OnDemandOBJ: A Laboratory for Strategy Annotations

María Alpuente\textsuperscript{2} Santiago Escobar\textsuperscript{3} Salvador Lucas\textsuperscript{4}

DSIC, Universidad Politécnica de Valencia, Spain

Abstract

Strategy annotations are used in rule-based programming languages such as OBJ2, OBJ3, CafeOBJ, and Maude to improve efficiency and/or reduce the risk of nontermination. Syntactically, they are given either as lists of natural numbers or as lists of integers associated to function symbols whose (absolute) values refer to the arguments of the corresponding symbol. A positive index forces the evaluation of an argument whereas a negative index means “evaluate on-demand”. In this paper, we present OnDemandOBJ, an implementation of strategy-guided on-demand evaluation, which improves previous mechanizations that were lacking satisfactory computational properties.

1 Introduction

Eager rule-based programming languages such as Lisp, OBJ*, CafeOBJ, ELAN, or Maude evaluate functional expressions by innermost rewriting. Since nontermination is a known problem of innermost reduction, syntactic annotations (generally specified as sequences of integers associated to function arguments, called local strategies) have been used in OBJ2 \cite{9}, OBJ3 \cite{11}, CafeOBJ \cite{10}, and Maude \cite{6} to improve efficiency and (hopefully) avoid nontermination. A local strategy for a \textit{k}-ary symbol \(f \in \mathcal{F}\) is a sequence \(\varphi(f)\) of integers taken from \((-k,\ldots,-1,0,1,\ldots,k)\) which are given in parentheses. Local strategies are used in OBJ programs\textsuperscript{5} for guiding the evaluation strategy (abbr. E-strategy): when considering a function call \(f(t_1,\ldots,t_k)\), if annotation \(i\) appears in the local strategy, then the subterm at argument \(i\) is evaluated. If 0

\textsuperscript{1} Work partially supported by CICYT TIC2001-2705-C03-01 and MCYT grants HA2001-0059 and HU2001-0019.  
\textsuperscript{2} Email:alpuente@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/alpuente.html  
\textsuperscript{3} Email:sescobar@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/sescobar.html  
\textsuperscript{4} Email:slucas@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/slucas.html  
\textsuperscript{5} As in \cite{11}, by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.
is found, then the evaluation of \( f \) is attempted. A mapping \( \varphi \) that associates a local strategy \( \varphi(f) \) to every \( f \in F \) is called an \( E \)-strategy map [17,18]. Whenever the user provides no local strategy for a given symbol, the (Maude, OBJ*, CafeOBJ) interpreter automatically assigns a default \( E \)-strategy. We adopt the default local strategy of Maude which associates the local strategy \( (1 2 \cdots k 0) \) to each \( k \)-ary symbol \( f \) having no explicit strategy, i.e. all arguments are marked as evaluable.

**Example 1.1** Consider the following Maude program \( \pi \) which codifies the well-known infinite series expansion to approximate number \( \pi \):

\[
\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots
\]

```maude
obj PI is
  sorts Nat LNat Recip LRecip .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op posrecip : Nat -> Recip .
  op negrecip : Nat -> Recip .
  op nil : -> LNat .
  op cons : Nat LNat -> LNat .
  op rnil : -> LRecip .
  op rcons : Recip LRecip -> LRecip .
  op from : Nat -> LNat .
  op seriepos : Nat LNat -> LRecip .
  op serieneg : Nat LNat -> LRecip .
  op pi : Nat -> LRecip .
vars N X Y : Nat . var Z : LNat .
eq from(X) = cons(X,from(s(X))) .
eq seriepos(0,Z) = rnil .
eq seriepos(s(N),cons(X,cons(Y,Z))) = rcons(posrecip(Y),serieneg(N,Z)) .
eq serieneg(0,Z) = rnil .
eq serieneg(s(N),cons(X,cons(Y,Z))) = rcons(negrecip(Y),seriepos(N,Z)) .
eq pi(X) = seriepos(X,from(0)) .
endo
```

A term\(^6\) \( \pi(2) \) approximates the number \( \pi/4 \) using 2 elements of the series expansion, i.e. the intended behavior is

\[
\pi(2) \rightarrow^* rcons(posrecip(1),rcons(negrecip(3),rnil))
\]

where \( \text{posrecip}(n) \) denotes the positive reciprocal \( 1/n \) and \( \text{negrecip}(n) \) denotes \( -1/n \). Note that since the Maude interpreter associates a default local strategy \( (1 2 \cdots k 0) \) to each \( k \)-ary symbol \( f \) having no explicit strategy, all arguments are marked as evaluable. Therefore, this program is non-terminating since innermost evaluation diverges:

\( ^6 \) Naturals \( 1, 2, \ldots \) are used as shorthand to numbers \( s^n(0) \) where \( n = 1, 2, \ldots \).
\[ \pi(2) \rightarrow \text{seriepos}(2, \text{from}(0)) \]
\[ \rightarrow \text{seriepos}(2, \text{cons}(0, \text{from}(1))) \]
\[ \rightarrow \text{seriepos}(2, \text{cons}(0, \text{cons}(1, \text{from}(2)))) \rightarrow \cdots \]

In order to avoid non-termination, annotation 2 should be removed from the (default) local strategy \((1 \ 2 \ 0)\) for symbol \texttt{cons}.

**Example 1.2** After removing annotation 2 from the local strategy for symbol \texttt{cons} in Example 1.1, the program becomes terminating under innermost rewriting with such restriction. The unique change in the program of Example 1.1 is:

\[ \text{op cons : Nat LNat -> LNat [strat (1)] .} \]

Unfortunately, this restriction of rewriting has a negative impact in the ability to compute normal forms as shown in the following example.

**Example 1.3** The evaluation of \(\pi(2)\) using the program of Example 1.3 yields the following sequence:

\[ \pi(2) \rightarrow \text{seriepos}(2, \text{from}(0)) \rightarrow \text{seriepos}(2,\text{cons}(0, \text{from}(1))) \]

The evaluation stops at this point since reductions on the second argument of \texttt{cons} are disallowed. Indeed, note that a further step

\[ \text{seriepos}(2, \text{cons}(0, \text{from}(1))) \rightarrow \text{seriepos}(2, \text{cons}(0, \text{cons}(1, \text{from}(2)))) \]

is required in order to apply the second rule of \texttt{seriepos} and be able to obtain the intended normal form of Example 1.1.

The handicaps of using only positive annotations regarding correctness and completeness of computations are discussed in [1,2,13,15,18,19]: essentially, the problem is that the absence of some indices in the local strategies can have a negative impact in the ability of such strategies to compute normal forms.

