

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)



Informe Técnico / Technical Report

Ref. No.: DSIC-II/02/04	Pages: 38
Title: On-demand Evaluation by Program Transformation	
Author(s): M. Alpuente, S. Escobar, and S. Lucas	
Date: 23/12/03	
Keywords: functional programming, term rewriting, demandness, implementation, lazy evaluation, OBJ, on-demand strategy annotations, program transformation	

Vº Bº
Leader of research Group

Author(s)

On-demand Evaluation by Program Transformation*

M. Alpuente, S. Escobar and S. Lucas
DSIC, UPV, Camino de Vera s/n, E-46022 Valencia, Spain.
{alpuente, sescobar, slucas}@dsic.upv.es

January 21, 2004

Abstract

Strategy annotations are used in eager programming languages (e.g., OBJ2, OBJ3, CafeOBJ, and Maude) for improving efficiency and/or reducing 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 “evaluation on-demand”. Recently, we have developed a formal description of the operational meaning of such on-demand strategy annotations which improves previous formalizations that were lacking satisfactory computational properties. In this paper, we introduce an automatic, semantics-preserving program transformation which produces a program (without negative annotations) which can be then correctly executed by typical OBJ interpreters. Moreover, to demonstrate the practicality of our ideas, the program transformation has been implemented and we compare the behavior of transformed programs with the original ones on a set of representative benchmarks.

Keywords: functional programming, term rewriting, demandness, implementation, lazy evaluation, OBJ, on-demand strategy annotations, program transformation

1 Introduction

Eager rewriting-based programming languages such as Lisp, OBJ*, CafeOBJ, ELAN, or Maude evaluate 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

*Work partially supported by MCyT under grants TIC2001-2705-C03-01, HA2001-0059 and HU2001-0019. This paper is a revised and improved version of [4].

local strategies) have been used in OBJ2 [12], OBJ3 [14], CafeOBJ [13], and Maude [9] to improve efficiency and (hopefully) avoid nontermination. Local strategies are used in OBJ programs¹ for guiding the *evaluation strategy* (abbr. *E-strategy*): when considering a function call $f(t_1, \dots, t_k)$, only the arguments whose indices are present as *positive* integers in the local strategy for f are evaluated (following the specified ordering). If 0 is found, then the evaluation of f is attempted. Whenever the user provides no local strategy for a given symbol, the (Maude, OBJ*, CafeOBJ) interpreter automatically assigns a *default E-strategy*. For instance, the default local strategy of Maude associates the list $(1\ 2 \dots k\ 0)$ to each k -ary symbol f having no explicit strategy, i.e. all arguments are marked as evaluable. We adopt the default local strategy of Maude.

Example 1 Consider the following program `pi` which codifies the well-known infinite series expansion to approximate number π :

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

```
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 seriespos : Nat LNat -> LRecip .
  op seriesneg : Nat LNat -> LRecip .
  op pi : Nat -> LRecip .
  vars N X Y : Nat . var Z : LNat .
  eq from(X) = cons(X,from(s(X))) .
  eq seriespos(0,Z) = rnil .
  eq seriespos(s(N),cons(X,cons(Y,Z)))
    = rcons(posrecip(Y),seriesneg(N,Z)) .
  eq seriesneg(0,Z) = rnil .
  eq seriesneg(s(N),cons(X,cons(Y,Z)))
    = rcons(negrecip(Y),seriespos(N,Z)) .
  eq pi(X) = seriespos(X,from(0)) .
endo
```

A term² `pi(2)` approximates the number $\pi/4$ using 2 elements of the series expansion, i.e. the intended behavior is

$$\text{pi}(2) \rightarrow^* \text{rcons}(\text{posrecip}(1), \text{rcons}(\text{negrecip}(3), \text{rnil}))$$

¹By OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

²Naturals $1, 2, \dots$ are used as shorthand to numbers $s^n(0)$ where $n = 1, 2, \dots$

where `posrecip(n)` denotes the positive reciprocal $1/n$ and `negrecip(n)` denotes $-1/n$. Note that since the default local strategy is applied, all arguments are marked as evaluable and the program is nonterminating since innermost evaluation diverges due to the equation defining symbol `from`:

```

pi(2) → seriespos(2,from(0))
      → seriespos(2,cons(0,from(1)))
      → seriespos(2,cons(0,cons(1,from(2)))) → ...

```

The evaluation of the term `pi(2)` as performed by a typical OBJ interpreter such as Maude³ is:

```

Maude> red pi(s(s(0))) .
reduce in PI : pi(s(s(0))) .
Segmentation fault

```

In order to avoid nontermination, annotation 2 should be removed from the (default) local strategy (1 2 0) for symbol `cons`.

Example 2 After removing annotation 2 from the (default) local strategy for symbol `cons` in Example 1, the program becomes terminating under innermost rewriting with the following resulting local strategy for symbol `cons`:

```

op cons : Nat LNat -> LNat [strat (1)] .

```

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

Example 3 The evaluation of `pi(2)` using the program of Example 2 yields the following sequence:

```

pi(2)
  → seriespos(2,from(0))
  → seriespos(2,cons(0,from(1)))

```

Its evaluation performed by Maude is:

```

Maude> red pi(s(s(0))) .
reduce in PI : pi(s(s(0))) .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result LRecip: seriespos(s(s(0)), cons(0, from(s(0))))

```

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

```

seriespos(2,cons(0,from(1)))
  → seriespos(2,cons(0,cons(1,from(2))))

```

is required in order to apply the second rule of `seriespos` which allows to obtain the intended constructor normal form of Example 1.

The handicaps of using only positive annotations regarding correctness and completeness of computations are discussed in [2, 3, 17, 18, 22, 23]: 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.

³We use version 2.0 available at <http://maude.cs.uiuc.edu/current/system/>.

In [22, 23], *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 [10, 14, 23]. For instance, subterm `from(1)` in Example 3 is *demand*ed by the second rule of `seriespos`. Thus, `(1 -2)` would be the appropriate local strategy for `cons` as pointed out in [22]; i.e. the first argument is always evaluated but the second argument is evaluated only “on-demand”. Then, the evaluation of the symbol `cons` under strategy `(1 -2)` is able to normalize `pi(2)` to its intended normal form without entering in a nonterminating evaluation, whereas evaluation only with positive annotations enters an infinite derivation (as shown in Example 1) or does not provide the intended normal form (as shown in Example 2).

It is worthy to note that the laziness provided by the calculus with on-demand strategy annotations is simpler than typical functional lazy rewriting and the appropriate (on-demand) strategy annotations for achieving suitable normal forms can be inferred from the program (see [2, 3, 17, 18, 23, 22]).

However, *on-demand* strategy annotations have not been properly 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 4 Consider the program of Example 1 where the local strategy for `cons` includes the on-demand annotation `-2` (this is the only change to the program):

```
op cons : Nat LNat -> LNat [strat (1 -2)] .
```

The OBJ3 interpreter does not implement negative (on-demand) annotations even if it does accept this program and the evaluation of `pi(2)` surprisingly delivers the very same result as in Example 2. That is, the negative annotation is just disregarded by the OBJ3 interpreter (which, in this case, causes loss of completeness). Also, the Maude interpreter does not implement negative annotations either but also does accept this program and the evaluation of the considered expression diverges as in Example 1. This is because the negative annotation `-2` is interpreted as a positive one thus resulting in nontermination.

On the other hand, CafeOBJ is able to deal with negative annotations using the on-demand evaluation model of [22] and is able to compute the intended value `rcons(posrecip(1),rcons(negrecip(3),rnil))` of input term `pi(2)`. However, a number of problems of the on-demand evaluation model of [22, 23] are pointed out in [2], which we recall in the following example.

Example 5 [2] Consider the following OBJ program:

```
obj LENGTH 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 length : LNat -> Nat      [strat (0)] .
op length' : LNat -> Nat    [strat (-1 0)] .
vars X Y : Nat . var Z : LNat .
eq from(X) = cons(X,from(s(X))) .
eq length(nil) = 0 .
eq length(cons(X,Z)) = s(length'(Z)) .
eq length'(Z) = length(Z) .
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 length' for enabling this step (the negative annotation -1 is included for length' but the corresponding rule includes a variable at the first argument of length') and no further evaluation on $\text{length}(\text{from}(0))$ should be performed (due to the local strategy (0) of length which forbids evaluation on any argument of length). However, the annotation -1 of function length' is treated in such a way by the operational model of [23, 22] that the on-demand evaluation of the expression $\text{length}'(\text{from}(0))$ yields an infinite evaluation sequence (see [2] for a more detailed explanation).

We proposed in [2] a solution to these problems which is based on a suitable extension of the E -evaluation strategy of OBJ-like languages (that only considers annotations given as natural numbers) to cope with on-demand strategy annotations. Our strategy incorporates a better treatment of demandness and also enjoys good computational properties; in particular, we show how to use it for computing (head-)normal forms and we prove it is conservative w.r.t. other on-demand strategies: *lazy rewriting* [11] and *on-demand rewriting* [17]. A program transformation for proving termination of the on-demand evaluation strategy was also formalized, which relies on standard techniques. Furthermore, a *direct* implementation of the on-demand evaluation strategy of [2] has been developed. The system is called `OnDemandOBJ` and is publicly available at <http://www.dsic.upv.es/users/elp/soft.html> (see [5] for a description).

In this paper, we show how OBJ programs that use local strategies containing negative annotations (and hence could be correctly executed by using the evaluation strategy proposed in [2]) can be (also) executed in the existing OBJ implementations which only admit positive annotations (e.g. `Maude`). This is done by means of an automatic program transformation which encodes the ‘on-demand’ strategy instrumented by the negative annotations within new function symbols (and corresponding program rules) that only use positive strategy annotations. Before entering into technical details, we give an example which illustrates the power of our transformation.

Example 6 *The program of Example 4 is transformed by using our method into the following OBJ program without negative annotations.*

```

obj PINoNeg 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 cons+2   : Nat LNat -> LNat      [strat (2)] .
op consroot : Nat LNat -> LNat      [strat (1 0)] .
op rnil     : -> LRecip .
op rcons    : Recip LRecip -> LRecip [strat (1 2)] .
op from     : Nat -> LNat           [strat (1 0)] .
op seriespos : Nat LNat -> LRecip   [strat (1 2 0)] .
op seriespos+2 : Nat LNat -> LRecip [strat (2 0)] .
op seriesneg : Nat LNat -> LRecip   [strat (1 2 0)] .
op seriesneg+2 : Nat LNat -> LRecip [strat (2 0)] .
op pi       : Nat -> LRecip         [strat (1 0)] .
ops quote  : Nat -> Nat             [strat (0)] .
ops quote  : LNat -> LNat           [strat (0)] .
ops quote  : Recip -> Recip         [strat (0)] .
ops quote  : LRecip -> LRecip      [strat (0)] .
vars N X Y : Nat . var Z : LNat .
var R W : Recip . var L V : LRecip .
eq from(X) = quote(cons(X,from(s(X)))) .
eq seriespos(0,Z) = quote(rnil) .
eq seriespos(s(N),cons(X,Z))
  = seriespos+2(s(N),cons+2(X,ondemand(Z))) .
eq seriespos+2(s(N),cons+2(X,cons(Y,Z)))
  = quote(rcons(posrecip(Y),seriesneg(N,Z))) .
eq seriesneg(0,Z) = quote(rnil) .
eq seriesneg(s(N),cons(X,Z))
  = seriesneg+2(s(N),cons+2(X,ondemand(Z))) .
eq seriesneg+2(s(N),cons+2(X,cons(Y,Z)))
  = quote(rcons(negrecip(Y),seriespos(N,Z))) .
eq pi(X) = quote(seriespos(X,from(0))) .
eq quote(0) = 0 .
eq quote(s(N)) = s(quote(N)) .
eq quote(nil) = nil .
eq quote(cons(X,Z)) = consroot(quote(X),Z) .
eq quote(from(X)) = from(quote(X)) .
eq quote(posrecip(X)) = posrecip(quote(X)) .
eq quote(negrecip(X)) = negrecip(quote(X)) .
eq quote(rnil) = rnil .
eq quote(rcons(W,V)) = rcons(quote(W),quote(V)) .
eq quote(seriespos(X,Z)) = seriespos(quote(X),quote(Z)) .
eq quote(seriesneg(X,Z)) = seriesneg(quote(X),quote(Z)) .
eq quote(pi(X)) = pi(quote(X)) .
eq consroot(X,Z) = cons(X,Z) .
eq quote(seriespos+2(X,Z)) = seriespos+2(X,Z) .
eq quote(cons+2(X,Z)) = cons+2(X,Z) .
eq quote(seriesneg+2(X,Z)) = seriesneg+2(X,Z) .
eq ondemand(0) = 0 .
eq ondemand(s(N)) = s(N) .
eq ondemand(nil) = nil .
eq ondemand(cons(X,Z)) = cons(X,Z) .
eq ondemand(from(X)) = from(quote(X)) .
eq ondemand(posrecip(X)) = posrecip(X) .

```

