Dei: A Theorem Prover for Terms with Integer Exponents

Hicham Bensaid[1], Ricardo Caferra[2], and Nicolas Peltier[2]

[1] INPT/LIG
Avenue Allal Al Fassi - Madinat Al Irfane - Rabat - Morocco
`bensaid@inpt.ac.ma`
[2] LIG, Grenoble INP/CNRS
Bâtiment IMAG C - 220, rue de la Chimie - 38400 Saint Martin d'Hères - France
`Ricardo.Caferra@imag.fr, Nicolas.Peltier@imag.fr`

Abstract. An extension of the superposition-based E-prover [8] is described. The extension allows terms with integer exponents [3] in the input language. Obviously, this possibility increases the capabilities of the E-prover particularly for preventing non-termination.

1 Introduction

Term schematisations allow one to denote infinite sequences of iterated terms, which frequently occur in many symbolic computation procedures (in particular in proof procedures). The number of iterations is part of the term syntax and may be a variable. In some cases, the capability to denote such sequences avoids non termination [6]. For instance the clause set $\{even(0), \forall x. even(x) \Rightarrow even(s(s(x)))\}$ can be replaced by the unit clause $\forall n. even(s^{2n}(0))$.

There exists a hierarchy of term schematisation languages, with different expressive powers. They mainly differ from each other by the class of inductive contexts that can be handled. The original formalism [2] allows only ground contexts with no nested iterations. [3] extends the language to any inductive context, provided that the inductive path is unique. [7] showed how to get rid of this last condition, allowing for instance sequences of the form $a, f(a,a), f(f(a,a), f(a,a)), \ldots$. Finally, the most powerful language of *primal grammars* [4] handles contexts depending on the iteration rank (as in the sequence $[], [0], [s(0), 0], [s(s(0)), s(0), 0], \ldots$). Unification is decidable for all these languages, thus they can be included in most symbolic computation procedures, in particular in first-order theorem provers. This significantly extends the expressive power of the input language.

In this paper we describe the first (to the best of our knowledge) system to perform inferences on clauses containing term schematisations. As the new prover is an extension of the *E*-prover [8], we have called it Dei (for **D**eduction with the **E**-prover and **I**-terms). The E-prover has been chosen as a starting point because it is well-known, widely distributed and efficient. Our system uses the language of *terms with integer exponents* [3] also called *I*-terms. This formalism is a good compromise between expressive power and simplicity: the ρ-terms [2] are easier to handle but lack expressivity and the primal grammars are very expressive but harder to use in practice (one has to define rewrite systems "outside" the iterated terms, moreover the unification algorithm is rather complex).

We hope that this work, by allowing practical experimentations, will promote the use of term schematisation languages in automated deduction and will allow further investigations (potential applications, better understanding of the formalisms etc.).

2 Terms with Integer Exponents (I-Terms)

We briefly recall the definition of terms with integer exponents (see [3] for details). We assume that three sets of symbols are given: function symbols Σ, standard variables $\mathcal{V}$ and arithmetic variables $\mathcal{V}_N$. Let $\diamond$ be a special symbol, called the "hole". This symbol denotes the changing part of a context.

The set of *terms with one hole* $T_\diamond$ and the set of *terms with integer exponents* T_I (or *I-terms*) are the smallest sets satisfying the following conditions: (i) $\diamond \in T_\diamond$ and $\mathcal{V} \subseteq T_I$. (ii) If $t_1, \ldots, t_n \in T_I$, $t'_i \in T_\diamond$ and $f \in \Sigma$ then $f(t_1, \ldots, t_n) \in T_I$ and $f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n) \in T_\diamond$. (iii) If $t \in T_\diamond, t \neq \diamond, s \in T_I$ and $n \in \mathcal{V}_N \cup \mathbb{N}$ then $t^n.s \in T_I$. An I-term of the last form is called an *N-term*. I-terms can be naturally incorporated as arguments of literals. A clause built on I-terms is called an *I-clause*.

The semantics of I-terms are specified by the following rewrite system: $\{t^0.s \to s, t^{n+1}.s \to t\{\diamond \leftarrow t^n.s\}\}$. These rules obviously rewrite every *ground* I-term t to a (unique) standard term, denoted by $t\downarrow$. The value of t in an interpretation I is the same as the value of $t\downarrow$. This allows one to assign a value to every ground term, hence to evaluate every ground clause. The semantics is extended to the non ground case by interpreting a clause set as the set of its ground instances (variables in $\mathcal{V}_N$ are mapped to natural numbers).