In [18,19], *negative* indices are proposed to indicate those arguments that should be evaluated only ‘on-demand’, where the ‘demand’ is an attempt to match an argument term with the left-hand side of a rewrite rule [7,11,19]. For instance, subterm \texttt{from}(1) in Example 1.3 is *demanded* by the second rule of \texttt{seriepos}. Thus, \((1 \ -2)\) would be the apt local strategy for \texttt{cons} as pointed out in [18]; i.e. the first argument is always evaluated but the second argument is evaluated only “on-demand”. Then, the evaluation of the symbol \texttt{cons} under strategy \((1 \ -2)\) is able to normalize \(\pi(2)\) to its intended normal form without entering in a non-terminating evaluation, whereas evaluation only with positive annotations enters an infinite derivation (as shown in Example 1.1) or does not provide the intended normal form (as shown in Example 1.2).

However, *on-demand* strategy annotations have not been implemented to date: even if negative annotations are (syntactically) accepted in current OBJ implementations, namely OBJ3 and Maude, unfortunately they do not have the expected (on-demand) effect over the computations.
Example 1.4 Consider the program of Example 1.1 where the local strategy for \texttt{cons} includes the on-demand annotation -2. The unique change to the program is:

\[
\text{op cons : Nat LNat -> LNat [strat (1 -2)] .}
\]

The OBJ3 interpreter does not implement negative (on-demand) annotations though does accept this program and the evaluation of \(\pi(2)\) surprisingly delivers the very same result as in Example 1.2. That is, the negative annotation is just disregarded by the OBJ3 interpreter (which, in this case, causes loss of completeness). On the other hand, the Maude interpreter neither implements negative annotations but also accepts this program and the evaluation of the same expression diverges as in Example 1.1. This is because the negative annotation -2 is interpreted as a positive one thus resulting in non-termination.

On the other hand, CafeOBJ is able to deal with negative annotations using the on-demand evaluation model of [18] and is able to compute the intended value \(\text{rcons}(	ext{posrecip}(1), \text{rcons}(	ext{negrecip}(3), \text{rnil}))\) of Example 1.1. However, in [1] we discussed a number of problems of the on-demand evaluation model of [18,19], as shown in the following example.

Example 1.5 [1] Consider the following OBJ program:

\[
\text{obj LENGTH is}
\]

\[
\begin{align*}
\text{sorts Nat LNat .} \\
\text{op 0 : -> Nat .} \\
\text{op s : Nat -> Nat .} \\
\text{op nil : -> LNat .} \\
\text{op cons : Nat LNat -> LNat [strat (1)] .} \\
\text{op from : Nat -> LNat .} \\
\text{op length : LNat -> Nat [strat (0)] .} \\
\text{op length' : LNat -> Nat [strat (-1 0)] .} \\
\text{vars X Y : Nat . var Z : LNat .} \\
\text{eq from(X) = cons(X,from(s(X))) .} \\
\text{eq length(nil) = 0 .} \\
\text{eq length(cons(X,Z)) = s(length'(Z)) .} \\
\text{eq length'(Z) = length(Z) .} \\
\end{align*}
\]

\text{endo}

When considering the expression \(\text{length'}(\text{from}(0))\), this expression is rewritten (in one step) to the expression \(\text{length}(\text{from}(0))\). No evaluation is demanded on the argument of \(\text{length'}\) for enabling this step (the negative annotation -1 is included for \(\text{length'}\) but the corresponding rule includes a variable at the first argument of \(\text{length'}\)) and no further evaluation on \(\text{length}(\text{from}(0))\) should be performed (due to the local strategy (0) of \(\text{length}\) which forbids evaluation on any argument of \(\text{length}\)). However, the annotation -1 of function \(\text{length'}\) is treated in such a way by the operational model of [19,18] that the on-demand evaluation of the expression \(\text{length'}(\text{from}(0))\) yields an infinite evaluation sequence (see [1] for a more detailed explanation).
In [1] we proposed a solution to these problems in order to cope with on-demand strategy annotations, which is based on a suitable extension of the \(E\)-evaluation strategy of OBJ-like languages which only considers annotations given as natural numbers. Our strategy incorporates a better treatment of demandness and also enjoys good computational properties; in particular, we show how it can be used for computing (head-)normal forms and we prove it is conservative w.r.t. other on-demand strategies: lazy rewriting [8] and on-demand rewriting [13]. A program transformation for proving termination of the on-demand evaluation strategy was also formalized, which relies on standard techniques.

In this paper, we address the implementation of on-demand evaluation strategy of [1] together with different techniques related to managing on-demand strategy annotations. This system is called OnDemandOBJ.

2 OnDemandOBJ

In order to demonstrate the practicality of our ideas, an interpreter of the computational model described in [1] has been implemented in Haskell (using GHC 5.04.2). The system is called OnDemandOBJ and is publicly available at http://www.dsic.upv.es/users/elp/soft.html

2.1 Programs

The prototype implements a subset of the Maude and CafeOBJ syntax, i.e. admits programs typed in either one of the two syntaxes. The BNF grammars associated to such syntax subsets are included in the distribution. Default strategy annotations are considered as in Maude, i.e. the default local strategy associated to a \(k\)-ary symbol \(f\), is \((1 \ 2 \ldots \ k \ 0)\). The prototype does not provides a prelude set of functions or operators as in Maude or CafeOBJ, i.e. if\_then\_else\_function is not directly available.

2.2 Evaluation

The evaluation of an expression according to the computational model described in [7,17] is available through the command \texttt{red}. This command is also available in OBJ2, OBJ3, CafeOBJ, or Maude.

If reductions on some arguments are constrained by means of strategy annotations, command \texttt{red} can fail to obtain the desired normal forms. Expressions obtained by \texttt{red} are called \(E\)-normal forms. Conditions ensuring that \(E\)-normal forms are (at least) head-normal forms have been investigated in [13,18]. In order to be able to obtain normal forms once head-normal forms are obtained, the OnDemandOBJ prototype provides a novel command \texttt{norm} which calculates normal forms following the normalization via \(\mu\)-normalization process described in [14]. Informally speaking, once the \(E\)-normal forms have
been obtained, the evaluation process starts on those positions which were not allowed for reduction. The following example explains how this normalization process works.

**Example 2.1** Consider the problem of selecting a collection of prime numbers. The following program codifies such problem where the expression primes is intended to arbitrarily approximate the list of prime numbers (see [12]).

```maude
obj SEL-FIRST-PRIMES is
  sorts Nat LNat .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  ops nil serieprimes : -> LNat .
  op cons : Nat LNat -> LNat [strat (1)] .
  op first : Nat LNat -> LNat .
  op nats primes : Nat -> LNat .
  op sieve : LNat -> LNat .
  op filter : LNat Nat Nat -> LNat .
vars X Y M N : Nat . var Z : LNat .
  eq filter(cons(X,Z),0,M) = cons(0,filter(Z,M,M)) .
  eq filter(cons(X,Z),s(N),M) = cons(X,filter(Z,N,M)) .
  eq sieve(cons(0,Z)) = sieve(Z) .
  eq sieve(cons(s(N),Z)) = cons(s(N),sieve(filter(Z,N,N))) .
  eq nats(N) = cons(N,nats(s(N))) .
  eq serieprimes = sieve(nats(s(s(0)))) .
  eq first(0,Z) = nil .
  eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
  eq primes(N) = first(N,serieprimes) .
endo
```

The intended behavior is \( \text{primes}(3) \rightarrow^* \text{cons}(2,\text{cons}(3,\text{cons}(5,nil))) \). Note that in order to avoid non-termination, the strategy for symbol cons does not include annotation 2, as in Example 1.2. Therefore, the program is not complete and some normalizations are not available. For instance, expression primes(3) is evaluated as follows:\(^7\):

```maude
Maude> red primes(s(s(s(0))))).
reduce in SEL-FIRST-PRIMES : primes(s(s(s(0)))) .
  rewrites: 5 in -1ms cpu (0ms real) (~ rewrites/second)
result LNat: cons(s(s(0)),first(s(s(0))),...)
```

where the expression\(^8\) \( \text{cons}(2,\text{first}(2,...)) \) is obtained instead of the right expression \( \text{cons}(2,\text{cons}(3,\text{cons}(5,nil))) \). Note that annotation -2 for symbol cons does not solve this problem since the second argument of cons is a variable in every lhs of the program.