```

eq ondemand(negrecip(X)) = negrecip(X) .
eq ondemand(rnil) = rnil .
eq ondemand(rcons(W,V)) = rcons(W,V) .
eq ondemand(seriespos(X,Z)) = seriespos(quote(X),quote(Z)) .
eq ondemand(seriesneg(X,Z)) = seriesneg(quote(X),quote(Z)) .
eq ondemand(pi(X)) = pi(quote(X)) .
eq ondemand(seriespos+2(X,Z)) = seriespos+2(X,Z) .
eq ondemand(cons+2(X,Z)) = cons+2(X,Z) .
eq ondemand(seriesneg+2(X,Z)) = seriesneg+2(X,Z) .
endo

```

Informally, new symbols `seriespos+2`, `seriesneg+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 `seriespos` and `seriesneg` in Example 4 are the only ones which could demand the evaluation of the second argument of `cons`. The extra symbols `quote` and `ondemand` are introduced to preserve correctness w.r.t. reductions with positive indices and on-demand computations, respectively.

Roughly speaking, for each constructor symbol `c` with a negative annotation $-i$, we remove all strategy annotations for `c` and introduce an auxiliary constructor symbol `c+i` with positive annotation i . Also, we introduce a defined symbol `croot` without the negative annotations but with the positive ones plus 0 and we introduce a new rule which is used to translate the new symbol `croot` back to `c`. Then, we add new rules which re-define symbols using constructor `c` in terms of `c`, `croot`, and `c+i`. Finally, for each program rule $l \rightarrow r$, we introduce a symbol `quote` in r and add a number of new rules for symbols `quote` and `ondemand` which transform `c` into `croot` and help to appropriately (head)-normalize terms.

Now, term `pi(2)` is correctly evaluated using the Maude interpreter (which simulates the on-demand evaluation of [2])

