167-6423/

but may diverge on satisfiable ones. Therefore, the superposition calculus provides a decision procedure for the theory of interest if one can show that it terminates on every input made of the (finitely many) axioms and any set of ground literals. The needed termination proof can be done by hand, by analyzing the (finitely many) forms of clauses generated by saturation, but the process is tedious and error-prone. To simplify this process, a schematic superposition calculus has been developed [3] to build the schematic form of the saturations. This schematic superposition calculus is very useful to analyze the behavior of the superposition calculus on a given input theory, as shown in [4] to prove automatically the termination and the combinability of the related decision procedure.

In [5] a schematic superposition calculus for *unit* theories has been considered. The calculus being defined by an inference system we have proposed a rule-based system to execute it. For the implementation the Maude system has been used because it includes support for unification and narrowing, which are key operations of the calculus of interest, and the Maude meta-level provides a flexible way to control the application of rules and powerful search mechanisms.

In this paper we go further and consider the general schematic superposition calculus for *arbitrary* theories, not only its restriction to unit ones. The main contribution is the presentation of a rule-based framework and a tool implementing a complete many-sorted schematic superposition calculus for non-unit theories. The tool allows us to automatically check whether the superposition calculus terminates for theories defined by arbitrary clauses. Moreover, the tool can be used to check modular termination when a combination of signature-disjoint theories is considered.

The design of a schematic paramodulation calculus for general clauses is much more involved than for unit clauses. Indeed the first version proposed in [3] contained a flaw, namely a non-termination issue. This issue was addressed by [4], by considering a new deletion rule specific to non-unit clauses. But that rule did not take into account the constants in the theory signature (such as the constant nil in the signature of the theory of possibly empty lists). One of our contributions is to propose and implement a new schematic paramodulation calculus properly taking these constants into account. Another contribution consists in showing that the schematic superposition calculus halts for the theories in [6], such as the theory of lists (with and without extensionality), the theory of records, the theory of possibly empty lists and the theory of arrays. Moreover, our implementation of schematic superposition provides a trace of each applied rule which is very useful to validate or invalidate saturation proofs previously described in the literature.

Our tool is also very useful for further investigations related to paramodulation calculi extending the standard paramodulation. We present here the case of a (schematic) paramodulation calculus modulo a simple fragment of arithmetic, called Integer Offsets. This extension allows us to consider classical data structures equipped with counting operators. Our tool produces automatically the schematic saturations of the corresponding theories.

The paper is structured as follows. After introducing preliminary notions and presenting paramodulation calculus in Section 2, Section 3 presents a schematic paramodulation calculus that can be used for proving termination of any fair paramodulation strategies and for checking whether paramodulation calculus decides some unions of finitely presented theories. Section 5 explains how we implement the schematic paramodulation calculus using the Maude system. Then Section 6 reports on our experimentations with the tool, to prove the termination of superposition for theories corresponding to classical data structures such as lists, records and arrays. Section 4 reports on our experiments with the extension to Integer Offsets. Finally, Section 7 concludes and presents future work.

2. Background

2.1. First-order logic

We consider many-sorted first-order logic with equality. A (many-sorted) *signature* Σ consists of a set of sorts S together with a set of function symbols and a set of predicate symbols. A function symbol is declared using a form $f : s_1 \times \cdots \times s_n \to s$, where $n \geqslant 0$ is its arity, $s_1, \ldots, s_n$ and s are *sorts* in S. The sorts $s_1, \ldots, s_n$ are called the *argument sorts* and s is called the *value sort* of f. The set of predicate symbols contains only equality symbols of the form $=_s : s \times s$ for each $s \in S$. The equality $=_s$ is simply denoted by $=$ when the sort s is clear from the context. Usually, we will just give the signature without explicitly mentioning the equality symbols. For instance, $\Sigma_L = \{\mathsf{nil} : \textsc{lists},\ \mathsf{car} : \textsc{lists} \to \textsc{elem},\ \mathsf{cdr} : \textsc{lists} \to \textsc{lists},\ \mathsf{cons} : \textsc{elem} \times \textsc{lists} \to \textsc{lists}\}$ is a usual signature for lists.

Given a signature Σ, we assume the usual first-order notions of term, position, literal, clause and substitution, as defined, e.g., in [7], and only detail here some notions and notations that are less classical and essential to understand the rest of the paper.

Given a term t and a position p, $t_{|p}$ denotes the subterm of t at position p, and $t[l]_p$ denotes a term t such that $t_{|p} = l$. By a standard abuse of notation, we (ambiguously) denote by $t[r]_p$ the term obtained from t by replacing the subterm at position p by r. 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.

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 $depth(f(t_1, \ldots, t_n)) = 1 + \max\{depth(t_i) \mid 1 \leqslant i \leqslant n\}$. 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 $l \neq r$, where l and r have the same sort. 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.