---

\(^7\) We use the SRI’s Maude interpreter (version 1.0.5) available at: [http://maude.csl.sri.com/system/](http://maude.csl.sri.com/system/).

\(^8\) This expression has been shorten since only the defined symbol first at position 2 is relevant.
However, when the command \texttt{norm} is used, the evaluation is restarted on the maximal non-evaluated subterms in order to produce an actual normal form, i.e. on subterm \texttt{first(s(s(0))),...}. For instance, when the previous expression is evaluated using OnDemandOBJ, we obtain the right expression:

\begin{verbatim}
SEL-FIRST-PRIMES> norm primes(s(s(s(0))).
Normal form: cons(s(s(0)),cons(s(s(s(0))),cons(s(s(s(s(s(0))))),nil)))
{ 0.0000 sec., 29 rewrites }
\end{verbatim}

2.3 Transformations

In the following, we recall two program transformations integrated into OnDemandOBJ.

2.3.1 Removing negative annotations

In [3] we introduced an automatic, semantics–preserving program transformation which produces a program (without negative annotations) which can be then correctly executed by typical OBJ interpreters. The idea is to encode the ‘on-demand’ strategy instrumented by the negative annotations within new function symbols (and corresponding program rules) that only use positive strategy annotations. Command \texttt{trNeg} of the OnDemandOBJ prototype applies this program transformation to eliminate on-demand annotations (negative indices) from an annotated program (see [3]) and then loads the new transformed program for evaluation.

The following example explains how this program transformation works.

Example 2.2 Consider the program of Example 1.4. The program obtained after the transformation of [3] is the following:

\begin{verbatim}
obj PINoNeg is
  sorts Nat LNat Recip LRecip .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op posrecip : Nat -> Recip .
  op negrecip : Nat -> Recip .
  op nil : -> LNat .
  op cons : Nat LNat -> LNat [strat (0)] .
  op cons+-2 : Nat LNat -> LNat [strat (2)] .
  op rnil : -> LRecip .
  op rcons : Recip LRecip -> LRecip .
  op from : Nat -> LNat .
  op seriepos : Nat LNat -> LRecip .
  op seriepos+-2 : Nat LNat -> LRecip [strat (2 0)] .
  op serieneg : Nat LNat -> LRecip .
  op serieneg+-2 : Nat LNat -> LRecip [strat (2 0)] .
  op pi : Nat -> LRecip .
  op quoteNat : Nat -> Nat [strat (0)] .
  op quoteLNat : LNat -> LNat [strat (0)] .
  op quoteRecip : Recip -> Recip [strat (0)] .
\end{verbatim}
op quoteLRecip : LRecip -> LRecip [strat (0)] .
vars N X Y : Nat . var Z : LNat . var R : Recip . var L : LRecip .
eq from(X) = quoteLNat(cons(X,from(s(X)))) .
eq seriepos(0,Z) = quoteLRecip(rnil) .
eq seriepos(s(N),cons(X,Z)) = seriepos++2(s(N),cons++2(X,Z)) .
eq seriepos++2(s(N),cons++2(X,cons(Y,Z))) =
quoteLRecip(rcons(seriepos(rposrecip(Y),seriepos(N,Z)))) .
eq serieneg(0,Z) = quoteLRecip(rnil) .
eq serieneg(s(N),cons(X,Z)) = serieneg++2(s(N),cons++2(X,Z)) .
eq serieneg++2(s(N),cons++2(X,cons(Y,Z))) =
quoteLRecip(rcons(negrecip(Y),seriepos(N,Z))) .
eq pi(X) = quoteLRecip(seriepos(X,from(0))) .
eq quoteNat(0) = 0 .
eq quoteNat(s(N)) = s(N) .
eq quoteRecip(posrecip(N)) = posrecip(N) .
eq quoteRecip(negrecip(N)) = negrecip(N) .
eq quoteLNat(nil) = nil .
eq quoteLNat(cons(X,Z)) = cons(quoteNat(X),Z) .
eq quoteLNat(from(X)) = from(X) .
eq quoteLRecip(rnil) = rnil .
eq quoteLRecip(rcons(R,L)) = rcons(quoteRecip(R),quoteLRecip(L)) .
eq quoteLRecip(seriepos(X,Z)) = seriepos(quoteNat(X),quoteLNat(Z)) .
eq quoteLRecip(serieneg(X,Z)) = serieneg(quoteNat(X),quoteLNat(Z)) .
eq quoteLRecip(pi(X)) = pi(X) .
endo

Informally, new symbols seriepos++2, serieneg++2 and cons++2 are introduced to enable the evaluation of the second argument of cons in those positions which could be eventually evaluated on-demand. Note that the rules for symbols seriepos and serieneg in Example 1.4 are the only one which could demand the evaluation of the second argument of cons. The extra symbols quote are introduced to preserve correctness w.r.t. reductions with positive indices. Now, term pi(2) is correctly evaluable using the Maude interpreter (which simulates the on-demand evaluation of [1])

Maude> red quoteLRecip(pi(s(s(0)))) .
reduce in PINoNeg : quoteLRecip(pi(s(s(0)))) .
rewrites: 33 in -1ms cpu (0ms real) (~ rewrites/second)
result LRecip: rcons(posrecip(s(0)),rcons(negrecip(s(s(s(0)))))), rnil)

In OnDemandOBJ, the execution of the program transformation and the reduction of term pi(2) works as follows:

PI> trNeg
Module PI successfully transformed
Module PINoNeg loaded
PINoNeg> red quoteLRecip(pi(s(s(0)))) .
Normal form: rcons(posrecip(s(0)),rcons(negrecip(s(s(s(0))))),rnil))
{ 0.0020 sec., 33 rewrites }
2.3.2 Ensuring constructor normal forms

In [2] we defined a program transformation methodology for (correct and) complete evaluations which applies to OBJ-like languages. We ascertain the conditions (on an strategy $\varphi$) ensuring that OBJ programs using strategy annotations do compute the value (i.e., the constructor normal form) of any given expression.

The following example explains how this program transformation works.