```

Maude> red quote(pi(s(s(0)))) .
reduce in PINoNeg : quote(pi(s(s(0)))) .
rewrites: 80 in -1ms cpu (0ms real) (~ rewrites/second)
result LRecip: rcons(posrecip(s(0)),
                    rcons(negrecip(s(s(0))), rnil))

```

After some preliminaries in Section 2, we recall the on-demand evaluation of [2] in Section 3. Then, Section 4 introduces the program transformation together with completeness and correctness results. In Section 5, we experimentally demonstrate that the program transformation pays off in practice. Section 6 concludes. Proofs of all technical results can be found at the Appendix A.

2 Preliminaries

We follow the standard framework of term rewriting for developing our results (see [8, 24] for missing definitions). Given a set A , $\mathcal{P}(A)$ denotes the set of all subsets of A . Let $R \subseteq A \times A$ be a binary relation on a set A . We denote the reflexive closure of R by R^- , its transitive closure by R^+ , and its reflexive and transitive closure by R^* . An element $a \in A$ is an R -normal form, if there exists

no b such that $a R b$. We say that b is an R -normal form of a (written $a R^! b$), if b is an R -normal form and $a R^* b$. We say that R is *terminating* iff there is no infinite sequence $a_1 R a_2 R a_3 \dots$.

Throughout the paper, \mathcal{X} denotes a countable set of variables and \mathcal{F} denotes a signature, i.e. a set of function symbols $\{\mathbf{f}, \mathbf{g}, \dots\}$, each having a fixed arity given by a function $ar : \mathcal{F} \rightarrow \mathbb{N}$. We denote the set of terms built from \mathcal{F} and \mathcal{X} by $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A term is said to be linear if it has no multiple occurrences of a single variable. Terms are viewed as labelled trees in the usual way. Let $Subst(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ denote the set of substitutions. We denote by id the “identity” substitution: $id(x) = x$ for all $x \in \mathcal{X}$. Positions p, q, \dots are represented by chains of positive natural numbers used to address subterms of t . We denote the empty chain by Λ . By $Pos(t)$ we denote the set of positions of a term t . Given a set $S \subseteq \mathcal{F} \cup \mathcal{X}$, $Pos_S(t)$ denotes positions in t where symbols in S occur. When no confusion arises, we denote $Pos_{\{f\}}(t)$ as $Pos_f(t)$ for a symbol $f \in \mathcal{F} \cup \mathcal{X}$. Given positions p, q , we denote its concatenation as $p.q$. Positions are ordered by the standard prefix ordering \leq . Positions can also be ordered by the lexicographical ordering: $p \leq_{lex} q$ iff $p \leq q$ or $p = w.i.p'$, $q = w.j.q'$, $i, j \in \mathbb{N}$, and $i < j$. Given a set of positions P , $minimal_{\leq}(P)$ is the set of minimal positions of P w.r.t. \leq . If p is a position, and Q is a set of positions, $p.Q$ is the set $\{p.q \mid q \in Q\}$. The subterm at position p of t is denoted as $t|_p$, and $t[s]_p$ is the term t with the subterm at position p replaced by s . The symbol labelling the root of t is denoted as $root(t)$.

A rewrite rule is an ordered pair (l, r) , written $l \rightarrow r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $Var(r) \subseteq Var(l)$. The left-hand side (*lhs*) of the rule is l and r is the right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of rewrite rules. $L(\mathcal{R})$ denotes the set of *lhs*'s of \mathcal{R} . A TRS \mathcal{R} is left-linear if for all $l \in L(\mathcal{R})$, l is a linear term. Given $\mathcal{R} = (\mathcal{F}, R)$, we take \mathcal{F} as the disjoint union $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors* and symbols $f \in \mathcal{D}$, called *defined functions*, where $\mathcal{D} = \{root(l) \mid l \rightarrow r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. We say that a rule $l \rightarrow r$ defines $f \in \mathcal{D}$ if $root(t) = f$. A TRS $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ is a constructor system (CS) if for all $f(l_1, \dots, l_k) \in L(\mathcal{R})$, $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, for $1 \leq i \leq k$. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to s (at position p), written $t \xrightarrow{\mathcal{R}}_p s$ (or just $t \rightarrow s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $l \rightarrow r \in R$, $p \in Pos(t)$ and substitution σ .

A mapping $\mu : \mathcal{F} \rightarrow \mathcal{P}(\mathbb{N})$ is a *replacement map* (or \mathcal{F} -map) if $\forall f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \dots, ar(f)\}$ [16]. Let $M_{\mathcal{F}}$ be the set of all \mathcal{F} -maps. The ordering \sqsubseteq on $M_{\mathcal{F}}$, the set of all \mathcal{F} -maps, is: $\mu \sqsubseteq \mu'$ if for all $f \in \mathcal{F}$, $\mu(f) \subseteq \mu'(f)$. Let $\mu_{\mathcal{R}}^{can}$ be the canonical replacement map, i.e. *the most restrictive replacement map which ensures that the non-variable subterms of the left-hand sides of the rules of \mathcal{R} are replacing*, which is easily obtained from \mathcal{R} : $\forall f \in \mathcal{F}$, $i \in \{1, \dots, ar(f)\}$, $i \in \mu_{\mathcal{R}}^{can}(f)$ iff $\exists l \in L(\mathcal{R}), p \in Pos_{\mathcal{F}}(l), (root(l)|_p) = f \wedge p.i \in Pos_{\mathcal{F}}(l)$. Let $CM_{\mathcal{R}} = \{\mu \in M_{\mathcal{F}} \mid \mu_{\mathcal{R}}^{can} \sqsubseteq \mu\}$ be the set of replacement maps which are less or equally restrictive than $\mu_{\mathcal{R}}^{can}$.

3 On-demand evaluation strategy

A local strategy for a k -ary symbol $f \in \mathcal{F}$ is a sequence $\varphi(f)$ of integers taken from $\{-k, \dots, -1, 0, 1, \dots, k\}$ which are given in parentheses. A mapping φ that associates a local strategy $\varphi(f)$ to every $f \in \mathcal{F}$ is called an E -strategy map [21, 22]. The E -strategy maps are used to correctly guide the evaluation strategy of OBJ-like languages in order to evaluate expressions. In the following, we recall the on-demand E -evaluation strategy of [2].

Let \mathbb{L} be the set of all lists consisting of integers. We define an (embedding) ordering \sqsubseteq between sequences of integers as: $nil \sqsubseteq L, \forall L \in \mathbb{L}; (i_1 i_2 \dots i_m) \sqsubseteq (j_1 j_2 \dots j_n)$ if $i_1 = j_1$ and $(i_2 \dots i_m) \sqsubseteq (j_2 \dots j_n)$; or $(i_1 i_2 \dots i_m) \sqsubseteq (j_1 j_2 \dots j_n)$ if $i_1 \neq j_1$ and $(i_1 i_2 \dots i_m) \sqsubseteq (j_2 \dots j_n)$. An ordering \sqsubseteq between strategy maps is defined: $\varphi \sqsubseteq \varphi'$ if, for all $f \in \mathcal{F}$, $\varphi(f) \sqsubseteq \varphi'(f)$. Roughly speaking, $\varphi \sqsubseteq \varphi'$ if for all $f \in \mathcal{F}$, $\varphi'(f)$ is $\varphi(f)$ except by the introduction of some additional indices. Sometimes, it is interesting to get rid of the ordering on (non-nullary) indices in a given local strategy; for this reason, given an E -strategy map φ , we introduce the following replacement map $\mu^\varphi(f) = \{|i| \mid i \in \varphi(f) \wedge i \neq 0\}$.

Let \mathbb{L}_n be the set of all lists of integers whose absolute value does not exceed $n \in \mathbb{N}$. Given a E -strategy map φ , we use the signature⁴ $\mathcal{F}_\varphi^\# = \cup\{\overline{f}_{L_1|L_2}, \overline{f}_{L_1|L_2} \mid f \in \mathcal{F} \wedge L_1, L_2 \in \mathbb{L}_{ar(f)}.(L_1++L_2 \sqsubseteq \varphi(f))\}$ and labeled variables $\mathcal{X}_\varphi^\# = \{x_{nil|nil} \mid x \in \mathcal{X}\}$ for marking ordinary terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ as terms $t \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$. Overlining the root symbol of a subterm means that no evaluation is required for that subterm and the control goes back to the parent; the auxiliary list L_1 in the subscript $L_1 \mid L_2$ is interpreted as a kind of memory of previously considered annotations. The main idea is that annotations move from L_2 to L_1 once they have been processed.

We use $f^\#$ to denote \overline{f} or \overline{f} for a symbol $f \in \mathcal{F}$. We define the list of active indices of a labeled symbol $f_{L_1|L_2}^\#$ as

$$active(f_{L_1|L_2}^\#) = \begin{cases} L_1 & \text{if } L_1 \neq nil \\ L_2 & \text{if } L_1 = nil \end{cases}$$

The operator φ is extended to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$ as follows:

$$\varphi(t) = \begin{cases} x_{nil|nil} & \text{if } t = x \in \mathcal{X} \\ f_{nil|\varphi(f)}(\varphi(t_1), \dots, \varphi(t_k)) & \text{if } t = f(t_1, \dots, t_k) \end{cases}$$

Also, the operator $erase : \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ drops labelings from terms.

Given terms $t, l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we let $Pos_{\neq}(t, l) = \{p \in Pos_{\mathcal{F}}(t) \cap Pos_{\mathcal{F}}(l) \mid root(l|_p) \neq root(t|_p)\}$. We define the set of demanded positions of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ w.r.t. l (a lhs of a rule defining $root(t)$), i.e. the set of (positions of) maximal disagreeing subterms as:

$$DP_l(t) = \begin{cases} minimal_{\leq}(Pos_{\neq}(t, l)) & \text{if } minimal_{\leq}(Pos_{\neq}(t, l)) \subseteq Pos_{\mathcal{D}}(t) \\ \emptyset & \text{otherwise} \end{cases}$$

⁴The function $++$ defines the concatenation of two sequences of integers.

Note that the restriction of disagreeing positions to positions that correspond to defined symbols (by using $\mathcal{Pos}_{\mathcal{D}}(t)$) disables the evaluation of subterms which could never produce a redex (see [2, 6, 15, 20]).

Example 7 Let us consider the following lhs's $l_1 = \text{seriespos}(0, Z)$ and $l_2 = \text{seriespos}(s(N), \text{cons}(X, \text{cons}(Y, Z)))$, where $\mathcal{C} = \{\text{cons}, \text{nil}, s, 0\}$ and $\mathcal{D} = \{\text{seriespos}, \text{from}\}$. Consider $t_1 = \text{seriespos}(s(0), \text{cons}(0, \text{nil}))$, then $DP_{l_1}(t_1) = DP_{l_2}(t_1) = \emptyset$, i.e. no position is demanded by l_1 or l_2 because of a constructor clash: with subterm $s(0)$ at position 1 for l_1 and with subterm nil at position 2.2 for l_2 . Let $t_2 = \text{seriespos}(s(0), \text{cons}(0, \text{from}(s(0))))$, we have $DP_{l_1}(t_2) = \emptyset$ and $DP_{l_2}(t_2) = \{2.2\}$, i.e. position 2.2 is demanded by l_2 since it is function-rooted.

We define the set of *positive* positions of a term $s \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ as $\mathcal{Pos}_P(s) = \{\Lambda\} \cup \{i.\mathcal{Pos}_P(s|_i) \mid i > 0 \text{ and } \text{active}(\text{root}(s)) \text{ contains } i\}$ and the set of *active* positions as $\mathcal{Pos}_A(s) = \{\Lambda\} \cup \{i.\mathcal{Pos}_A(s|_i) \mid i > 0 \text{ and } \text{active}(\text{root}(s)) \text{ contains } i \text{ or } -i\}$. We also define the set of positions with *empty* annotation list as $\mathcal{Pos}_{\text{nil}}(s) = \{p \in \mathcal{Pos}(s) \mid \text{root}(s|_p) = f_{L|\text{nil}}\}$. Then, the set of *active demanded* positions of a term $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ w.r.t. l (a lhs of a rule defining $\text{root}(\text{erase}(t))$) is defined as follows:

$$ADP_l(t) = \begin{cases} DP \cap \mathcal{Pos}_A(t) & \text{if } DP \not\subseteq \mathcal{Pos}_P(t) \cup \mathcal{Pos}_{\text{nil}}(t) \\ & \text{where } DP = DP_l(\text{erase}(t)) \\ \emptyset & \text{otherwise} \end{cases}$$

and the set of active demanded positions of $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ w.r.t. TRS \mathcal{R} as $ADP_{\mathcal{R}}(t) = \cup\{ADP_l(t) \mid l \rightarrow r \in \mathcal{R} \wedge \text{root}(\text{erase}(t)) = \text{root}(l)\}$.

Note that the restriction of active demanded positions to non-positive and non-empty positions is consistent w.r.t. the intended meaning of strategy annotations since positive or empty positions should not be evaluated on-demand (see [2]).

Example 8 Consider lhs $l = \text{seriespos}(s(N), \text{cons}(X, \text{cons}(Y, Z)))$ from Example 4. Let

$$t_1 = \text{seriespos}_{(2)|(0)}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}), \text{cons}_{(1\ 2)|\text{nil}}(0_{\text{nil}|\text{nil}}, \text{from}_{\text{nil}|(1\ 0)}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}))))$$

we have $DP_l(\text{erase}(t_1)) = \{2.2\}$ (as the term t_2 in Example 7) but $ADP_l(t_1) = \emptyset$, i.e. position 2.2 is demanded by l but it is a positive position (see index 2 in the memoizing list of symbol cons). Let

$$t_2 = \text{seriespos}_{(2)|(0)}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}), \text{cons}_{(1\ -2)|\text{nil}}(0_{\text{nil}|\text{nil}}, \text{from}_{\text{nil}|\text{nil}}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}))))$$

we have $ADP_l(t_2) = \emptyset$, i.e. position 2.2 is still demanded by l but it is rooted by a symbol with an empty annotation list (see list nil in the strategy list of symbol from). Let

$$t_3 = \text{seriespos}_{(2)|(0)}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}), \text{cons}_{(1)|\text{nil}}(0_{\text{nil}|\text{nil}}, \text{from}_{\text{nil}|(1\ 0)}(s_{\text{nil}|(1)}(0_{\text{nil}|\text{nil}}))))$$

we have $ADP_l(t_3) = \emptyset$, i.e. position 2.2 is again demanded by l but it is not an active position (see the absence of index 2 or -2 in the memoizing and strategy lists of symbol `cons`). Finally, let

$$t_4 = \text{seriespos}_{(2)|(0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}), \\ \text{cons}_{(1\ -2)|nil}(\mathbf{0}_{nil|nil}, \mathbf{from}_{nil|(1\ 0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}))))$$

we have $ADP_l(t_4) = \{2.2\}$.

When different active demanded positions are available in the set $ADP_l(s)$, we use an ordering \leq_s which is based on the user's annotations to select the position (see the use of \min_{\leq_s} below). Given a term $s \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$, the total ordering \leq_s between active positions of s is defined as (1) $\Lambda \leq_s p$ for all $p \in \mathcal{Pos}_A(s)$; (2) if $i.p, i.q \in \mathcal{Pos}_A(s)$ and $p \leq_{s|i} q$, then $i.p \leq_s i.q$; and (3) if $i.p, j.q \in \mathcal{Pos}_A(s)$, $i \neq j$, and i (or $-i$) appears before j (or $-j$) in $\text{active}(\text{root}(s))$, then $i.p \leq_s j.q$. Now, we are able to define the set of demanded positions which would be considered for reduction. We define the set $OD_{\mathcal{R}}(s)$ of on-demand positions of a term $s \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$ w.r.t. TRS \mathcal{R} as follows:

$$\text{if } ADP_{\mathcal{R}}(s) = \emptyset \text{ then } OD_{\mathcal{R}}(s) = \emptyset \text{ else } OD_{\mathcal{R}}(s) = \{\min_{\leq_s}(ADP_{\mathcal{R}}(s))\}$$

Example 9 Consider $l = \text{seriespos}(\mathbf{s}(N), \text{cons}(X, \text{cons}(Y, Z)))$ from Example 4 together with an extra function symbol `inf` and the term

$$t = \text{seriespos}_{(-1\ 2)|(0)}(\mathbf{inf}_{nil|(0)}, \\ \text{cons}_{(-1\ -2)|nil}(\mathbf{0}_{nil|nil}, \mathbf{from}_{nil|(1\ 0)}(\mathbf{0}_{nil|nil})))$$

We have $OD_{\{l\}}(t) = \{1\}$ where $ADP_{\{l\}}(t) = \{1, 2.2\}$, since index -1 appears before index 2 in the memoizing list of symbol `seriespos` and, hence, $1 \leq_t 2.2$.

We use symbols \bar{f} to mark non-evaluable positions, which helps the evaluation of a demanded position to come back to the position which demanded the evaluation. Given a term $t \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$ and a position $p \in \mathcal{Pos}(t)$, $\text{mark}(t, p)$ is the term s with all symbols above p (except the root) marked as non-evaluable, in symbols $\mathcal{Pos}(s) = \mathcal{Pos}(t)$ and $\forall q \in \mathcal{Pos}(t)$, if $\Lambda < q < p$ and $\text{root}(t|_q) = f_{L_1|L_2}$, then $\text{root}(s|_q) = \bar{f}_{L_1|L_2}$, otherwise $\text{root}(s|_q) = \text{root}(t|_q)$.

Example 10 Consider the program of Example 4 and the term $t =$

$$\text{seriespos}_{(1\ 2)|(0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}), \\ \text{cons}_{(1\ -2)|nil}(\mathbf{0}_{nil|nil}, \mathbf{from}_{nil|(1\ 0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}))))$$

We have that $\text{mark}(t, 2.2) =$

$$\text{seriespos}_{(1\ 2)|(0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}), \\ \overline{\text{cons}}_{(1\ -2)|nil}(\mathbf{0}_{nil|nil}, \mathbf{from}_{nil|(1\ 0)}(\mathbf{s}_{nil|(1)}(\mathbf{0}_{nil|nil}))))$$

Given a TRS \mathcal{R} and an E -strategy map φ , we formulate the on-demand strategy via the $eval_{\mathcal{R}}^{\varphi}$ function, which returns the set of terms achievable from a given term through the strategy map φ . In the following definition, the symbol $@$ denotes appending an element at the end of a list.

Definition 1 *Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and an arbitrary E -strategy map φ for \mathcal{F} , we define:*

$$eval_{\mathcal{R}}^{\varphi}(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle \varphi(t), \Lambda \rangle \xrightarrow{\sharp!}_{\varphi} \langle s, \Lambda \rangle\}$$

The binary relation $\xrightarrow{\sharp}_{\varphi, \mathcal{R}}$ on $\mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp}) \times \mathbb{N}_{+}^{*}$ is defined as follows: $\langle t, p \rangle \xrightarrow{\sharp}_{\varphi, \mathcal{R}} \langle s, q \rangle$ if and only if $p \in Pos(t)$ and either

1. $t|_p = f_{L|nil}(t_1, \dots, t_k)$, $s = t$ and $p = q.i$ for some i ; or
2. $t|_p = f_{L_1|i:L_2}(t_1, \dots, t_k)$, $i > 0$, $s = t[f_{L_1@i|L_2}(t_1, \dots, t_k)]_p$ and $q = p.i$; or
3. $t|_p = f_{L_1|-i:L_2}(t_1, \dots, t_k)$, $i > 0$, $s = t[f_{L_1@-i|L_2}(t_1, \dots, t_k)]_p$ and $q = p$; or
4. $t|_p = f_{L_1|0:L_2}(t_1, \dots, t_k) = \sigma(l')$, $erase(l') = l$, $s = t[\sigma(\varphi(r))]_p$ for some $l \rightarrow r \in R$ and substitution σ , $q = p$; or
5. $t|_p = f_{L_1|0:L_2}(t_1, \dots, t_k)$, $erase(t|_p)$ is not a redex, $OD_{\mathcal{R}}(t|_p) = \emptyset$, $s = t[f_{L_1|L_2}(t_1, \dots, t_k)]_p$, and $q = p$; or
6. $t|_p = f_{L_1|0:L_2}(t_1, \dots, t_k)$, $erase(t|_p)$ is not a redex, $OD_{\mathcal{R}}(t|_p) = \{p'\}$, $s = t[mark(t|_p, p')]_p$, $q = p.p'$; or
7. $t|_p = \overline{f}_{L_1|L_2}(t_1, \dots, t_k)$, $s = t[f_{L_1|L_2}(t_1, \dots, t_k)]_p$ and $p = q.i$ for some i .

Case 1 denotes that no more annotations are provided and the evaluation is completed. Case 2 indicates that a positive argument index is provided and the evaluation proceeds by selecting the subterm at this argument (note that the index is stored). Case 3 only stores the negative index for further use. Cases 4, 5, and 6 consider the attempt to match the term against the left-hand sides of the program rules. Case 4 applies if the considered (unlabeled) subterm is a redex (which is, then, contracted). If the subterm is not a redex, cases 5 and 6 are considered (possibly involving some on-demand evaluation). We use the lists of indices labeling the symbols for fixing the concrete positions on which it is safe to allow on-demand evaluations; in particular, the first (memoizing) list is crucial for achieving this (by means of the function *active* and the order \leq_s used in the definition of the set $OD_{\mathcal{R}}(s)$ of on-demand positions of a term s). Case 5 applies if no demanded evaluation is allowed (or required). Case 6 applies if the on-demand evaluation of the subterm $t|_{p.p'}$ is required, i.e. $OD_{\mathcal{R}}(t|_p) = \{p'\}$. In this case, the symbols lying on the path from $t|_p$ to $t|_{p.p'}$ (excluding the ending ones) are overlined. Then, the evaluation process continues on term $t|_{p.p'}$ (with the overlined symbols above it). Once the evaluation of $t|_{p.p'}$ is completed, the

Figure 1: Sketch of the on-demand evaluation of term $\text{pi}(2)$.

$$\begin{aligned}
& \langle \text{pi}_{\text{nil}|(1\ 0)}(\text{s}_{\text{nil}|(1)}(\text{s}_{\text{nil}|(1)}(\text{0}_{\text{nil}|(1)}))), \Lambda \rangle \\
& \xrightarrow{\#}_{\varphi}^* \langle \text{seriespos}_{(1\ 2)|(0)}(\text{s}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{0}_{\text{nil}|(1)})), \\
& \quad \text{cons}_{(1\ -2)|\text{nil}}(\text{0}_{\text{nil}|(1)}), \\
& \quad \text{from}_{\text{nil}|(1\ 0)}(\text{s}_{\text{nil}|(1)}(\text{0}_{\text{nil}|(1)}))), \Lambda \rangle \\
& \xrightarrow{\#}_{\varphi} \langle \text{seriespos}_{(1\ 2)|(0)}(\text{s}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{0}_{\text{nil}|(1)})), \\
& \quad \text{cons}_{(1\ -2)|\text{nil}}(\text{0}_{\text{nil}|(1)}), \\
& \quad \text{from}_{\text{nil}|(1\ 0)}(\text{s}_{\text{nil}|(1)}(\text{0}_{\text{nil}|(1)}))), 2.2 \rangle \\
& \xrightarrow{\#}_{\varphi}^* \langle \text{rcons}_{(1\ 2)|\text{nil}}(\text{posrecip}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{0}_{\text{nil}|(1)})), \\
& \quad \text{rcons}_{(1\ 2)|\text{nil}}(\text{negrecip}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{s}_{(1)|\text{nil}}(\text{0}_{\text{nil}|(1)}))), \\
& \quad \text{rnil}_{\text{nil}|(1)}), \Lambda \rangle
\end{aligned}$$

only possibility is the repeated (but possibly idle) application of steps issued according to the last case 7 which sends the evaluation process back to position p (which originated the on-demand evaluation) using overlined symbols \bar{f} .

Example 11 *Continuing Example 4. Figure 1 provides a sketch of the evaluation sequence for the term $\text{pi}(\text{s}(\text{s}(0)))$ where the on-demand evaluation step associated to symbol `seriespos` is shown. Roughly speaking, the following term rewriting sequence is performed:*

