Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 117 (2005) 263–284
www.elsevier.com/locate/entcs

New Evaluation Commands for Maude Within Full Maude[1]

Francisco Durán[a] Santiago Escobar[b] Salvador Lucas[b]

[a] *LCC, Universidad de Málaga, Campus de Teatinos, Málaga, Spain.* `duran@lcc.uma.es`

[b] *DSIC, UPV, Camino de Vera s/n, 46022 Valencia, Spain.* `{sescobar,slucas}@dsic.upv.es`

Abstract

Maude is able to deal with infinite data structures and avoid infinite computations by using *strategy annotations*. However, they can eventually make the computation of the normal form(s) of some input expressions impossible. We have used Full Maude to implement two new commands `norm` and `eval` which furnish Maude with the ability to compute (constructor) normal forms of initial expressions even when the use of strategy annotations together with the built-in computation strategy of Maude is not able to obtain them. These commands have been integrated into Full Maude, making them available inside the programming environment like any other of its commands.

Keywords: Declarative programming, Maude, reflection, strategy annotations.

1 Introduction

The ability of dealing with infinite objects is typical of lazy (functional) languages. Although the reductions in Maude are basically *innermost* (or eager), Maude is able to exhibit a similar behavior by using *strategy annotations* (see [17]). Maude strategy annotations are lists of non-negative integers associated to function symbols which specify the ordering in which the arguments are (eventually) evaluated in function calls: when considering a function call $f(t_1, \ldots, t_k)$, only the arguments whose indices are present as *positive* integers in the local strategy $(i_1 \cdots i_n)$ for f are evaluated (following the specified

[1] Work partially supported by CICYT TIC2001-2705-C03-01 and TIC2001-2705-C03-02, MCyT Acción Integrada HU 2003-0003, Agencia Valenciana de Ciencia y Tecnología GR03/025, and EU-India Cross-Cultural Dissemination project ALA/95/23/2003/077-054.

doi:10.1016/j.entcs.2004.06.014

ordering). If 0 is found, a reduction step on the whole term $f(t_1, \ldots, t_k)$ is attempted. The introduction of 'true' *replacement restrictions* (i.e., that forbid reductions on some arguments) is often sufficient to ensure (and even prove) a terminating behavior of a Maude program even though some expressions that denote an infinite object are involved (see [18] for an overview of methods to formally prove termination in these cases).

Full Maude is a language extension of Maude written in Maude itself, that endows Maude with notation for object-oriented programming and with a powerful and extensible module algebra in which Maude modules can be combined together to build more complex modules [9,10,4]. Every Maude module can be loaded in Full Maude by just enclosing it into parentheses. Then, the usual evaluation commands of Maude (e.g., `reduce`, `rewrite`, etc.) are available in Full Maude by also enclosing them into parentheses.

As a first example, let us consider the following parameterized module `LAZY-LIST` with a 'polymorphic' sort `List(X)`, and symbols `nil` (the empty list) and `_._` for the construction of polymorphic lists.

```
(fth TRIV is
   sort Elt .
 endfth)

(fmod LAZY-LIST(X :: TRIV) is
   protecting INT .
   sort List(X) .
   subsort X@Elt < List(X) .
   op nil : -> List(X) [ctor] .
   op _._ : X@Elt List(X) -> List(X) [ctor strat (1 0)] .
   op take : Int List(X) -> List(X) .
   var N : Int .   var X : X@Elt .   var Z : List(X) .
   eq take(0, Z) = nil .
   eq take(N, X . Z) = X . take(N - 1, Z) .
 endfm)
```

Note the strategy `(1 0)` associated to the operator `_._`, which forbids replacements on its second argument. Given a term of the form `X . L`, the strategy indicates that its first argument, the subterm `X`, will first be reduced, and then a reduction step on the whole term would be attempted. The `LAZY-LIST` module also includes a typical polymorphic operator `take` which selects the first n components of a list. Even though `take` has no explicit strategy annotation, Maude internally assigns a *by default* one `(1 2 0)`. In fact, Maude gives a strategy annotation $(1\ 2\ \cdots k\ 0)$ to each symbol f without an explicit strategy annotation.

The instantiation of the formal parameters of a parameterized module with actual parameter modules requires the use of *views*, which provide the interpretation of the actual parameters. Given a view `Nat` from the functional theory `TRIV` to the predefined module `NAT`, we may then define a function `natsFrom`, which is able to generate the infinite list of natural numbers, as