**Example 2.3** [2] Consider the following OBJ program:

```obj
obj EXAMPLE is
    sorts Nat LNat .
    op 0 : -> Nat .
    op s : Nat -> Nat .
    op nil : -> LNat .
    op cons : Nat LNat -> LNat [strat (1)] .
    op from : Nat -> LNat .
    op sel : Nat LNat -> Nat .
    op first : Nat LNat -> LNat .
    vars X Y : Nat . var Z : LNat .
    eq sel(s(X),cons(Y,Z)) = sel(X,Z) .
    eq sel(0,cons(X,Z)) = X .
    eq first(0,Z) = nil .
    eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
    eq from(X) = cons(X,from(s(X))) .
endo
```

The evaluation of expression $t = \text{first}(s(0),\text{from}(0))$ of sort LNat yields:

```
Maude> reduce first(s(0),from(0)) .
reduce in EXAMPLE : first(s(0), from(0)) .
rewrites: 2 in -10ms cpu (0ms real) (~ rewrites/second)
result LNat: cons(0, first(0, from(s(0))))
```

Note that $\text{cons}(0,\text{first}(0,\text{from}(s(0))))$ is not a normal form and differs from the expected normal form $\text{cons}(0,\text{nil})$. Indeed, this value cannot be obtained by using the Maude interpreter.

The application of the program transformation of [2] produces the following program:

```obj
obj EXAMPLE-TR is
    sorts Nat LNat .
    ops 0 0' : -> Nat .
    ops s s' : Nat -> Nat .
    ops nil nil' : -> LNat .
    op cons : Nat LNat -> LNat [strat (1)] .
    op cons' fcons : Nat LNat -> LNat .
    op from : Nat -> LNat .
    ops sel sel' : Nat LNat -> Nat .
    ops first first' : Nat LNat -> LNat .
    op quoteNat : Nat -> Nat [strat (0)] .
    op unquoteNat : Nat -> Nat .
    op quoteLNat : LNat -> LNat [strat (0)] .
```

op unquoteLNat : LNat -> LNat .
vars X Y : Nat . var Z : LNat .
eq sel(s(X),cons(Y,Z)) = sel(X,Z) .
eq sel(0,cons(X,Z)) = X .
eq first(0,Z) = nil .
eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
eq from(X) = cons(X,from(s(X))) .
eq sel'(s(X),cons(Y,Z)) = sel'(X,Z) .
eq sel'(0,cons(X,Z)) = quoteNat(X) .
eq first'(0,Z) = nil' .
eq first'(s(X),cons(Y,Z)) = cons'(quoteNat(Y),first'(X,Z)) .
eq quoteNat(0) = 0' .
eq quoteLNat(cons(X,Z)) = cons'(quoteNat(X),quoteLNat(Z)) .
eq quoteLNat(nil) = nil' .
eq quoteNat(s(X)) = s'(quoteNat(X)) .
eq quoteNat(sel(X,Z)) = sel'(X,Z) .
eq quoteLNat(first(X,Z)) = first'(X,Z) .
eq unquoteNat(0') = 0 .
eq unquoteNat(s'(X)) = s(unquoteNat(X)) .
eq unquoteLNat(nil') = nil .
eq unquoteLNat(cons'(X,Z)) = fcons(unquoteNat(X),unquoteLNat(Z)) .
eq fcons(X,Z) = cons(X,Z) .
endo

Informally, all constructors and defined symbols which participate in the sort LNat of the goal term are duplicated in order to enable reduction on the second argument of cons. Symbols quote translate from original symbols to duplicated symbols and symbol unquote translates back to original symbols. Now, the evaluation of unquoteLNat(quoteLNat(first(s(0),from(0)))) yields:

Maude> reduce unquoteLNat(quoteLNat(first(s(0),from(0)))) .
reduce in EXAMPLE-TR : unquoteLNat(quoteLNat(first(s(0),from(0)))) .
rewrites: 10 in -10ms cpu (0ms real) (~ rewrites/second)
result LNat: cons(0, nil)

This procedure is implemented in OnDemandOBJ by using the command eval which applies the transformation of [2] and automatically quotes/unquotes the input expression.

EXAMPLE> eval first(s(0),from(0)).
Normal form: cons(0, nil)
{ 0.0000 sec., 10 rewrites }

3 Experiments

Tables 1, 2, and 3 show the runtimes\(^9\) and the number of rewrite steps of a selection of benchmarks for the different OBJ-family systems. The source code

---

\(^9\) The average of 10 executions measured in a Pentium III 350 Mhz machine with 256 Mbytes running RedHat 7.2.
of the benchmarks can be found at Appendix A. These experimental results are also available at

http://www.dsic.upv.es/users/elp/ondemandOBJ/experiments

CafeOBJ is developed in Lisp at the Japan Advanced Inst. of Science and Technology (JAIST); OBJ3, also written in Lisp, is maintained by the University of California at San Diego; Maude is developed in C++ and maintained by the Computer Science Lab at SRI International. Moreover, OBJ3 and Maude provide only computations with positive annotations whereas CafeOBJ provides also computations with negative annotations using the on-demand evaluation of [18,19]. On the other hand, OnDemandOBJ computes with negative annotations using the on-demand evaluation strategy provided in [1]. Note that CafeOBJ and OBJ3 implement sharing of variables whereas Maude and OnDemandOBJ do not. It is worth noting that the mark overflow in Tables 2 and 3 indicates that the execution raised a memory overflow and normal form was not achieved whereas the mark unavailable in Tables 1 and 2 indicates that the program can not be executed in such OBJ implementation. Note that since Maude is implemented in C++, typical execution times are nearly 0 milliseconds.

The benchmark pi codifies the program of Example 1.4. It is worth noting that uses negative annotations to obtain a terminating and complete example, which can not be obtained by using only positive annotations. Termination of the program can be formally proved using the technique of [1] (see Appendix B below). Also, by using the results in [2], we can guarantee that every expression such as pi(n) for some n of sort Nat produces (as expected) a completely evaluated expression of sort LRecip. Table 1 compares the evaluation of the expression 10 pi(square(square(3))) using existing OBJ implementations. It demonstrates that negative annotations are actually useful in practice and that the implementation of the on-demand evaluation strategy in other systems is quite promising.