```

pi(2)
→ seriespos(2, from(0))
→ seriespos(2, cons(0, from(1)))
→ seriespos(2, cons(0, cons(1, from(2))))
→ rcons(posrecip(1), seriesneg(1, from(2)))
→ rcons(posrecip(1), seriesneg(1, cons(2, from(3))))
→ rcons(posrecip(1), seriesneg(1, cons(2, cons(3, from(4)))))
→ rcons(posrecip(1), rcons(negrecip(3), seriesneg(0, from(4))))
→ rcons(posrecip(1), rcons(negrecip(3), rnil))

```

4 The Program Transformation

In the following, we formalize a program transformation which translates OBJ programs with arbitrary indices into OBJ programs with positive indices alone. We first explain the awkward points associated to the evaluation with negative indices in order to discern how to transform these indices into positive ones.

Example 12 *Consider the following OBJ program which is mainly borrowed from a CafeOBJ program in [23] where we consider negative indices for `cons`:*

```

obj Ex3rd is
  sorts Nat LNat .
  op 0      : -> Nat .
  op s      : Nat -> Nat      [strat (1)] .
  op nil    : -> LNat .

```

```

op cons : Nat LNat -> LNat [strat (1 -2)] .
op from : Nat -> LNat      [strat (1 0)] .
op 3rd  : LNat -> Nat      [strat (1 0)] .
vars X Y Z : Nat . var Zs : LNat .
eq 3rd(cons(X,cons(Y,cons(Z,Zs)))) = Z .
eq from(X) = cons(X,from(s(X))) .
endo

```

Let us introduce an auxiliary function symbol `inf`, which returns an infinite sequence of symbols `s` (note that the evaluation of the call `inf` is nonterminating since the strategy annotation for the constructor `s` above is 1):

```

op inf : -> Nat .
eq inf = s(inf) .

```

Consider the term $t = 3rd(cons(0,cons(inf,cons(s(0),nil))))$. The on-demand E-evaluation of t terminates and returns `s(0)`, since the term `inf` is not under a positive (reducible) position nor is demanded by the rule defining `3rd`.

Let us consider a naïve approach for transforming negative indices into positive indices which duplicates the symbols containing negative indices into new symbols containing the positive counterparts (as in the program transformation showed in [2] for approximating termination). The raw application of such program transformation to the previous example delivers the following rules:

```

eq 3rd(cons(X,Zs)) = 3rd(cons+2(X,Zs)) .
eq 3rd(cons+2(X,cons(Y,Zs))) = 3rd(cons+2(X,cons+2(Y,Zs))) .
eq 3rd(cons+2(X,cons+2(Y,cons(Z,Zs)))) = Z .

```

together with the following definitions for symbols `cons` and `cons+2`:

```

op cons : Nat LNat -> LNat [strat (1)] .
op cons+2 : Nat LNat -> LNat [strat (2)] .

```

However, the evaluation of the previous call t w.r.t. this new program enters in an infinite reduction sequence, since the term `inf` will be under a positive (reducible) position after transforming the leftmost symbol `cons` of t into `cons+2`.

In order to avoid this problem, the solution is to remove all positive annotations of symbol `cons`, i.e. we obtain⁵:

```

op cons : Nat LNat -> LNat [strat (0)] .
op cons+2 : Nat LNat -> LNat [strat (2)] .

```

However, occurrences of symbol `cons` appearing at positive positions of the original program rules do not behave correctly now, e.g. `3rd(cons(inf,from(s(0)))` does not have an infinite reduction sequence as in the original program because the subterm `inf` does not appear now under a positive position.

Nevertheless, we can define a new symbol `consroot` which behaves as the original symbol `cons` (i.e. the positive indices of `cons`), and consistently rename the symbols in the TRS by rewriting special symbols `quote` and `ondemand` before evaluating a given term at a positive (or reducible) position or at a negative

⁵Since Maude does not accept the empty strategy list `nil`, we give the strategy list (0) to symbol `cons`. This is not a problem since `cons` is a constructor symbol.

(or on-demand) position, respectively. Moreover, we introduce a new rule for translating cons_{root} back to cons . We finally obtain the program:

```
obj Ex3rdA is
  sorts Nat LNat .
  op 0      : -> Nat .
  op s      : Nat -> Nat [strat (1)] .
  op nil    : -> LNat .
  op cons   : Nat LNat -> LNat [strat (0)] .
  op consroot : Nat LNat -> LNat [strat (1 0)] .
  op cons+2 : Nat LNat -> LNat [strat (2)] .
  op from   : Nat -> LNat [strat (1 0)] .
  op 3rd    : LNat -> Nat [strat (1 0)] .
  op quote ondemand : Nat -> Nat [strat (0)] .
  op quote ondemand : LNat -> LNat [strat (0)] .
  vars X Y Z : Nat . var Zs : LNat .
  eq 3rd(cons(X,Zs)) = 3rd(cons+2(X,ondemand(Zs))) .
  eq 3rd(cons+2(X,cons(Y,Zs)))
    = 3rd(cons+2(X,cons+2(Y,ondemand(Zs)))) .
  eq 3rd(cons+2(X,cons+2(Y,cons(Z,Zs)))) = quote(Z) .
  eq from(X) = quote(cons(X,from(s(X)))) .
  eq quote(3rd(Zs)) = 3rd(quote(Zs)) .
  eq quote(s(X)) = s(quote(X)) .
  eq quote(0) = 0 .
  eq quote(from(X)) = from(quote(X)) .
  eq quote(cons(X,Zs)) = consroot(quote(X),Zs) .
  eq quote(nil) = nil .
  eq consroot(X,Zs) = cons(X,Zs) .
  eq ondemand(3rd(Zs)) = 3rd(quote(Zs)) .
  eq ondemand(s(X)) = s(X) .
  eq ondemand(0) = 0 .
  eq ondemand(from(X)) = from(quote(X)) .
  eq ondemand(cons(X,Zs)) = cons(X,Zs) .
  eq ondemand(nil) = nil .
endo
```

However, in order to speed up the evaluation, we use more f_{+i} symbols from the root of the left hand side traversing the on-demand path (see Section 4.2).

We define the complete transformation for a TRS by two different transformers which tackle the two difficulties described above. That is, the transformation starts by applying the first transformer which introduces symbols f_{root} and symbols quote and ondemand in order to set the stage for the second transformer. Then, the second transformer is applied iteratively, which turns the negative indices into positive ones by introducing symbols f_{+i} . This transformer finally removes all negative annotations together with positive annotations of conflictive symbols (such as symbol cons in the previous example).

4.1 Transformation for fixing rules

Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, and φ be an E -strategy map. We define the set of positions which are active (but not reducible) of a term $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ as

$\mathcal{P}os_{A-P}(t) = \mathcal{P}os_A(t) - \mathcal{P}os_P(t)$. Given a strategy map φ , φ_+ denotes the result of removing all negative indices from φ .

Let us define the set of symbols of the original TRS at positions active but not reducible which have positive indices which can be lost as $\mathcal{F}_R^\varphi = \{f \in \mathcal{F} \mid ar(f) > 0 \wedge \varphi_+(f) \neq nil \wedge \exists l \in L(\mathcal{R}) : \mathcal{P}os_f(l) \cap \mathcal{P}os_{A-P}(\varphi(l)) \neq \emptyset\}$. Note that if \mathcal{R} is a CS, then $\mathcal{F}_R^\varphi \subseteq \mathcal{C}$. In the following, we define the two sets of auxiliary rules $Quote_{\mathcal{R}, \mathcal{F}_R^\varphi}$ and $OnDemand_{\mathcal{R}}$ to appropriately (head)-normalize terms. Intuitively, **quote** translates a symbol $f \in \mathcal{F}_R^\varphi$ at a positive position into the new symbol f_{root} and **ondemand** uses **quote** for function-rooted terms which are evaluated on-demand.

$$Quote_{\mathcal{R}, \mathcal{F}_R^\varphi} = \bigcup_{f \in \mathcal{F}} \begin{cases} \mathbf{quote}(f(\bar{x})) \rightarrow f_{root}(\rho_f(\bar{x})) & \text{if } f \in \mathcal{F}_R^\varphi \\ \mathbf{quote}(f(\bar{x})) \rightarrow f(\rho_f(\bar{x})) & \text{if } f \notin \mathcal{F}_R^\varphi \end{cases}$$

$$OnDemand_{\mathcal{R}} = \bigcup_{f \in \mathcal{F}} \begin{cases} \mathbf{ondemand}(f(\bar{x})) \rightarrow f(\bar{x}) & \text{if } f \in \mathcal{C} \\ \mathbf{ondemand}(f(\bar{x})) \rightarrow f(\rho_f(\bar{x})) & \text{if } f \in \mathcal{D} \end{cases}$$

where $\rho_f(x_i) = \begin{cases} \mathbf{quote}(x_i) & \text{if } (i) \sqsubseteq \varphi^\mathbb{I}(f) \\ x_i & \text{if } (i) \not\sqsubseteq \varphi^\mathbb{I}(f) \end{cases}$

We define the *first transformer* for fixing strategy annotations $\mathcal{R}^\mathbb{I} = (\mathcal{F}^\mathbb{I}, R^\mathbb{I})$ and $\varphi^\mathbb{I}$ as follows: $\mathcal{F}^\mathbb{I} = \mathcal{F} \cup \{f_{root} \mid f \in \mathcal{F}_R^\varphi\} \cup \{\mathbf{quote}, \mathbf{ondemand}\}$, and

$$R^\mathbb{I} = \{l \rightarrow \mathbf{quote}(r) \mid l \rightarrow r \in R\} \cup \{f_{root}(\bar{x}) \rightarrow f(\bar{x}) \mid f \in \mathcal{F}_R^\varphi\} \\ \cup Quote_{\mathcal{R}, \mathcal{F}_R^\varphi} \cup OnDemand_{\mathcal{R}}$$

Also, $\varphi^\mathbb{I}(f) = \varphi(f)$ for all $f \in \mathcal{F}$, $\varphi^\mathbb{I}(f_{root}) = \varphi_+(f)@0$ for all $f \in \mathcal{F}_R^\varphi$, and $\varphi^\mathbb{I}(\mathbf{quote}) = \varphi^\mathbb{I}(\mathbf{ondemand}) = (0)$.

Example 13 Consider the TRS \mathcal{R} and the E-strategy map φ of Example 12. The TRS $\mathcal{R}^\mathbb{I}$ together with $\varphi^\mathbb{I}$ is:

```
obj Ex3rdI is
  sorts Nat LNat .
  op 0      : -> Nat .
  op s      : Nat -> Nat [strat (1)] .
  op nil    : -> LNat .
  op cons   : Nat LNat -> LNat [strat (1 -2)] .
  op consroot : Nat LNat -> LNat [strat (1 0)] .
  op from   : Nat -> LNat [strat (1 0)] .
  op 3rd    : LNat -> Nat [strat (1 0)] .
  op quote ondemand : Nat -> Nat [strat (0)] .
  op quote ondemand : LNat -> LNat [strat (0)] .
  vars X Y Z : Nat . var Zs : LNat .
  eq 3rd(cons(X, cons(Y, cons(Z, Zs)))) = quote(Z) .
  eq from(X) = quote(cons(X, from(s(X)))) .
  eq quote(3rd(Zs)) = 3rd(quote(Zs)) .
  eq quote(s(X)) = s(quote(X)) .
  eq quote(0) = 0 .
  eq quote(from(X)) = from(quote(X)) .
  eq quote(cons(X, Zs)) = consroot(quote(X), Zs) .
  eq quote(nil) = nil .
```