3 The DEI System

The DEI system is freely available from the webpage: `http://capp.imag.fr/dei.html`. It is based on the $0.999 - 004$ *"Longview2"* release of E-Prover. The extension as for E-Prover is written in ANSI C, using the gcc compiler (the 4.2.4 version) and was successfully tested on a GNU Linux x86 platform (Ubuntu 8.04 Hardy Heron).

In order to preserve modularity and allow further extensions, we have tried to restrict as much as possible the modifications in the original code. Additional data structures have been designed to represent linear Diophantine expressions and the algorithm of [3] for unifying I-terms has been implemented. The Polylib library [5] is used for solving arithmetic constraints. About 4000 lines of code have been added to the E-prover (excluding the Polylib library). Modifications have been necessary in various parts of the code: term sharing handling, term definitions, input/output algorithms and inference machine. As unification of I-terms gives in general *several* maximal unifiers, many changes in the code were needed to take into account this important new feature.

3.1 Input Syntax

The syntax has been kept almost identical to the one of the E-prover. All options are supported. An N-term $t^n.s$ is denoted by `$iterm(t,s)^{n}`, where `$iterm` is a reserved keyword. `@` denotes the hole $\diamond$. Exponents can be any linear diophantine expression, which makes the language more flexible than using variables only (as it is done in [3] to simplify theoretical work). For instance the I-term $even(\diamond)^{2n}(0)$ of page 1 is encoded by `$iterm(even(@),0)^{ n+n }`. This flexibility also has the advantage that all the arithmetic constraints can be expressed in the terms themselves, which avoids having to use constrained clauses (for instance the clause $p(f^n(\diamond).0)$ with constraint $\exists m.n = m + m$ can be denoted by $p(f^{2\times m}(\diamond).0)$. Arithmetic expressions must only occur in an exponent (e.g. the term $g(n, f(\diamond)^n.a)$ is not allowed).

The arithmetic variables are interpreted by non-zero natural numbers. This assumption slightly simplifies some steps of the unification algorithm. For instance the occur check rule $(x = t) \rightarrow \mathit{false}$ can be applied as usual, whereas it would not be correct if t contains exponents that can be interpreted as 0, for instance $f(x, \diamond)^n.a = x$ has a solution, namely $\{n \mapsto 0, x \mapsto a\}$.

3.2 Inferences

All the inference rules of the E-prover (i.e. the **SP** calculus) are supported. Obviously, correctness is preserved. The calculus is complete for non equational I-clause sets (it is easy to see that one can "lift" the ground derivations to I-clauses). However, it is no more complete in general for *equational* I-clauses (this does not depend on the strategy). A simple counterexample proving this is the following: $\{p(f(a, \diamond)^n.b, \neg p(c), f(a, b) \approx d, f(a, d) \approx c\}$ is clearly unsatisfiable (it suffices to replace n by 2 in the first clause) but the superposition calculus generates only the clause $p(d)$. This is due to the fact that superposition can be applied at an arbitrary deep position in the standard terms denoted by an N-term – in particular at positions not occurring in the N-term. Allowing superposition at arbitrary deep positions in an N-term is possible, but this would make the inference rule infinitary (and inefficient). This has been avoided in DEI, for practical reasons. The calculus is still complete for some subclasses, e.g. if the superposition rule cannot be applied along iterated paths (of course, the formal definition of these classes is outside the scope of the present system description).

The usual term orderings (KBO or LPO) have been adapted in order to be applied on I-terms (it is obvious that they are not compatible with unfolding[3]). The simplification and redundancy rules are supported, but using an incomplete matching algorithm (thus DEI miss some simplifications). This is due to the fact that the matching is harder to perform on I-terms than in standard terms. In particular, the indexing techniques used by the E-prover (e.g. the perfect discrimination trees) cannot be directly applied.

[3] Of course the orderings can be defined as usual at the ground level i.e. on the standard terms obtained by instantiating arithmetic variables, but this does not help because there exist an infinite number of ground instances.