On the other hand, Table 2 illustrates the interest of using negative annotations to improve the behavior of programs: the benchmark msquare_eager codifies the functions square, minus, times, and plus over natural numbers using only positive annotations. This benchmark is called eager because every k-ary symbol f is given a strategy (1 2 ⋯ k 0) (this corresponds to default strategies in OnDemandOBJ). Note that the program is terminating as a TRS (i.e., without any strategy annotation). The benchmark msquare_apt is similar to msquare_eager, but canonical positive strategies are provided: the i-th argument of a symbol f is annotated with a positive index if there is an occurrence of f in the left-hand side of a rule having a non-variable i-th argument; otherwise, the argument is not annotated (see [4]). The benchmark msquare_neg is similar to msquare_apt, though canonical arbitrary strategies

10 Rules for function square are not included in Example 1.4 but can be found at Appendix A.
Table 1

<table>
<thead>
<tr>
<th>ms./rewrites</th>
<th>pi</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDemandOBJ</td>
<td>25/364</td>
</tr>
<tr>
<td>CafeOBJ</td>
<td>30/364</td>
</tr>
<tr>
<td>OBJ3</td>
<td>unavailable</td>
</tr>
<tr>
<td>Maude</td>
<td>unavailable</td>
</tr>
</tbody>
</table>

Execution of call \( \text{pi(square(square(3)))) \)

Table 2

<table>
<thead>
<tr>
<th>ms./rewrites</th>
<th>msquare_eager</th>
<th>msquare/apt</th>
<th>msquare_neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDemandOBJ</td>
<td>33/715</td>
<td>62/1640</td>
<td>0/1</td>
</tr>
<tr>
<td></td>
<td>40/914</td>
<td>78/1992</td>
<td>80/1992</td>
</tr>
<tr>
<td>CafeOBJ</td>
<td>40/715</td>
<td>50/715</td>
<td>0/1</td>
</tr>
<tr>
<td></td>
<td>50/914</td>
<td>60/914</td>
<td>60/914</td>
</tr>
<tr>
<td>OBJ3</td>
<td>20/715</td>
<td>overflow</td>
<td>unavailable</td>
</tr>
<tr>
<td></td>
<td>30/914</td>
<td>overflow</td>
<td>unavailable</td>
</tr>
<tr>
<td>Maude</td>
<td>0/715</td>
<td>0/1640</td>
<td>unavailable</td>
</tr>
<tr>
<td></td>
<td>0/914</td>
<td>3/1992</td>
<td>unavailable</td>
</tr>
</tbody>
</table>

Execution of terms \( \text{minus(0,square(square(5)))} \) and \( \text{minus(square(square(5)),square(square(3)))} \)

are provided: now (from left-to-right), the \( i \)-th argument of a defined symbol \( f \) is annotated with a positive index \( i \) if all occurrences of \( f \) in the left-hand side of the rules contain a non-variable \( i \)-th argument; if all occurrences of \( f \) in the left-hand side of the rules have a variable \( i \)-th argument, then the argument is not annotated; in any other case, annotation \(- i \) is given to \( f \) (see [4]). Then, for instance, program \( \text{msquare_neg} \) runs in less time and requires a smaller number of rewrite steps than \( \text{msquare_eager} \) or \( \text{msquare/apt} \), which do not include negative annotations. Note the difference in the number of rewrite steps of benchmarks \( \text{msquare_eager} \) and \( \text{msquare/apt} \) for the Maude and OnDemandOBJ systems, which is due to the absence of variable sharing.

Finally, Table 3 compares the execution of typical functional programs with canonical arbitrary strategies in OnDemandOBJ and in CafeOBJ, and demonstrates that there are clear advantages in using our implementation of the on-demand evaluation. We have used benchmarks \text{quicksort}, \text{minsort}, \text{mod}, and \text{average} which are borrowed from [5], and use canonical arbitrary strategies. Benchmark \text{mod'} is similar to \text{mod} but extra annotations are provided in order to avoid differences due to sharing (again, OnDemandOBJ does not implemented sharing of variables).
In this paper we have addressed the implementation of the on-demand evaluation strategy of [1] together with different techniques related to managing on-demand strategy annotations. For instance, we provide a novel command norm which calculates normal forms following the normalization via $\mu$-normalization process described in [14] (see Section 2.2) and two program transformations integrated into OnDemandOBJ. One program transformation for encoding the on-demand strategy instrumented by the negative annotations within new function symbols (and corresponding program rules) that only use positive strategy annotations (see Section 2.3.1) and the other program transformation for ensuring (correct and) complete evaluations within OBJ programs using strategy annotations to compute the value (i.e., the constructor normal form) of any given expression (see Section 2.3.2). This new features apply to OBJ-like languages in general.

References


A Benchmarks code

A.1 Program pi

This program codifies the well-known infinite series expansion to approximate the \( \pi \) number:

\[
\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots
\]

To make the program terminating and complete, the strategy for symbol cons must include annotation \(-2\). The rest of strategy annotations are positive since termination of the whole program can be proved. Note that the auxiliary functions plus, times and square for natural numbers are also included.

```
obj PI is
  sorts Nat LNat Recip LRecip .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op posrecip : Nat -> Recip [strat (1)] .
  op negrecip : Nat -> Recip [strat (1)] .
  op nil : -> LNat .
  op cons : Nat LNat -> LNat [strat (1 -2)] .
  op nnil : -> LRecip .
  op rcons : Recip LRecip -> LRecip [strat (1 2)] .
  op from : Nat -> LNat [strat (1 0)] .
  op 2ndspos : Nat LNat -> LRecip [strat (1 2 0)] .
  op 2ndsneg : Nat LNat -> LRecip [strat (1 2 0)] .
  op pi : Nat -> LRecip [strat (1 0)] .
  op plus : Nat Nat -> Nat [strat (1 2 0)] .
  op times : Nat Nat -> Nat [strat (1 2 0)] .
  op square : Nat -> Nat [strat (1 0)] .
vars N X Y : Nat . var Z : LNat .
eq from(X) = cons(X,from(s(X))) .
eq 2ndspos(0,Z) = nnil .
eq 2ndspos(s(N),cons(X,cons(Y,Z))) =
  rcons(posrecip(Y),2ndsneg(N,Z)) .
eq 2ndsneg(0,Z) = nnil .
eq 2ndsneg(s(N),cons(X,cons(Y,Z))) =
  rcons(negrecip(Y),2ndspos(N,Z)) .
eq pi(X) = 2ndspos(X,from(0)) .
eq plus(0,Y) = Y .
eq plus(s(X),Y) = s(plus(X,Y)) .
eq times(0,Y) = 0 .
eq times(s(X),Y) = plus(Y,times(X,Y)) .
eq square(X) = times(X,X) .
endo
```

A.2 Transformed program pi\_noneg

The application of the program transformation for removing negative annotations presented in this paper to the program pi produces the following program
(this is obtained automatically in our implementation). Note that annotation
(0) for symbol cons is necessary due to problems in Maude for representing
and interpreting an empty strategy.