```

eq consroot(X,Zs) = cons(X,Zs) .
eq ondemand(3rd(Zs)) = 3rd(quote(Zs)) .
eq ondemand(s(X)) = s(X) .
eq ondemand(0) = 0 .
eq ondemand(from(X)) = from(quote(X)) .
eq ondemand(cons(X,Zs)) = cons(X,Zs) .
eq ondemand(nil) = nil .
endo

```

4.2 Transformation for eliminating negative indices

We formulate the *transformer* for switching negative indices into positive ones. Intuitively, we transform a rule $l \rightarrow r$ with a position p which is active but not reducible into the rules $l[x]_p \rightarrow l'[x]_p$ and $l' \rightarrow r$, where each symbol f in l from the root to p has been replaced in l' by the new symbol f_{+i} whose strategy list is (i) (or (i 0) if it is a function symbol).

Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, and φ be an E -strategy map. Given $l \in L(\mathcal{R})$, we define the set of positions of a lhs l which are active but not reducible as $\mathcal{I}^\varphi(l) = \text{Pos}_{A-P}(\varphi(l)) \cap \text{Pos}_{\mathcal{F}}(l)$. Assume that $\mathcal{I}^\varphi(l) \neq \emptyset$ for a rule $l \rightarrow r \in \mathcal{R}$, the position to be transformed is $p.i = \max_{\leq \varphi(l)}(\mathcal{I}^\varphi(l))$ for $p.i \in \text{Pos}(l)$ and $i \in \mathbb{N}$ (note that $\Lambda \notin \mathcal{I}^\varphi(l)$ by definition), and the set of symbols involved in the transformation are f^Λ, \dots, f^p such that $\text{root}(l|_q) = f^q \in \mathcal{F}$ for $q \leq p$. Then, the *transformer for eliminating negative indices* $\mathcal{R}^{\text{neg}} = (\mathcal{F}^{\text{neg}}, R^{\text{neg}})$ and φ^{neg} is as follows: $\mathcal{F}^{\text{neg}} = \mathcal{F} \cup \mathcal{F}_p^+$ where $\mathcal{F}_p^+ = \{f_{+j_\Lambda}^\Lambda, \dots, f_{+j_p}^p\}$ such that $j_\Lambda, \dots, j_p \in \mathbb{N}$ and $q.j_q \leq p.i$ for $q \leq p$. Also,

$$\begin{aligned}
R^{\text{neg}} = R - \{l \rightarrow r\} \cup & \{l[y]_{p.i} \rightarrow l'[\text{ondemand}(y)]_{p.i}, l' \rightarrow r\} \\
& \cup \{\text{quote}(f_{+j_q}^q(\bar{x})) \rightarrow f_{+j_q}^q(\bar{x}) \mid f_{+j_q}^q \in \mathcal{F}_p^+\} \\
& \cup \{\text{ondemand}(f_{+j_q}^q(\bar{x})) \rightarrow f_{+j_q}^q(\bar{x}) \mid f_{+j_q}^q \in \mathcal{F}_p^+\}
\end{aligned}$$

where y is a fresh variable and l' is obtained from l such that $\forall q \in \text{Pos}(l')$:

$$\text{root}(l'|_q) = \begin{cases} f_{+j_q}^q & \text{if } q \leq p \\ \text{root}(l|_q) & \text{otherwise} \end{cases}$$

We let $\varphi^{\text{neg}}(f) = \varphi(f)$ for $f \in \mathcal{F}$ and

$$\varphi^{\text{neg}}(f_{+j}) = \begin{cases} (j \ 0) & \text{if } (-j \ 0) \sqsubseteq \varphi(f) \vee (j \ 0) \sqsubseteq \varphi(f) \\ (j) & \text{otherwise} \end{cases}$$

Example 14 Consider the TRS $\mathcal{R} = \mathcal{R}^\mathbb{I}$ and the E -strategy map $\varphi = \varphi^\mathbb{I}$ in Example 13. The application of the transformer, i.e. the TRS \mathcal{R}^{neg} together with φ^{neg} , is:

```

obj Ex3rdINeg is
  sorts Nat LNat .
  op 0      : -> Nat .
  op s      : Nat -> Nat      [strat (1)] .
  op nil    : -> LNat .

```

```

op cons : Nat LNat -> LNat      [strat (1 -2)] .
op consroot : Nat LNat -> LNat  [strat (1 0)] .
op cons+2 : Nat LNat -> LNat   [strat (2)] .
op from : Nat -> LNat          [strat (1 0)] .
op 3rd : LNat -> Nat           [strat (1 0)] .
op 3rd+1 : LNat -> Nat        [strat (1 0)] .
op quote ondemand : Nat -> Nat [strat (0)] .
op quote ondemand : LNat -> LNat [strat (0)] .
vars X Y Z : Nat . var Zs : LNat .
eq 3rd(cons(X,cons(Y,Zs)))
  = 3rd+1(cons+2(X,cons+2(Y,ondemand(Zs)))) .
eq 3rd+1(cons+2(X,cons+2(Y,cons(Z,Zs)))) = quote(Z) .
eq from(X) = quote(cons(X,from(s(X)))) .
eq quote(3rd(Zs)) = 3rd(quote(Zs)) .
eq quote(3rd+1(Zs)) = 3rd+1(Zs) .
eq quote(s(X)) = s(quote(X)) .
eq quote(0) = 0 .
eq quote(from(X)) = from(quote(X)) .
eq quote(cons(X,Zs)) = consroot(quote(X),Zs) .
eq quote(cons+2(X,Zs)) = cons+2(X,Zs) .
eq quote(nil) = nil .
eq consroot(X,Zs) = cons(X,Zs) .
eq ondemand(3rd(Zs)) = 3rd(quote(Zs)) .
eq ondemand(s(X)) = s(X) .
eq ondemand(0) = 0 .
eq ondemand(from(X)) = from(quote(X)) .
eq ondemand(cons(X,Zs)) = cons(X,Zs) .
eq ondemand(nil) = nil .
endo

```

Note the relevant changes in the rule for symbol 3rd:

```

eq 3rd(cons(X,cons(Y,Zs)))
  = 3rd+1(cons+2(X,cons+2(Y,ondemand(Zs)))) .
eq 3rd+1(cons+2(X,cons+2(Y,cons(Z,Zs)))) = quote(Z) .

```

The second transformation process starts from \mathcal{R}^{I} and φ^{I} and applies as many transformation steps \mathcal{R}^{neg} and φ^{neg} for removing negative indices as necessary to obtain $\mathcal{R}' = (\mathcal{F}', R')$ and φ' such that no negative index is necessary, i.e. $\mathcal{I}^{\varphi'}(l) = \emptyset$ for all $l \in L(\mathcal{R}')$.

The final TRS $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} is obtained as $\mathcal{F}^{\text{III}} = \mathcal{F}'$, $R^{\text{III}} = R'$, and $\varphi^{\text{III}}(f) = \varphi'_+(f)$ for $f \in \mathcal{F}' - \mathcal{F}_{\mathcal{R}}^{\varphi}$ and $\varphi^{\text{III}}(f) = \text{nil}^6$ for $f \in \mathcal{F}_{\mathcal{R}}^{\varphi}$.

Example 15 Continuing with Example 14. The final TRS \mathcal{R}^{III} together with φ^{III} is:

```

obj Ex3rdII is
  sorts Nat LNat .
  op 0 : -> Nat .
  op s : Nat -> Nat      [strat (1)] .
  op nil : -> LNat .
  op cons : Nat LNat -> LNat [strat (0)] .

```

⁶Since Maude does not accept the empty strategy list *nil*, we give the strategy list (0).

