Proving Termination of Rewriting with C*i*ME

Evelyne Contejean, Claude Marché, Benjamin Monate and Xavier Urbain

Laboratoire de Recherche en Informatique (LRI) CNRS UMR 8623
Bât. 490, Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
E-mail: {contejea,marche,monate,urbain}@lri.fr

1 Introduction

The various and successful applications of rewriting has brought the need for automated manipulation of TRSs, thus of rewriting tools. Amongst those is the C*i*ME system [2].

C*i*ME is a toolbox which may be used for dealing with matching and unification of strings or terms modulo equational theories (A, AC, ...), tree automata, parameterized string rewriting [4], etc. Its capabilities w.r.t. TRSs include confluence checking, completion; it is also able to find (AC-)termination proofs with full automation, thanks to its termination experts and its efficient constraint solver over finite domains.

2 Proving Termination

A termination proof search in C*i*ME begins with computation of termination constraints. These may be checked by ACRPO [5] or translated into Diophantine constraints in order to obtain polynomial interpretations. In that last case, the finite domain constraint solver of C*i*ME tries to find a solution. Our policy is to provide a tool for proving termination of the TRS one meets *in practice*. Hence, we focus on modular/incremental proofs, a case for which the search for termination preserved under non-deterministic collapse is a particularly important issue.

Basic Features. Several criteria may be used in C*i*ME: The standard one (all rules strictly decrease) but also *dependency pairs* criteria [1], with or without marking of symbols, and with or without dependency graphs. Our extensions of dependency pairs to the AC case are also implemented [3].

We use two kinds of cycle analysis for the dependency graph refinement: one which treats all strongly connected components (and which is, thus, very efficient) and one which treats all strongly connected parts of the graph; the latter being more powerful but of higher complexity than the former.

The search for polynomial interpretations may be parameterized by the kind of polynomial to restrict to (linear, simple ou simple-mixed) and by the bound of their coefficients. In any case, AC-compatible interpretations will be used for AC-symbols.

Restricting to linear polynomials leads to fewer and easier (smaller) constraints than restricting to more complex polynomials, but at the cost of some termination power. Similarly, restricting to very small coefficients leads to faster constraint solving but might be not enough to find a suitable ordering.

Thus, in order to deal with TRS with numerous rules that are common in practice, and so as to make constraints as weak as possible, C*i*ME makes use of powerful modular and incremental criteria.

Modularity The algorithms at work in C*i*ME involve powerful methods so as to discover proofs of $\mathcal{CE}$-termination in an incremental and modular fashion [6,7]. This *divide*

and conquer approach significantly weakens constraints over orderings and decreases their number.

TRSs may be defined in C*i*ME as hierarchies of rewriting modules, we denote those as Hierarchical TRS (HTRS for short), and termination proofs are performed incrementally/modularly on modules constituting those extensions. In particular $\mathcal{CE}$-termination of a module R_i is proven *only once*, and never again in a termination proof of any extension of R_i. The function for searching an incremental/modular termination proof of a HTRS is `h_termination`. It tries to find a termination proof of the rewrite system given as argument by providing, for each sub-hierarchy (w.r.t. topological sorting), a suitable ordering.

Optimisations such as symbol marking and dependency graphs may be used as they affect both C*i*ME incremental/modular and classical termination experts.

In order to boost the efficiency of the termination expert, it may prove useful to restrict to nonexpensive criteria on most of the incremental proof.

For instance, when searching for a proof using polynomial interpretations, such tuning may be done with help of function `h_termination_with` which takes as first argument a list of pairs *criterion, bound*. They denote the kind of polynomials and the bounds to be tried successively on a module when a search fails.

Finally, a HTRS may be considered as a hierarchy of minimal modules. HTRS are then split up in minimal modules for termination proofs, which is particularly useful when dealing with a huge bunch of rules in a single HTRS.

3 Example

The following TRS describes natural numbers in binary notation and multisets and computes the sum of all numbers in a multiset. The termination proof using C*i*ME is done as follows. Firstly we enter variables, the signature and the TRS into the C*i*ME system:

```
CiME> let X = vars "x y z l b";
CiME> let F = signature "
    #, empty   : constant ; 0,1 : postfix unary ;
    singl, sum : unary ;
    U : AC ;  + : infix binary ;";
CiME> let R = HTRS {} F X "
  (#)0 -> #;      # + x -> x;      x + # -> x;
  (x)0 + (y)0 -> (x+y)0;  (x)0 + (y)1 -> (x+y)1;
  (x)1 + (y)0 -> (x+y)1;  (x)1 + (y)1 -> (x+y+(#)1)0;
  (x + y) + z -> x + (y + z);
  empty U b -> b;         sum(empty) -> (#)0;
  sum(singl(x)) -> x;     sum(x U y) -> sum(x) + sum(y);";
```

(Note the AC operator `U`.) Then we may choose minimal decomposition of `R` and an incremental/modular proof using linear polynomials with coefficients in $[0, 1]$ or simple polynomials with coefficients in $[0, 2]$ if no suitable linear interpretation is found.

```
CiME> termcrit "minimal";
CiME> h_termination_with {("linear",1);("simple",2)} R;
```

In less than a second, the hierarchy is splitted up in 8 modules (6 of which yielding *no constraint*) and we obtain interpretations for the remaining 2, namely for module

$$\begin{cases} \#+x \to x \qquad x+\# \to x \qquad (x+y)+z \to x+(y+z) \\ (x)0+(y)0 \to (x+y)0 \quad (x)0+(y)1 \to (x+y)1 \\ (x)1+(y)0 \to (x+y)1 \quad (x)1+(y)1 \to ((x+y)+(\#)1)0 \end{cases}$$

the tool proposes:

```
 checking each of the 1 strongly connected components :
 checking component 1 (disjunction of 1 constraints)
[#] = 0;
[0](X0) = X0 + 1;
[1](X0) = 2*X0 + 2;
[+](X0,X1) = X1*X0 + X1 + 2*X0 + 1;
['+'](X0,X1) = X1*X0 + 2*X0;
```

where `'+'` is the marked copy of `+`, and for module

$$\begin{cases} sum(singl(x)) \to x \qquad sum(empty) \to (\#)0 \\ sum(x \cup y) \to sum(x)+sum(y) \end{cases}$$

```
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 0;
[0](X0) = 0;
[1](X0) = 0;
[+](X0,X1) = X1 + X0;
[empty] = 0;
[singl](X0) = X0;
[U](X0,X1) = X1 + X0 + 1;
[sum](X0) = X0;
['sum'](X0) = X0;
Termination proof found.
```

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2, 2000. Available at `http://cime.lri.fr/`.
3. K. Kusakari, C. Marché, and X. Urbain. Termination of associative-commutative rewriting using dependency pairs criteria. Research Report 1304, LRI, 2002.
4. B. Monate. *Propriétés uniformes de familles de systèmes de réécriture de mots paramétrées par des entiers*. Thèse de doctorat, Université Paris-Sud, Orsay, France, Jan. 2002.
5. A. Rubio. A fully syntactic AC-RPO. In P. Narendran and M. Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, Trento, Italy, July 1999. Springer-Verlag.
6. X. Urbain. Automated Incremental Termination Proofs for Hierarchically defined Terms Rewriting Systems. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, Siena, Italy, june 2001. Springer-Verlag.
7. X. Urbain. Modular and incremental automated termination proofs. To appear in the *Journal of Automated Reasoning*, 2003.