obj PI4 is

sorts Nat LNat Recip LRecip .
op 0 : -> Nat .
op s : Nat -> Nat [strat (1)] .
op posrecip : Nat -> Recip [strat (1)] .
op negrecip : Nat -> Recip [strat (1)] .
op nil : -> LNat .
op cons : Nat LNat -> LNat [strat (0)] .
op rnil : -> LRecip .
op rcons : Recip LRecip -> LRecip [strat (1 2)] .
op from : Nat -> LNat [strat (1 0)] .
op 2ndspos : Nat LNat -> LRecip [strat (1 2 0)] .
op 2ndsneg : Nat LNat -> LRecip [strat (1 2 0)] .
op pi-4 : Nat -> LRecip [strat (1 0)] .
op plus : Nat Nat -> Nat [strat (1 2 0)] .
op times : Nat Nat -> Nat [strat (1 2 0)] .
op square : Nat -> Nat [strat (1 0)] .
op quoteNat : Nat -> Nat [strat (0)] .
op quoteLNat : LNat -> LNat [strat (0)] .
op quoteRecip : Recip -> Recip [strat (0)] .
op quoteLRecip : LRecip -> LRecip [strat (0)] .
op cons-root : Nat LNat -> LNat [strat (1 0)] .
op 2ndspos-+2 : Nat LNat -> LRecip [strat (2 0)] .
op 2ndsneg-+2 : Nat LNat -> LRecip [strat (2 0)] .
vars N X Y : Nat .
var W : Recip .
var V : LRecip .
eq from(N) = quoteLNat(cons(N,from(s(N)))) .
eq 2ndspos(0,Z) = quoteLRecip(rnil) .
eq 2ndspos(s(N),cons(X,Z)) =
    2ndspos-+2(s(X),cons-+2(X,Z)) .
eq 2ndspos-+2(s(N),cons-+2(X,cons(Y,Z))) =
    quoteLRecip(rcons(posrecip(Y),2ndsneg(N,Z))) .
eq 2ndsneg(0,X4) = quoteLRecip(rnil) .
eq 2ndsneg(s(N),cons(X,Z)) =
    2ndsneg-+2(s(N),cons-+2(X,Z)) .
eq 2ndsneg-+2(s(N),cons-+2(X,cons(Y,Z))) =
    quoteLRecip(rcons(negrecip(Y),2ndspos(N,Z))) .
eq pi-4(X) = quoteLRecip(2ndspos(0,from(X))) .
eq plus(0,Y) = quoteNat(Y) .
eq plus(s(X),Y) = quoteNat(s(plus(X,Y))) .
eq times(0,Y) = quoteNat(0) .
eq times(s(X),Y) = quoteNat(s(times(X,Y))) .
eq square(X) = quoteNat(s(times(X,X))) .
eq quoteNat(0) = 0 .
A.3 Program msquare\_eager

This program uses functions \texttt{minus}, \texttt{square}, \texttt{times}, and \texttt{plus} over natural numbers; they are common to several examples included in this Appendix. The key point of this program is that it is terminating using only positive annotations and including the indices of all symbols.

\begin{verbatim}
obj MINUS-SQUARE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op plus : Nat Nat -> Nat [strat (1 2 0)] .
  op times : Nat Nat -> Nat [strat (1 2 0)] .
  op square : Nat -> Nat [strat (1 2 0)] .
  op minus : Nat Nat -> Nat [strat (1 2 0)] .

  vars M N : Nat .

  eq plus(0,N) = N .
  eq plus(s(M),N) = s(plus(M,N)) .
  eq times(0,N) = 0 .
  eq times(s(M),N) = plus(N,times(M,N)) .
  eq square(N) = times(N,N) .
  eq minus(0,N) = 0 .
  eq minus(s(M),0) = s(M) .
  eq minus(s(M),s(N)) = minus(M,N) .

endo
\end{verbatim}
A.4 Program msquare_apt

This program is identical to msquare_eager but only the annotations which are necessary to make the program complete are included, i.e. we use only canonical positive strategies.

\[
\begin{align*}
\text{obj MINUS-SQUARE is} \\
\quad \text{sort Nat.} \\
\quad \text{op 0 : } \rightarrow \text{Nat.} \\
\quad \text{op s : Nat } \rightarrow \text{Nat } [\text{strat (1)}]. \\
\quad \text{op plus : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 0)}]. \\
\quad \text{op times : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 0)}]. \\
\quad \text{op square : Nat } \rightarrow \text{Nat } [\text{strat (0)}]. \\
\quad \text{op minus : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 2 0)}]. \\
\quad \text{vars M N : Nat.} \\
\quad \text{eq plus(0,N)=N.} \\
\quad \text{eq plus(s(M),N) = s(plus(M,N)).} \\
\quad \text{eq times(0,N)=0.} \\
\quad \text{eq times(s(M),N) = plus(N,times(M,N)).} \\
\quad \text{eq square(N) = times(N,N).} \\
\quad \text{eq minus(0,N)=0.} \\
\quad \text{eq minus(s(M),0) = s(M).} \\
\quad \text{eq minus(s(M),s(N)) = minus(M,N).} \\
\end{align*}
\]

endo

A.5 Program msquare_neg

This program is identical to msquare_apt but negative annotations are included, i.e. we consider canonical arbitrary strategies.

\[
\begin{align*}
\text{obj MINUS-SQUARE is} \\
\quad \text{sort Nat.} \\
\quad \text{op 0 : } \rightarrow \text{Nat.} \\
\quad \text{op s : Nat } \rightarrow \text{Nat } [\text{strat (1)}]. \\
\quad \text{op plus : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 0)}]. \\
\quad \text{op times : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 0)}]. \\
\quad \text{op square : Nat } \rightarrow \text{Nat } [\text{strat (0)}]. \\
\quad \text{op minus : Nat Nat } \rightarrow \text{Nat } [\text{strat (1 -2 0)}]. \\
\quad \text{vars M N : Nat.} \\
\quad \text{eq plus(0,N)=N.} \\
\quad \text{eq plus(s(M),N) = s(plus(M,N)).} \\
\quad \text{eq times(0,N)=0.} \\
\quad \text{eq times(s(M),N) = plus(N,times(M,N)).} \\
\quad \text{eq square(N) = times(N,N).} \\
\quad \text{eq minus(0,N)=0.} \\
\quad \text{eq minus(s(M),0) = s(M).} \\
\quad \text{eq minus(s(M),s(N)) = minus(M,N).} \\
\end{align*}
\]

endo
A.6 Transformed program $\text{msquare\_neg\_noneg}$

The application of the program transformation for removing negative annotations presented in this paper to the program $\text{msquare\_neg}$ produces the following program. Note that annotation 0 in strategy for symbol $s$ is necessary due to problems in Maude for representing and interpreting an empty strategy.