```

op consroot : Nat LNat -> LNat [strat (1 0)] .
op cons+2 : Nat LNat -> LNat [strat (2)] .
op from : Nat -> LNat [strat (1 0)] .
op 3rd : LNat -> Nat [strat (1 0)] .
op 3rd+1 : LNat -> Nat [strat (1 0)] .
op quote ondemand : Nat -> Nat [strat (0)] .
op quote ondemand : LNat -> LNat [strat (0)] .
vars X Y Z : Nat . var Zs : LNat .
eq 3rd(cons(X,Zs))
  = 3rd+1(cons+2(X,ondemand(Zs))) .
eq 3rd+1(cons+2(X,cons(Y,Zs)))
  = 3rd+1(cons+2(X,cons+2(Y,ondemand(Zs)))) .
eq 3rd+1(cons+2(X,cons+2(Y,cons(Z,Zs)))) = quote(Z) .
eq from(X) = quote(cons(X,from(s(X)))) .
eq quote(3rd(Zs)) = 3rd(quote(Zs)) .
eq quote(3rd+1(Zs)) = 3rd+1(Zs) .
eq quote(s(X)) = s(quote(X)) .
eq quote(0) = 0 .
eq quote(from(X)) = from(quote(X)) .
eq quote(cons(X,Zs)) = consroot(quote(X),Zs) .
eq quote(cons+2(X,Zs)) = cons+2(X,Zs) .
eq quote(nil) = nil .
eq consroot(X,Zs) = cons(X,Zs) .
eq ondemand(3rd(Zs)) = 3rd(quote(Zs)) .
eq ondemand(s(X)) = s(X) .
eq ondemand(0) = 0 .
eq ondemand(from(X)) = from(quote(X)) .
eq ondemand(cons(X,Zs)) = cons(X,Zs) .
eq ondemand(nil) = nil .
endo

```

We would like to emphasize that the transformation defined in this paper is able to deal with the general case where more than one negative annotation exist for the same function symbol and different demandness paths are possible; which are just ordered using the ordering \leq_s between positive and negative annotations and managed by using symbols f_{+i} which traverse the path from the root. This is not illustrated in our main example due to space restrictions, though we give some hints in the following example.

Example 16 Consider the following program rule:

$$\min(s(X), 0, s(0), s(s(Y))) = 0$$

with the strategy map $\varphi(\min) = (-3 \ -2 \ 1 \ -4 \ 0)$, $\varphi(s) = (-1)$, and $\varphi(0) = \text{nil}$.

The transformation of this rule produces (without considering symbol `quote`):

$$\begin{aligned}
\min(s(X), Z, \bar{W}, Y) &= \min_{+3}(s(X), Z, \text{ondemand}(W), Y) \\
\min_{+3}(s(X), Z, s(W), Y) &= \min_{+3}(s(X), Z, s_{+1}(\text{ondemand}(W)), Y) \\
\min_{+3}(s(X), Z, s_{+1}(0), Y) &= \min_{+2}(s(X), \text{ondemand}(Z), s(0), Y) \\
\min_{+2}(s(X), 0, s(0), Y) &= \min_{+4}(s(X), 0, s(0), \text{ondemand}(Y)) \\
\min_{+4}(s(X), 0, s(0), s(Y)) &= \min_{+4}(s(X), 0, s(0), s_{+1}(\text{ondemand}(Y))) \\
\min_{+4}(s(X), 0, s(0), s_{+1}(s(Y))) &= \text{quote}(0)
\end{aligned}$$

and the strategy map $\varphi^{\text{III}}(\min) = (1 \ 0)$, $\varphi^{\text{III}}(\min_{+2}) = (2 \ 0)$, $\varphi^{\text{III}}(\min_{+3}) = (3 \ 0)$, $\varphi^{\text{III}}(\min_{+4}) = (4 \ 0)$, $\varphi^{\text{III}}(s) = \text{nil}$, $\varphi^{\text{III}}(s_{+1}) = (1)$, and

$\varphi^{\text{III}}(0) = \text{nil}$. This transformed program reproduces the ordering between the different on-demand paths of the left-hand side.

4.3 Properties

In the following, we establish the main results of the program transformation. We define a *standard E-strategy map* φ as an *E-strategy map* where only one index 0 is allowed at the end of each strategy list. Given a strategy map φ for \mathcal{F} , we say that a TRS $\mathcal{R} = (\mathcal{F}, R)$ is φ -terminating if, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, there is no infinite $\xrightarrow{\#}_{\varphi, \mathcal{R}}$ -rewrite sequence starting from $\langle \varphi(t), \Lambda \rangle$. Note that termination of the program `pi` of Example 4 is formally proved in Appendix C using the technique of [2]. Furthermore, Example 12 can be also proved φ -terminating using the technique developed in [2]. Termination is preserved by the transformation.

Theorem 1 (Termination) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. \mathcal{R} is φ -terminating iff \mathcal{R}^{III} is φ^{III} -terminating.*

In the following theorem we prove completeness of the transformation, i.e. that the transformation preserves normal forms.

Theorem 2 (Completeness) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that \mathcal{R} is φ -terminating. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in \text{eval}_{\mathcal{R}}^{\varphi}(t)$, then $s \in \text{eval}_{\mathcal{R}^{\text{III}}}^{\varphi^{\text{III}}}(\text{quote}(t))$.*

Correctness of the transformation is also proved without any condition on termination of the TRS.

Theorem 3 (Correctness) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in \text{eval}_{\mathcal{R}^{\text{III}}}^{\varphi^{\text{III}}}(\text{quote}(t))$, then $s \in \text{eval}_{\mathcal{R}}^{\varphi}(t)$.*

5 Experiments

A prototype implementation of the transformation proposed in this paper has been integrated into the OnDemandOBJ system [5]. The system is publicly available at

<http://www.dsic.upv.es/users/elp/soft.html>

Tables 1, and 2, show the runtimes⁷ in milliseconds and the number of evaluation steps of the benchmarks for the different OBJ-family systems. The OnDemandOBJ interpreter is the on-demand prototype interpreter of the on-demand evaluation of [2]. CafeOBJ⁸ is developed in Lisp at the Japan Advanced Inst. of Science and Technology (JAIST); OBJ3⁹, also written in Lisp, is maintained

⁷The average of 10 executions measured in an AMD XP machine running `redhat 9.0`.

⁸Available at <http://www.ldl.jaist.ac.jp/Research/CafeOBJ/system.html>.

⁹Available at <http://www.kindsoftware.com/products/opensource/obj3/OBJ3/>.

by the University of California at San Diego; *Maude*¹⁰ is developed in C++ and maintained by the Department of Computer Science Lab at the University of Illinois, Urbana-Champaign. *OBJ3* and *Maude* provide only computations with positive annotations whereas *CafeOBJ* provides computations with negative annotations as well, using the on-demand evaluation of [23, 22]. *OnDemandOBJ* computes with negative annotations using the on-demand evaluation of [2]. Note that *CafeOBJ* and *OBJ3* implement sharing of variables whereas *Maude* and *OnDemandOBJ* do not; thus, the number of evaluation steps in Table 2 is pairwise equivalent: *CafeOBJ* and *OBJ3* in one hand and *Maude* and *OnDemandOBJ* in the other hand. The mark *overflow* in Table 2 indicates that 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.

The benchmark *pi_noneg* consists of the application of the program transformation described in this paper to program *pi* of Example 4. Table 1 compares the evaluation of expression *pi(square(square(4)))* using *pi* and *pi_noneg*. Note that the right input expression for the program *pi_noneg* is *quote(pi(square(square(4))))*. It witnesses that negative annotations are extremely useful in practice and that the program transformation enables the execution of negatively annotated programs in all OBJ implementations. On the other hand, Table 1 also evidences that the implementation of the on-demand evaluation strategy in other systems such as *Maude* 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. Every *k*-ary symbol *f* is given a strategy (1 2 \dots *k* 0) (this corresponds to *default* strategies in *Maude*). 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 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 [7]). The benchmark *msquare_neg* is similar to *msquare_eager*, though *canonical* arbitrary strategies are provided: now (from left-to-right), the *i*-th argument of a defined symbol *f* is annotated 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 [7]). The benchmark *msquare_noneg* represents the application of the program transformation to *msquare_neg*. Note that the right input expressions for the program *msquare_noneg* are preceded by the symbol *quote*, i.e. *quote(minus(0,square(square(5))))* and *quote(minus(square(square(5)),square(square(4))))*. Then, for instance, program *msquare_neg* runs in less time and requires a smaller number of rewrite

¹⁰Available at <http://maude.cs.uiuc.edu/current/system/>.

Table 1: Execution of call `pi(square(square(4)))`

ms./rewrites	pi	pi_nonneg
OnDemandOBJ	20/1087	6390/402917
CafeOBJ	20/1087	6770/402917
OBJ3	<i>unavailable</i>	<i>overflow</i>
Maude	<i>unavailable</i>	100/402917

Table 2: Execution of terms `minus(0,square(square(5)))` and `minus(square(square(5)),square(square(4)))`

ms./rewrites	msquare_eager	msquare_apt	msquare_neg	msquare_nonneg
OnDemandOBJ	10/ 715	18/ 1640	0/ 1	0/ 4
	23/ 1287	30/ 2628	28/ 2628	5080/292966
CafeOBJ	10/ 715	12/ 715	0/ 1	0/ 4
	25/ 1287	27/ 1287	26/ 1287	320/ 27726
OBJ3	0/ 715	<i>overflow</i>	<i>unavailable</i>	0/ 4
	10/ 1287	<i>overflow</i>	<i>unavailable</i>	<i>overflow</i>
Maude	0/ 715	0/ 1640	<i>unavailable</i>	0/ 4
	0/ 1287	3/ 2628	<i>unavailable</i>	50/292966

steps than `msquare_eager` or `msquare_apt`, which do not include negative annotations. Note the difference in the number of rewrite steps of benchmarks `msquare_eager` and `msquare_apt` for the Maude and OnDemandOBJ systems, which is due to the absence of variable sharing. Moreover, note that the program transformation is also very useful since execution of the expression `minus(0,square(square(5)))` is improved. The OBJ source programs can be found at the Appendix B.

6 Conclusions

The paper presents a contribution to the extension of evaluation strategies for functional languages of the OBJ family with evaluation on demand, thereby introducing a flavour of laziness into such languages. Our proposal is based on a program transformation for OBJ programs which achieves correctness and works well in current OBJ interpreters. The main technical results of this work are as follows:

- The proposed transformation preserves the termination of the original program which uses (positive and) negative annotations. That is, if the original program terminates under the evaluation strategy of [2], then the transformed program terminates also.
- Correct and completeness of the transformation holds w.r.t. the semantics of strategy annotations given in [2]. That is, the semantics of input

expressions in the original program (under the on-demand E -strategy of [2]) and in the transformed program (under the E -strategy) do coincide.

Moreover, our transformation is useful both for

1. making possible the use of arbitrary strategy annotations in languages that (syntactically) allow them but that still do not provide the necessary operational support (e.g., OBJ3).
2. providing a notion of negative strategy annotation (somewhat laziness) for languages that does not allow them (e.g., Maude).

and hence we think that our work contributes to foster the use of OBJ in programming. As future work, we plan to formally determine the overhead associated to the evaluation in the transformed program.

References

- [1] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. Improving on-demand strategy annotations. In Matthias Baaz and Andrei Voronkov, editors, *Proc. 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, volume 2514 of *Lecture Notes in Computer Science*, pages 1–18, Tbilisi, Georgia, 2002. Springer-Verlag, Berlin.
- [2] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. On-demand strategy annotations revisited. Technical Report DSIC-II/18/03, Departamento de Sistemas Informáticos y Computación, UPV, 2003. Available from URL: <http://www.dsic.upv.es/users/elp/papers.html>. This is a revised and improved version of [1].
- [3] M. Alpuente, S. Escobar, and S. Lucas. Correct and complete (positive) strategy annotations for OBJ. In J.L. Giavitto and P.E. Moreau, editors, *Proc. of the 4th International Workshop on Rewriting Logic and its Applications, WRLA 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2002.
- [4] M. Alpuente, S. Escobar, and S. Lucas. On-demand evaluation by program transformation. In J.L. Giavitto and P.E. Moreau, editors, *Proc. of the 4th International Workshop on Rule-Based Programming, RULE 2003*, volume 86.2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2003.
- [5] M. Alpuente, S. Escobar, and S. Lucas. OnDemandOBJ: a laboratory for strategy annotations. In J.L. Giavitto and P.E. Moreau, editors, *Proc. of the 4th International Workshop on Rule-Based Programming, RULE 2003*, volume 86.2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2003.

- [6] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Specialization of Lazy Functional Logic Programs. In *Proc. of the ACM SIGPLAN Conf. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, volume 32, number 12 of *ACM Sigplan Notices*, pages 151–162. ACM Press, New York, 1997.
- [7] S. Antoy and S. Lucas. Demandness in rewriting and narrowing. In M. Comini and M. Falaschi, editors, *Proc. of the 11th Int'l Workshop on Functional and (Constraint) Logic Programming WFLP'02*, volume 76 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2002.
- [8] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [9] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proc. of the 1st International Workshop on Rewriting Logic and its Applications, RLLW 96*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89. Elsevier Sciences Publisher, 1996.
- [10] S. Eker. Term rewriting with operator evaluation strategies. In C. Kirchner and H. Kirchner, editors, *Proc. of the 2nd International Workshop on Rewriting Logic and its Applications, WRLA 98*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2000.
- [11] W. Fokkink, J. Kamperman, and P. Walters. Lazy rewriting on eager machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45–86, 2000.
- [12] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proc. of 12th Annual ACM Symp. on Principles of Programming Languages (POPL'85)*, pages 52–66. ACM Press, New York, 1985.
- [13] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specification over networks –. In *1st International Conference on Formal Engineering Methods*, 1997.
- [14] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.
- [15] R. Loogen, F. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In *Proc. of PLILP'93*, volume 714 of *Lecture Notes in Computer Science*, pages 184–200. Springer-Verlag, Berlin, 1993.

- [16] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1–61, 1998.
- [17] S. Lucas. Termination of on-demand rewriting and termination of obj programs. In *Proc. of 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'01*, pages 82–93. ACM Press, New York, 2001.
- [18] S. Lucas. Lazy rewriting and context-sensitive rewriting. In M. Hanus, editor, *Proc. of the 10th Int'l Workshop on Functional and (Constraint) Logic Programming WFLP'01*, volume 64 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2002.
- [19] S. Lucas. Termination of (Canonical) Context-Sensitive Rewriting. In S. Tiison, editor, *Proc. of 13th International Conference on Rewriting Techniques and Applications, RTA'02*, volume 2378 of *Lecture Notes in Computer Science*, pages 296–310. Springer-Verlag, Berlin, 2002.
- [20] J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *Journal of Logic Programming*, 12(3):191–224, 1992.
- [21] T. Nagaya. *Reduction Strategies for Term Rewriting Systems*. PhD thesis, School of Information Science, Japan Advanced Institute of Science and Technology, March 1999.
- [22] M. Nakamura and K. Ogata. The evaluation strategy for head normal form with and without on-demand flags. In K. Futatsugi, editor, *Proc. of the 3rd International Workshop on Rewriting Logic and its Applications, WRLA 2000*, volume 36 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2001.
- [23] K. Ogata and K. Futatsugi. Operational semantics of rewriting with the on-demand evaluation strategy. In *Proc. of 2000 International Symposium on Applied Computing, SAC'00*, pages 756–763. ACM Press, New York, 2000.
- [24] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.
- [25] H. Zantema. Termination of context-sensitive rewriting. In *Proc. of 8th International Conference on Rewriting Techniques and Applications, RTA'97*, volume 1232 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, Berlin, 1997.

A Proofs

Theorem 1 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. \mathcal{R} is φ -terminating iff $\mathcal{R}^{\mathbb{I}}$ is $\varphi^{\mathbb{I}}$ -terminating.*

Proof The first transformer $\mathcal{R}^{\mathbb{I}}$ preserves termination of the program since only introduces symbols `quote` and `ondemand` which have strategy (0) and symbols f_{root} which keep the same indices that the respective symbol f . For the second transformation, we prove it by induction on the number n of transformation steps \mathcal{R}^{neg} . The case $n = 0$ is trivial. If $n > 0$, then it is not difficult to see that the splitting of rule $l \rightarrow r$ into rules $l[x]_{p.i} \rightarrow l'[\text{ondemand}(x)]_{p.i}$ and $l' \rightarrow r$ does preserve termination since rule $l \rightarrow r$ is applicable iff rule $l' \rightarrow r$ becomes applicable after applying rule $l[x]_{p.i} \rightarrow l'[\text{ondemand}(x)]_{p.i}$. \square