The term sharing is maintained except in the exponent part of the term (only the variable(s) are shared). For example if we consider the terms `$iterm(s(@),0)^{N1}`, `$iterm(s(@),0)^{N2}`, `$iterm(s(@),0)^{N1}` and `$iterm(s(@),0)^{2.N1}` the splay tree will be the following one:

Fig. 1. The splay tree

4 A Short Example

We show DEI at work on the following set of clauses:

$$\{p(f(x_1, g(x_2, \diamond))^n, s(\diamond)^n) \vee \\ \neg p'(x_3), p'(a), q(g(x_1, f(x_2, \diamond))^n.g(x_3, x_2), s(\diamond)^m(0)) \vee \\ \neg q'(x_3), q'(b), r(g(y, x), z) \vee \neg p(x, z), \neg q(x, y) \vee \neg r(x, y)\}.$$

This little example has been constructed to illustrate the possibilities of the language (contexts containing variables, shared arithmetic variables etc.). The reader can check that it is unsatisfiable. DEI constructs the following refutation (due to space restriction we use the lowest level of verbosity):

```
#p1(a) <- .
#
#q1(b) <- .
#
#r(g(X1,X2),X3) <- p(X2,X3).
#
# <- r(X1,X2), q(X1,X2).
#
#p($iterm(f(X1,g(X2,@)),X3)^{0+1.X4},$iterm(s(@),0)^{0+1.X4}) <- p1(X3).
#
#p($iterm(f(X2,g(X3,@)),a)^{0+1.X1},$iterm(s(@),0)^{0+1.X1}) <- .
#
#r(g(X5,$iterm(f(X2,g(X3,@)),a)^{0+1.X1}),$iterm(s(@),0)^{0+1.X1}) <- .
#
#q($iterm(g(X1,f(X2,@)),g(X3,X2))^{0+1.X4},$iterm(s(@),0)^{0+1.X4})
<- q1(X3).
#
#q($iterm(g(X2,f(X3,@)),g(b,X3))^{0+1.X1},$iterm(s(@),0)^{0+1.X1}) <- .
#
```

```
#r(g(X6,$iterm(f(X2,g(X3,@)),X4)^{0+1.X1}),$iterm(s(@),0)^{0+1.X1})
<- p1(X4).
#
# <- r($iterm(g(X2,f(X3,@)),g(b,X3))^{0+1.X1},$iterm(s(@),0)^{0+1.X1}).
```

Of course, I-terms can be "encoded" into first-order logic (by adding the semantic axioms in Section 2 in the clause set). However, a system as Dei with built-in I-terms handling allows one to encompass some deductive steps in the unification algorithm, which reduces the length of the proofs, improves the readability and the termination behavior. I-terms are especially useful for satisfiability detection.

5 Future Work

Future work includes the extension of Dei to more expressive term schematisation languages (such as the primal grammars [4] or the terms with several holes [7]) and the adaptation to these languages of the reasoning techniques that are commonly used by successful deduction systems (in particular the indexing techniques). We are presently working on an extension of the discrimination trees handling I-terms. Designing a superposition calculus that is complete on I-clauses is a problem that also deserves to be investigated.

In order to fully benefit of the expressive power of I-terms, we also plan to implement additional inference rules to generate automatically I-terms from standard clauses (as in [7,6]). The use of inductive reasoning techniques (in connection with the system presented in [1]) will also be investigated.

References

1. Bensaid, H., Caferra, R., and Peltier, N. Towards systematic analysis of theorem provers search spaces: First steps. In *Proc. Wollic'07 (Workshop on Logic, Language, Information and Computation)* (July 2007), Springer, pp. 38–52. LNCS 4576.
2. Chen, H., Hsiang, J., and Kong, H. On finite representations of infinite sequences of terms. In *Conditional and Typed Rewriting Systems, 2nd International Workshop* (1990), Springer, LNCS 516, pp. 100–114.
3. Comon, H. On unification of terms with integer exponents. *Mathematical System Theory 28* (1995), 67–88.
4. Hermann, M., and Galbavý, R. Unification of Infinite Sets of Terms schematized by Primal Grammars. *Theoretical Computer Science 176*, 1–2 (1997), 111–158.
5. Loechner, V. Polylib: A library for manipulating parametrized polyhedra. Tech. rep., ICPS, Universite Louis Pasteur de Strasbourg, 1999.
6. Peltier, N. A General Method for Using Terms Schematizations in Automated Deduction. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)* (2001), Springer LNCS 2083, pp. 578–593.
7. Salzer, G. The unification of infinite sets of terms and its applications. In *Logic Programming and Automated Reasoning (LPAR'92)* (July 1992), Springer, LNAI 624, pp. 409–429.
8. Schulz, S. System Description: E 0.81. In *Proc. of the 2nd IJCAR, Cork, Ireland* (2004), D. Basin and M. Rusinowitch, Eds., vol. 3097 of *LNAI*, Springer, pp. 223–228.