$$\text{obj MINUS-FACT is}$$
$$\quad \text{sort Nat.}$$
$$\quad \text{op 0 : } \rightarrow \text{Nat.}$$
$$\quad \text{op s : Nat } \rightarrow \text{Nat } \quad \text{[strat (\(0\)).]}$$
$$\quad \text{op plus : Nat Nat } \rightarrow \text{Nat } \quad \text{[strat (1 0)].}$$
$$\quad \text{op times : Nat Nat } \rightarrow \text{Nat } \quad \text{[strat (1 0)].}$$
$$\quad \text{op square : Nat } \rightarrow \text{Nat } \quad \text{[strat (0)].}$$
$$\quad \text{op minus : Nat Nat } \rightarrow \text{Nat } \quad \text{[strat (1 0)].}$$
$$\quad \text{op quoteNat : Nat } \rightarrow \text{Nat } \quad \text{[strat (0)].}$$
$$\quad \text{op s-root : Nat } \rightarrow \text{Nat } \quad \text{[strat (1 0)].}$$
$$\quad \text{op minus-+2 : Nat Nat } \rightarrow \text{Nat } \quad \text{[strat (2 0)].}$$
$$\quad \text{vars X Y : Nat.}$$
$$\quad \text{eq plus(0,Y) = quoteNat(Y).}$$
$$\quad \text{eq plus(s(X),Z) = quoteNat(s(plus(X,Y))).}$$
$$\quad \text{eq times(0,Y) = quoteNat(0).}$$
$$\quad \text{eq times(s(X),Y) = quoteNat(times(Y,times(X,Y))).}$$
$$\quad \text{eq square(Y) = quoteNat(times(Y,Y)).}$$
$$\quad \text{eq minus(0,Y) = quoteNat(0).}$$
$$\quad \text{eq minus(s(X),Y) = minus-+2(s(X),Y).}$$
$$\quad \text{eq minus-+2(s(X),0) = quoteNat(s(X)).}$$
$$\quad \text{eq minus(s(X),Y) = minus-+2(s(X),Y).}$$
$$\quad \text{eq minus-+2(s(X),s(Y)) = quoteNat(minus(X,Y)).}$$
$$\quad \text{eq quoteNat(0) = 0.}$$
$$\quad \text{eq quoteNat(s(X)) = s-root(quoteNat(X)).}$$
$$\quad \text{eq quoteNat(plus(X,Y)) = plus(quoteNat(X),Y).}$$
$$\quad \text{eq quoteNat(times(X,Y)) = times(quoteNat(X),Y).}$$
$$\quad \text{eq quoteNat(square(X)) = square(X).}$$
$$\quad \text{eq quoteNat(minus(X,Y)) = minus(quoteNat(X),Y).}$$
$$\quad \text{eq s-root(X) = s(X).}$$
$$\quad \text{eq quoteNat(minus-+2(X,Y)) = minus(X,Y).}$$
$$\text{endo}$$

A.7 Program quicksort

This program is borrowed from Example 3.11 of [5]. Note that auxiliary functions $\text{from}$ and $\text{take}$ for constructing lists are included, as well as two predicates $\text{nflNat}$ and $\text{nfNat}$ to normalize terms, and the connective $\text{and}$. The term used for evaluation is: $\text{nflNat(quickSort(take(10,from(0))))}$

$$\text{obj quickSort is}$$
$$\quad \text{sorts Nat LNat Bool2.}$$
$$\quad \text{op 0 : } \rightarrow \text{Nat.}$$
$$\quad \text{op s : Nat } \rightarrow \text{Nat } \quad \text{[strat (1)].}$$
A.8 Program minsort

This program is borrowed from Example 3.10 of [5]. The call considered for evaluation is: \( \text{nfLNat} \left( \text{minsort} \left( \text{take}(10, \text{from}(0)), \text{nil} \right) \right) \)

\[
\text{obj Minsort is}
\]
\[
\text{sorts Nat LNat Bool2 .}
\]
\[
\text{op 0 : } \to \text{ Nat .}
\]

\[
\text{end}
\]
op \( s : \text{Nat} \to \text{Nat} \) [strat (1)].
op \( \text{nil} : \to \text{LNat} \).
op \( \text{cons} : \text{Nat} \ \text{LNat} \to \text{LNat} \) [strat (1 -2)].
op \( \text{true2} : \to \text{Bool2} \).
op \( \text{false2} : \to \text{Bool2} \).
op \( \text{le} : \text{Nat} \ \text{Nat} \to \text{Bool2} \) [strat (1 2 0)].
op \( \text{app} : \text{LNat} \ \text{LNat} \to \text{LNat} \) [strat (1 0)].
op \( \text{rm} : \text{Nat} \ \text{LNat} \to \text{LNat} \) [strat (2 0)].
op \( \text{min} : \text{LNat} \to \text{Nat} \) [strat (1 0)].
op \( \text{eqNat} : \text{Nat} \ \text{Nat} \to \text{Bool2} \) [strat (1 2 0)].
op \( \text{ifNat} : \text{Nat} \ \text{Nat} \ \text{Nat} \to \text{Nat} \) [strat (1 0)].
op \( \text{ifLNat} : \text{Nat} \ \text{LNat} \ \text{LNat} \to \text{LNat} \) [strat (1 0)].
op \( \text{and} : \text{Bool2} \ \text{Bool2} \to \text{Bool2} \) [strat (1 0)].
op \( \text{minsort} : \text{LNat} \ \text{LNat} \to \text{LNat} \) [strat (1 2 0)].
op \( \text{nfLNat} : \text{LNat} \to \text{Bool2} \) [strat (1 0)].
op \( \text{nfNat} : \text{Nat} \to \text{Bool2} \) [strat (1 0)].

vars \( X, Y : \text{Nat} \). vars \( Z, W : \text{LNat} \). vars \( A, B : \text{Bool2} \).

eq \( \text{le}(0, Y) = \text{true2} \).

22
eq and(false2,A) = false2.
endo

A.9 Program mod

This program is borrowed from Example 3.5 of [5]. Auxiliary functions for natural numbers are included, namely fact, times, and plus. The call considered for evaluation is: mod(fact(fact(3)),2)

obj MOD is
  sorts Nat Bool2.
  op 0 : -> Nat.
  op s : Nat -> Nat [strat (1)].
  op true2 : -> Bool2.
  op false2 : -> Bool2.
  op minus : Nat Nat -> Nat [strat (1 -2 0)].
  op mod : Nat Nat -> Nat [strat (1 -2 0)].
  op le : Nat Nat -> Bool2 [strat (1 -2 0)].
  op ifNat : Bool2 Nat Nat -> Nat [strat (1 0)].
  op plus : Nat Nat -> Nat [strat (1 0)].
  op times : Nat Nat -> Nat [strat (1 0)].
  op fact : Nat -> Nat [strat (1 0)].
vars M N : Nat.
eq le(0,M) = true2.
eq le(s(N),0) = false2.
eq le(s(N),s(M)) = le(N,M).
eq minus(0,N) = 0.
eq minus(s(M),0) = s(M).
eq minus(s(M),s(N)) = minus(M,N).
eq mod(0,M) = 0.
eq mod(s(N),0) = 0.
eq mod(s(N),s(M)) = ifNat(le(M,N),mod(minus(N,M),s(M)),s(N)).
eq ifNat(true2,N,M) = N.
eq ifNat(false2,N,M) = M.
eq plus(0,N) = N.
eq plus(s(M),N) = s(plus(M,N)).
eq times(0,N) = 0.
eq times(s(M),N) = plus(N,times(M,N)).
eq fact(0) = s(0).
eq fact(s(N)) = times(s(N),fact(N)).
endo

A.10 Program mod'