Given two strategy lists $L, L' \in \mathbb{L}$, the restriction of L to L' , denoted as $L_{\downarrow L'}$, is the maximal sublist L'' such that $L'' \sqsubseteq L$ and $L'' \sqsubseteq L'$. In the following, we define three different translations of terms in $\mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ into terms of $\mathcal{T}(\mathcal{F}_{\varphi^{\mathbb{I}}}^{\sharp}, \mathcal{X}_{\varphi^{\mathbb{I}}}^{\sharp})$: without considering extra symbols f_{root} and f_{+i} (translation $t_{\downarrow \varphi^{\mathbb{I}}}$), considering the insertion of symbols f_{root} (translation $pos_{p, \mathcal{F}_{\mathcal{R}}^{\varphi}}(t)$), or considering the insertion of symbols f_{+i} (translation $neg_p(t)$). Note that for all $f \in \mathcal{F}$, $\varphi^{\mathbb{I}}(f) \sqsubseteq \varphi(f)$.

Definition 2 *Let φ and φ' be strategy maps over signature \mathcal{F} . Let $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$. We define the translation of t into a term $\mathcal{T}(\mathcal{F}_{\varphi'}^{\sharp}, \mathcal{X}_{\varphi'}^{\sharp})$ as $t_{\downarrow \varphi'} = s$ where $\text{Pos}(s) = \text{Pos}(t)$ and $\forall q \in \text{Pos}(t). \text{root}(t|_q) = f_{L_1|L_2}$, $\text{root}(s|_q) = f_{L'_1|L'_2}$ where $L'_1 = L_{\downarrow \varphi'}$ and $L'_2 = L_{\downarrow \varphi'}$.*

Definition 3 *Let φ be a strategy map over signature \mathcal{F} and φ' be a strategy map over signature $\mathcal{F}' = \mathcal{F} \cup \{f_{root} \mid f \in \mathcal{F}_{root}\}$ for a set $\mathcal{F}_{root} \subseteq \mathcal{F}$. Let $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ and $p \in \text{Pos}_A(t)$. We define the translation of t into terms $\mathcal{T}(\mathcal{F}'_{\varphi'}^{\sharp}, \mathcal{X}'_{\varphi'}^{\sharp})$ as $pos_{p, \mathcal{F}_{root}}(t) = s$ where $\text{Pos}(s) = \text{Pos}(t)$ and $\forall q \in \text{Pos}(t). \text{root}(t|_q) = f_{L_1|L_2}$, we have:*

$$\text{root}(s|_q) = \begin{cases} f_{root L'_1|L'_2 @ 0} & \text{if } p \in \text{Pos}_P(t), q \leq p, \text{ and } f \in \mathcal{F}_{root}; \\ & \text{where } L'_1 = L_{\downarrow \varphi'}(f_{root}) \text{ and } L'_2 = L_{\downarrow \varphi'}(f_{root}) \\ f_{L'_1|L'_2} & \text{otherwise;} \\ & \text{where } L'_1 = L_{\downarrow \varphi'}(f) \text{ and } L'_2 = L_{\downarrow \varphi'}(f) \end{cases}$$

Definition 4 *Let φ be a strategy map over signature \mathcal{F} and φ' be a strategy map over signature $\mathcal{F}' = \mathcal{F} \cup \{f_{+j_{\Lambda}}^{\Lambda}, \dots, f_{+j_p}^p\}$ such that $\text{root}(t|_q) = f^q \in \mathcal{F}$, $j_{\Lambda}, \dots, j_p \in \mathbb{N}$, and $q.j_q \leq p.i$ for $q \leq p$. Let $t \in \mathcal{T}(\mathcal{F}_{\varphi}^{\sharp}, \mathcal{X}_{\varphi}^{\sharp})$ and $p \in \text{Pos}_A(t)$. We define the translation of t into terms $\mathcal{T}(\mathcal{F}'_{\varphi'}^{\sharp}, \mathcal{X}'_{\varphi'}^{\sharp})$ as $neg_p(t) = s$ where*

$\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t). \text{root}(t|_q) = f_{L_1|L_2}$, we have:

$$\text{root}(s|_q) = \begin{cases} f_{+iL'_1|L'_2} & \text{if } p \in \mathcal{P}os_{A-P}(t) \text{ and } q.i \leq p; \\ & \text{where } L'_1 = (i) \text{ and } L'_2 = (0) \downarrow_{\varphi'(f_{+i})} \\ f_{L'_1|L'_2} & \text{otherwise;} \\ & \text{where } L'_1 = L_1 \downarrow_{\varphi'(f)}, \text{ and } L'_2 = L_2 \downarrow_{\varphi'(f)} \end{cases}$$

The following proposition shows that each evaluation step with negative annotations can be simulated by the transformed program.

Proposition 1 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that \mathcal{R} is φ -terminating. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t, s \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$, and $q \in \mathcal{P}os_A(s)$, if $\langle t, \Lambda \rangle \xrightarrow{\#}_{\varphi, \mathcal{R}} \langle s, q \rangle$, then $\langle \text{pos}_{\Lambda, \mathcal{F}_\mathcal{R}^\varphi}(t), \Lambda \rangle \xrightarrow{\#}_{\varphi^{\text{III}}, \mathcal{R}^{\text{III}}} \langle s', q \rangle$ where either (1) $s' = \text{quote}(s \downarrow_{\varphi^{\text{III}}})$, (2) $s' = \text{pos}_{\Lambda, \mathcal{F}_\mathcal{R}^\varphi}(s)$, (3) $s' = \text{neg}_q(s)$, or (4) s and s' are $\xrightarrow{\#}$ -normal forms with a defined symbol at root position.*

Proof We consider the different cases for $\xrightarrow{\#}_{\varphi, \mathcal{R}}$.

1. The cases $t = f_{L|\text{nil}}(t_1, \dots, t_k)$ or $t = \bar{f}_{L_1|L_2}(t_1, \dots, t_k)$ are impossible.
2. Let $t = f_{L_1|i:L_2}(t_1, \dots, t_k)$, $i > 0$, $q = i$, and $s = f_{L_1 \oplus i|L_2}(t_1, \dots, t_k)$. If $f \notin \mathcal{F}_\mathcal{R}^\varphi$, then $\text{root}(\text{pos}_\Lambda(s)) = f_{L_1 \oplus i|L_2}$ and the conclusion follows. If $f \in \mathcal{F}_\mathcal{R}^\varphi$, then $\text{root}(\text{pos}_\Lambda(s)) = f_{\text{root}L'_1 \oplus i|L'_2 \oplus 0}$ such that L'_1 and L'_2 do not contain negative annotations. Thus, the conclusion follows and condition (2) is fulfilled.
3. Let $t = f_{L_1|-i:L_2}(t_1, \dots, t_k)$, $i > 0$, $q = \Lambda$, and $s = f_{L_1 \oplus -i|L_2}(t_1, \dots, t_k)$. In this case, the index $-i$ would not appear in $\text{pos}_\Lambda(t)$. Thus, condition (2) is fulfilled and $s' = \text{pos}_\Lambda(s)$.
4. Let $t = f_{L_1|0:L_2}(t_1, \dots, t_k) = \sigma(l')$, $\text{erase}(l') = l$ for $l \rightarrow r \in \mathcal{R}$ and substitution σ , $s = \sigma(\varphi(r))$, and $q = \Lambda$. Note that since \mathcal{R} is a CS and $f \in \mathcal{D}$, $f \notin \mathcal{F}_\mathcal{R}^\varphi$ and $\text{pos}_\Lambda(t) = t \downarrow_{\varphi^{\text{III}}}$.

If $l \rightarrow r \in \mathcal{R}^{\text{III}}$, then the conclusion follows and condition (1) is fulfilled. Consider $l \rightarrow r \notin \mathcal{R}^{\text{III}}$. Then, there is a set of rules $l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n \in \mathcal{R}^{\text{III}}$ such that $l_1 = l[\bar{x}]_Q$ for a set of positions Q and $r_n = \text{quote}(r)$. Moreover, when these rules are applied sequentially to term $\text{pos}_\Lambda(t)$, they produce term $s' = \text{quote}(\sigma(\varphi^{\text{III}}(r)))$. Hence, the conclusion follows and condition (1) is fulfilled.

5. Let $t = f_{L_1|0:L_2}(t_1, \dots, t_k)$, $\text{erase}(t)$ is not a redex w.r.t. \mathcal{R} , $OD_\mathcal{R}(t) = \emptyset$, $s = f_{L_1|L_2}(t_1, \dots, t_k)$, and $q = \Lambda$. If $\text{erase}(t)$ is not a redex neither w.r.t. \mathcal{R}^{III} , then the conclusion follows and condition (2) is fulfilled.

Consider $\text{erase}(t)$ is a redex w.r.t. \mathcal{R}^{III} , i.e. $\text{erase}(t) = \sigma(l)$ for $l \rightarrow r \in \mathcal{R}^{\text{III}}$. First, note that since \mathcal{R} is a CS and $f \in \mathcal{D}$, $f \notin \mathcal{F}_\mathcal{R}^\varphi$ and $\text{pos}_\Lambda(t) = t \downarrow_{\varphi^{\text{III}}}$. By

definition of the transformation \mathcal{R}^{neg} (which is the only one that modifies lhs's), $\exists l' \rightarrow r' \in \mathcal{R}$ s.t. l differs from l' in a set of variables positions, $\exists Q \subseteq \mathcal{P}os(l')$ s.t. $l = l'[\bar{x}]_Q$ for a set \bar{x} of free variables. Then, there is a set of rules $l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n \in \mathcal{R}^{\text{III}}$ such that $l_1 = l$ and $r_n = \text{quote}$. Moreover, when these rules are applied sequentially to term $pos_\Lambda(t)$, they produce a term t' which differs from t in a set of symbols $\{f^\Lambda, \dots, f^{q'}\}$ for a position $q' \in \mathcal{P}os(t)$ such that $t' = t[f^\Lambda, \dots, f^{q'}]_{\{\Lambda, \dots, q'\}}$. Note that since $erase(t)$ is not a redex w.r.t. \mathcal{R} , the sequence of rules $l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n$ will not be able to evaluate t to r_n at it will stop at some intermediate step, which is just the expression t' with symbols f^{+i} .

Note that since $OD_{\mathcal{R}}(t) = \emptyset$, there must be a constructor clash between t and each lhs in \mathcal{R} or a positive or empty position is demanded. Then, since the pattern matching is sequentially performed by the rules $l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n \in \mathcal{R}^{\text{III}}$, some evaluations could be started w.r.t. \mathcal{R}^{III} which would not be performed in the original program \mathcal{R} . However, since \mathcal{R} is φ -terminating, by Theorem 1, the term t' (with symbols f_+) can be safely produced. Hence, condition (4) is fulfilled.

6. Let $t = f_{L_1|0:L_2}(t_1, \dots, t_k)$, $erase(t)$ is not a redex w.r.t. \mathcal{R} , $OD_{\mathcal{R}}(t) = \{p'\}$, $s = \text{mark}(t, p')$ and $q = p'$.

First, since \mathcal{R} is a CS and $f \in \mathcal{D}$, $f \notin \mathcal{F}_{\mathcal{R}}^\varphi$ and $pos_\Lambda(t) = t_{\downarrow\varphi^{\text{III}}}$. Again, there is a set of rules $l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n \in \mathcal{R}^{\text{III}}$ such that $l_1 = l[\bar{x}]_Q$ for a set of positions Q and l_n differs from l in a set of symbols $\{f^\Lambda, \dots, f^{p''}\}$ for a position $p'' \in \mathcal{P}os(l)$ s.t. $p''.i = p'$ for $i \in \mathbb{N}$ and $l_n = l[\bar{y}]_{Q'}[f^\Lambda, \dots, f^{p''}]_{\{\Lambda, \dots, p''\}}$ for another set of positions Q' . That is, these rules are auxiliary rules introduced to stepwise the pattern matching process until position p' (which is the demanded position) is reached and its evaluation started. Hence, condition (3) is fulfilled. □

We denote by $pos(t)$ the extension of $pos_p(t)$ to include symbols f_{root} w.r.t. all positive positions in t , i.e. $root(pos(t)|_p) = f_{root L'_1|L'_2}$ if $p \in \mathcal{P}os_P(t) \cap \mathcal{P}os_{\mathcal{F}_{\mathcal{R}}^\varphi}(t)$, where $t|_p = f_{L_1|L_2}$, $L'_1 = L_{1\downarrow\varphi^{\text{III}}(f_{root})}$, and $L'_2 = L_{2\downarrow\varphi^{\text{III}}(f_{root})}$; and $root(pos(t)|_p) = f_{L'_1|L'_2}$ otherwise, where $t|_p = f_{L_1|L_2}$, $L'_1 = L_{1\downarrow\varphi^{\text{III}}(f)}$, and $L'_2 = L_{2\downarrow\varphi^{\text{III}}(f)}$. The following lemma ensures that normalization from a term $\text{quote}(t)$ is equivalent to from a term $pos(t)$.

Lemma 1 *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS and φ be a standard strategy map. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . Let $t, s \in \mathcal{T}(\mathcal{F}^{\text{III}\sharp}_{\varphi^{\text{III}}}, \mathcal{X}^{\text{III}\sharp}_{\varphi^{\text{III}}})$. Then, $\langle \text{quote}(t), \Lambda \rangle \xrightarrow{\sharp}_{\varphi^{\text{III}}, \mathcal{R}^{\text{III}}} \langle pos(t), \Lambda \rangle$.*

Proof Straightforward because pos and quote_τ modify the same symbols in $\mathcal{P}os_P(t) \cap \mathcal{P}os_{\mathcal{F}_{\mathcal{R}}^\varphi}(t)$ and, since φ is a standard strategy map, positive indices are always reduced before the root symbol. □

In the following theorem we prove completeness of the transformation, i.e. that the transformation preserves normal forms (while they can include extra symbols at some inner positions).

Definition 5 (Maximal constructor context) Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and φ be an E-strategy map. The maximal constructor context $C_t[\]$ of a term $t \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$ is defined as: $C_t[\] = \square$ if $\text{root}(\text{erase}(t)) \notin \mathcal{C}$; $C_t[\] = c(C_{t_1}[\], \dots, C_{t_k}[\])$ if $\text{root}(\text{erase}(t)) = c \in \mathcal{C}$.

Theorem 4 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that \mathcal{R} is φ -terminating. Let $\mathcal{R}^\mathbb{I} = (\mathcal{F}^\mathbb{I}, R^\mathbb{I})$ and $\varphi^\mathbb{I}$. For all $t, s \in \mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$, if $\langle t, \Lambda \rangle \xrightarrow{\varphi, \mathcal{R}}^\# \langle s, \Lambda \rangle$, then $\langle \text{pos}(t), \Lambda \rangle \xrightarrow{\varphi^\mathbb{I}, \mathcal{R}^\mathbb{I}}^\# \langle s', \Lambda \rangle$ where $C_s = C_{s'}$ and $\forall p \in \text{minimal}_{\leq}(\text{Pos}_{\mathcal{F}^\mathbb{I} - \mathcal{F}}(s'))$, $s'|_p$ is a $\xrightarrow{\varphi^\mathbb{I}}$ -normal form.

Proof (Sketch) By induction on the length of the sequence $\langle t, \Lambda \rangle \xrightarrow{\varphi, \mathcal{R}}^\# \langle s, \Lambda \rangle$ and considering Proposition 1 and Lemma 1. \square

Theorem 2 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that \mathcal{R} is φ -terminating. Let $\mathcal{R}^\mathbb{I} = (\mathcal{F}^\mathbb{I}, R^\mathbb{I})$ and $\varphi^\mathbb{I}$. For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in \text{eval}_{\mathcal{R}}^\varphi(t)$, then $s \in \text{eval}_{\mathcal{R}^\mathbb{I}}^{\varphi^\mathbb{I}}(\text{quote}(t))$.

Proof By Theorem 4 and Lemma 1. \square

On the other hand, to prove correctness, we provide a translation of the labeling of terms in $\mathcal{T}(\mathcal{F}_{\varphi^\mathbb{I}}^\#, \mathcal{X}_{\varphi^\mathbb{I}}^\#)$ back to the labeling of $\mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$. Given two strategy lists $L, L' \in \mathbb{L}$, we denote by $L_{\uparrow L'}$ the maximal sublist L'' such that $L \sqsubseteq L''$ and $L'' \sqsubseteq L'$.

Definition 6 Let φ be a strategy map over signature \mathcal{F} . Let $\varphi^\mathbb{I}$ be a strategy map over signature $\mathcal{F}^\mathbb{I}$. Let $t \in \mathcal{T}(\mathcal{F}_{\varphi^\mathbb{I}}^\#, \mathcal{X}_{\varphi^\mathbb{I}}^\#)$. We define the translation of t into terms $\mathcal{T}(\mathcal{F}_\varphi^\#, \mathcal{X}_\varphi^\#)$ as $\text{rem}(t) = s$ where $\text{Pos}(s) = \text{Pos}(t)$ and $\forall q \in \text{Pos}(t)$,

$$\text{root}(s|_q) = \begin{cases} f_{L'_1|L'_2} & \text{if } t|_q = f_{L_1|L_2}; \\ & \text{where } L'_1 = L_{1\uparrow\varphi(f)}, L'_2 = L_{2\uparrow\varphi(f)}, \\ & \text{and } L_1 + L_2 \sqsubseteq \varphi(f) \\ f_{L'_1|L'_2} & \text{if } t|_q = f_{\text{root}L_1|L_2}; \\ & \text{where } L'_1 = L_{1\uparrow\varphi(f)}, L'_2 = L_{2\uparrow\varphi(f)}, \\ & \text{and } L_1 + L_2 \sqsubseteq \varphi(f) \\ f_{L'_1|L'_2} & \text{if } t|_q = f_{+iL_1|L_2}; \\ & \text{where } L'_1 = \begin{cases} (-i)_{\uparrow\varphi(f)} & \text{if } (-i) \sqsubseteq \varphi(f) \\ (i)_{\uparrow\varphi(f)} & \text{if } (i) \sqsubseteq \varphi(f) \end{cases} \\ & L'_2 = L_{2\uparrow\varphi(f)}, \text{ and } L_1 + L_2 \sqsubseteq \varphi(f) \end{cases}$$

Proposition 2 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t, s \in \mathcal{T}(\mathcal{F}^{\text{III}}_{\varphi^{\text{III}}}, \mathcal{X}_{\varphi^{\text{III}}})$ without symbols `quote`, and $q \in \text{Pos}_A(s)$, if $\langle t, \Lambda \rangle \xrightarrow{\#}_{\varphi^{\text{III}}, \mathcal{R}^{\text{III}}} \langle s, q \rangle$, then $\langle \text{rem}(t), \Lambda \rangle \xrightarrow{\#}_{\varphi, \mathcal{R}} \langle \text{rem}(s), q \rangle$.

Proof Straightforward since cases of $\xrightarrow{\#}$ for positive strategy annotations are only considered. \square

Theorem 5 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t, s \in \mathcal{T}(\mathcal{F}^{\text{III}}_{\varphi^{\text{III}}}, \mathcal{X}_{\varphi^{\text{III}}})$ without symbols `quote`, and $q \in \text{Pos}_A(s)$, if $\langle \text{quote}(t), \Lambda \rangle \xrightarrow{\#!}_{\varphi^{\text{III}}, \mathcal{R}^{\text{III}}} \langle s, \Lambda \rangle$, then $\langle \text{rem}(t), \Lambda \rangle \xrightarrow{\#!}_{\varphi, \mathcal{R}} \langle s', q \rangle$ where $C_s = C_{s'}$ and $\forall p \in \text{minimal}_{\leq}(\text{Pos}_{\mathcal{F}^{\text{III}} - \mathcal{F}}(s))$, $s'|_p$ is a $\xrightarrow{\#}$ -normal form.

Proof Similar to Theorem 4 but using Proposition 2 and without any condition on termination. \square

Theorem 3 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\text{III}} = (\mathcal{F}^{\text{III}}, R^{\text{III}})$ and φ^{III} . For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in \text{eval}_{\mathcal{R}^{\text{III}}}^{\varphi^{\text{III}}}(\text{quote}(t))$, then $s \in \text{eval}_{\mathcal{R}}^{\varphi}(t)$.

Proof By Theorem 5 and Lemma 1. \square

B Benchmarks code

B.1 Program pi

This program codifies the well-known infinite series expansion to approximate the π number: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$. 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 rnil : -> LRecip .
op rcons : Recip LRecip -> LRecip [strat (1 2)] .
op from : Nat -> LNat [strat (1 0)] .
op seriespos : Nat LNat -> LRecip [strat (1 2 0)] .
op seriesneg : 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 seriespos(0,Z) = rnil .
eq seriespos(s(N),cons(X,cons(Y,Z)))
  = rcons(posrecip(Y),seriesneg(N,Z)) .
eq seriesneg(0,Z) = rnil .
eq seriesneg(s(N),cons(X,cons(Y,Z)))
  = rcons(negrecip(Y),seriespos(N,Z)) .
eq pi(X) = seriespos(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