This program is similar to program mod but positive annotations are provided for symbols times and plus in order to avoid differences due to sharing of variables. The call considered for evaluation is: mod(fact(fact(3)),2)

obj MOD is
  sorts Nat Bool2.
  op 0 : -> Nat [strat ()].
A.11 Program average

This program is borrowed from Example 3.15 of [5]. Auxiliary functions for natural numbers are included, namely fact, times, and plus. The call considered for evaluation is: average(square(square(4)),square(square(4)))

obj AVERAGE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op average : Nat Nat -> Nat [strat (1 -2 0)] .
  op plus : Nat Nat -> Nat [strat (1 2 0)] .
  op times : Nat Nat -> Nat [strat (1 0)] .
  op fact : Nat -> Nat [strat (1 0)] .
  op square : Nat -> Nat [strat (1 0)] .
  vars M N : Nat .
  eq average(0,0) = 0 .
  eq average(0,s(0)) = 0 .
  eq average(0,s(s(0))) = s(0) .
  eq average(s(M),N) = average(M,s(N)) .
  eq average(M,s(s(N))) = s(average(s(M),N)) .
endo
B Proof of termination of pi program

Consider the program of Section A.1. After applying the transformation included in [1] for proving termination, we obtain the following program:

```plaintext
obj PI4tr is
  sorts Nat LNat Recip LRecip .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op posrecip : Nat -> Recip [strat (1)] .
  op negrecip : Nat -> Recip [strat (1)] .
  op nil : -> LNat .
  op cons : Nat LNat -> LNat [strat (1)] .
  op cons2 : Nat LNat -> LNat [strat (2)] .
  op rnil : -> LRecip .
  op rcons : Recip LRecip -> LRecip [strat (1 2)] .
  op from : Nat -> LNat [strat (1 0)] .
  op 2ndspos : Nat LNat -> LRecip [strat (1 2 0)] .
  op 2ndsneg : Nat LNat -> LRecip [strat (1 2 0)] .
  op pi : Nat -> LRecip [strat (1 0)] .
  op plus : Nat Nat -> Nat [strat (1 2 0)] .
  op times : Nat Nat -> Nat [strat (1 2 0)] .
  op square : Nat -> Nat [strat (1 0)] .
vars N X Y : Nat . var Z : LNat .
  eq from(X) = cons(X,from(s(X))) .
  eq 2ndspos(0,Z) = rnil .
  eq 2ndspos(s(N),cons(X,Z)) = 2ndspos(s(N),cons2(X,Z)) .
  eq 2ndspos(s(N),cons2(X,cons(Y,Z))) =
     rcons(posrecip(Y),2ndsneg(N,Z)) .
  eq 2ndsneg(0,Z) = rnil .
  eq 2ndsneg(s(N),cons(X,Z)) = 2ndsneg(s(N),cons2(X,Z)) .
  eq 2ndsneg(s(N),cons2(X,cons(Y,Z))) =
     rcons(negrecip(Y),2ndspos(N,Z)) .
  eq pi(X) = 2ndspos(X,from(0)) .
  eq plus(0,Y) = Y .
  eq plus(s(X),Y) = s(plus(X,Y)) .
  eq times(0,Y) = 0 .
  eq times(s(X),Y) = plus(Y,times(X,Y)) .
  eq square(X) = times(X,X) .
endo
```

Following [13], in order to prove termination of PI4tr (which only contains
positive annotations), we can use the techniques for proving termination of context-sensitive rewriting (see [16] for a survey of these techniques). The application of Zantema’s transformation ([20]) to remove positive annotations, yields the following TRS (in a generic syntax not bound to OBJ programs):

\[
\begin{align*}
\text{from}(X) & \rightarrow \text{cons}(X, \text{n\_from}(s(X))) \\
2\text{ndspos}(0, Z) & \rightarrow \text{rnil} \\
2\text{ndspos}(s(N), \text{cons}(X, Z)) & \rightarrow 2\text{ndspos}(s(N), \text{cons2}(X, \text{activate}(Z))) \\
2\text{ndspos}(s(N), \text{cons2}(X, \text{cons}(Y, Z))) & \rightarrow \text{rcons}(\text{posrecip}(Y), 2\text{ndspos}(N, \text{activate}(Z))) \\
2\text{ndspos}(0, Z) & \rightarrow \text{rnil} \\
2\text{ndspos}(s(N), \text{cons}(X, Z)) & \rightarrow 2\text{ndspos}(s(N), \text{cons2}(X, \text{activate}(Z))) \\
2\text{ndspos}(s(N), \text{cons2}(X, \text{cons}(Y, Z))) & \rightarrow \text{rcons}(\text{negrecip}(Y), 2\text{ndspos}(N, \text{activate}(Z))) \\
\text{pi}(X) & \rightarrow 2\text{ndspos}(X, \text{from}(0)) \\
\text{plus}(0, Y) & \rightarrow Y \\
\text{plus}(s(X), Y) & \rightarrow s(\text{plus}(X, Y)) \\
\text{times}(0, Y) & \rightarrow 0 \\
\text{times}(s(X), Y) & \rightarrow \text{plus}(Y, \text{times}(X, Y)) \\
\text{square}(X) & \rightarrow \text{times}(X, X) \\
\text{from}(X) & \rightarrow \text{n\_from}(X) \\
\text{activate}(\text{n\_from}(X)) & \rightarrow \text{from}(X) \\
\text{activate}(X) & \rightarrow X
\end{align*}
\]

Termination of this program can be proved with the CiME 2.0 system (available at http://cime.lri.fr/) by using dependency graphs and simple-mixed interpretations:

CiME> termination R;
Entering the termination expert. Verbose level = 0
checking each of the 3 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[rnil] = 0;
[0] = 0;
[activate](X0) = X0;
[n\_from](X0) = 0;
[square](X0) = X0^2;
[pi](X0) = 0;
[negrecip](X0) = 0;
[posrecip](X0) = 0;
[s](X0) = X0 + 1;
[from](X0) = 0;
[times](X0,X1) = X1\times X0;
[plus](X0,X1) = X1 + X0;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](X0,X1) = 0;
[cons](X0,X1) = 0;
[\text{\textquoteleft plus\textquoteleft}](X0,X1) = X0;

26
checking component 2 (disjunction of 1 constraints)
[rnil] = 0;
[0] = 0;
[activate](X0) = X0;
[n_from](X0) = 0;
[square](X0) = X0^2;
[pi](X0) = 0;
[negrecip](X0) = 0;
[posrecip](X0) = 0;
[s](X0) = X0 + 1;
[from](X0) = 0;
[times](X0,X1) = X1*X0;
[plus](X0,X1) = X1 + X0;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](X0,X1) = 0;
[cons](X0,X1) = 0;
['times'](X0,X1) = X0;

checking component 3 (disjunction of 2 constraints)
[rnil] = 0;
[0] = 0;
[activate](X0) = X0;
[n_from](X0) = 0;
[square](X0) = X0^2;
[pi](X0) = 0;
[negrecip](X0) = 0;
[posrecip](X0) = 0;
[s](X0) = X0 + 1;
[from](X0) = 0;
[times](X0,X1) = X1*X0;
[plus](X0,X1) = X1 + X0;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](X0,X1) = 0;
[cons](X0,X1) = 0;
['2ndsneg'](X0,X1) = X0;
['2ndspos'](X0,X1) = X0;

Termination proof found.
Execution time: 4.200000 sec
- : unit = ()