```

B.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 PINoNeg 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 cons+2 : Nat LNat -> LNat [strat (2)] .
  op cons-root : Nat LNat -> LNat [strat (1 0)] .
  op rnil : -> LRecip .
  op rcons : Recip LRecip -> LRecip [strat (1 2)] .
  op from : Nat -> LNat [strat (1 0)] .
  op seriespos : Nat LNat -> LRecip [strat (1 2 0)] .
  op seriespos+2 : Nat LNat -> LRecip [strat (2 0)] .
  op seriesneg : Nat LNat -> LRecip [strat (1 2 0)] .
  op seriesneg+2 : Nat LNat -> LRecip [strat (2 0)] .
  op pi : Nat -> LRecip [strat (1 0)] .
  ops quote ondemand : Nat -> Nat [strat (0)] .
  ops quote ondemand : LNat -> LNat [strat (0)] .

```

```

ops quote ondemand : Recip -> Recip [strat (0)] .
ops quote ondemand : LRecip -> LRecip [strat (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 .
var R W : Recip . var L V : LRecip .
eq from(X) = quote(cons(X,from(s(X)))) .
eq seriespos(0,Z) = quote(rnil) .
eq seriespos(s(N),cons(X,Z))
  = seriespos--+2(s(N),cons--+2(X,ondemand(Z))) .
eq seriespos--+2(s(N),cons--+2(X,cons(Y,Z)))
  = quote(rcons(posrecip(Y),seriesneg(N,Z))) .
eq seriesneg(0,Z) = quote(rnil) .
eq seriesneg(s(N),cons(X,Z))
  = seriesneg--+2(s(N),cons--+2(X,ondemand(Z))) .
eq seriesneg--+2(s(N),cons--+2(X,cons(Y,Z)))
  = quote(rcons(negrecip(Y),seriespos(N,Z))) .
eq pi(X) = quote(seriespos(X,from(0))) .
eq plus(0,Y) = quote(Y) .
eq plus(s(X),Y) = quote(s(plus(X,Y))) .
eq times(0,Y) = quote(0) .
eq times(s(X),Y) = quote(plus(Y,times(X,Y))) .
eq square(X) = quote(times(X,X)) .
eq quote(0) = 0 .
eq quote(s(N)) = s(quote(N)) .
eq quote(nil) = nil .
eq quote(cons(X,Z)) = cons-root(quote(X),Z) .
eq quote(from(X)) = from(quote(X)) .
eq quote(posrecip(X)) = posrecip(quote(X)) .
eq quote(negrecip(X)) = negrecip(quote(X)) .
eq quote(rnil) = rnil .
eq quote(rcons(W,V)) = rcons(quote(W),quote(V)) .
eq quote(seriespos(X,Z)) = seriespos(quote(X),quote(Z)) .
eq quote(seriesneg(X,Z)) = seriesneg(quote(X),quote(Z)) .
eq quote(pi(X)) = pi(quote(X)) .
eq quote(plus(X,Y)) = plus(quote(X),quote(Y)) .
eq quote(times(X,Y)) = times(quote(X),quote(Y)) .
eq quote(square(X)) = square(quote(X)) .
eq cons-root(X,Z) = cons(X,Z) .
eq quote(seriespos--+2(X,Z)) = seriespos--+2(X,Z) .
eq quote(cons--+2(X,Z)) = cons--+2(X,Z) .
eq quote(seriesneg--+2(X,Z)) = seriesneg--+2(X,Z) .
eq ondemand(0) = 0 .
eq ondemand(s(N)) = s(N) .
eq ondemand(nil) = nil .
eq ondemand(cons(X,Z)) = cons(X,Z) .
eq ondemand(from(X)) = from(quote(X)) .
eq ondemand(posrecip(X)) = posrecip(X) .
eq ondemand(negrecip(X)) = negrecip(X) .
eq ondemand(rnil) = rnil .
eq ondemand(rcons(W,V)) = rcons(W,V) .
eq ondemand(seriespos(X,Z)) = seriespos(quote(X),quote(Z)) .

```

```

eq ondemand(seriesneg(X,Z)) = seriesneg(quote(X),quote(Z)) .
eq ondemand(pi(X)) = pi(quote(X)) .
eq ondemand(plus(X,Y)) = plus(quote(X),quote(Y)) .
eq ondemand(times(X,Y)) = times(quote(X),quote(Y)) .
eq ondemand(square(X)) = square(quote(X)) .
eq ondemand(seriespos--+2(X,Z)) = seriespos--+2(X,Z) .
eq ondemand(cons--+2(X,Z)) = cons--+2(X,Z) .
eq ondemand(seriesneg--+2(X,Z)) = seriesneg--+2(X,Z) .
endo

```

B.3 Program msquare_eager

This program uses functions `minus`, `square`, `times`, and `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.

```

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 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

```

B.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.

```

obj MINUS-SQUARE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op plus : Nat Nat -> Nat [strat (1 0)] .
  op times : Nat Nat -> Nat [strat (1 0)] .
  op square : Nat -> Nat [strat (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

```

B.5 Program msquare_neg

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

```

obj MINUS-SQUARE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1)] .
  op plus : Nat Nat -> Nat [strat (1 0)] .
  op times : Nat Nat -> Nat [strat (1 0)] .
  op square : Nat -> Nat [strat (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

```

B.6 Transformed program msquare_neg_noneg

The application of the program transformation for removing negative annotations presented in this paper to the program `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.

```

obj MINUS-SQUARE is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (0)] .
  op s-root : Nat -> Nat [strat (1 0)] .
  op plus : Nat Nat -> Nat [strat (1 0)] .
  op times : Nat Nat -> Nat [strat (1 0)] .
  op square : Nat -> Nat [strat (0)] .
  op minus : Nat Nat -> Nat [strat (1 0)] .
  op minus+2 : Nat Nat -> Nat [strat (2 0)] .
  ops quote ondemand : Nat -> Nat [strat (0)] .
  vars X Y : Nat .
  eq plus(0,Y) = quote(Y) .

```

```

eq plus(s(X),Y) = quote(s(plus(X,Y))) .
eq times(0,Y) = quote(0) .
eq times(s(X),Y) = quote(plus(Y,times(X,Y))) .
eq square(Y) = quote(times(Y,Y)) .
eq minus(0,Y) = quote(0) .
eq minus(s(X),Y) = minus-+2(s(X),ondemand(Y)) .
eq minus-+2(s(X),0) = quote(s(X)) .
eq minus(s(X),Y) = minus-+2(s(X),ondemand(Y)) .
eq minus-+2(s(X),s(Y)) = quote(minus(X,Y)) .
eq quote(0) = 0 .
eq quote(s(X)) = s-root(quote(X)) .
eq quote(plus(X,Y)) = plus(quote(X),Y) .
eq quote(times(X,Y)) = times(quote(X),Y) .
eq quote(square(X)) = square(X) .
eq quote(minus(X,Y)) = minus(quote(X),Y) .
eq s-root(X) = s(X) .
eq quote(minus-+2(X,Y)) = minus-+2(X,Y) .
eq ondemand(0) = 0 .
eq ondemand(s(X)) = s(X) .
eq ondemand(plus(X,Y)) = plus(quote(X),Y) .
eq ondemand(times(X,Y)) = times(quote(X),Y) .
eq ondemand(square(X)) = square(X) .
eq ondemand(minus(X,Y)) = minus(quote(X),Y) .
eq ondemand(minus-+2(X,Y)) = minus-+2(X,Y) .

```

endo

C Termination Proof of program pi

Consider the program of Section B.1. After applying the transformation included in [2] for proving termination, we obtain the following program:

```

obj Pitr 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 seriespos : Nat LNat -> LRecip [strat (1 2 0)] .
  op seriesneg : 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 seriespos(0,Z) = rnil .

```

```

eq seriespos(s(N),cons(X,Z)) = seriespos(s(N),cons2(X,Z)) .
eq seriespos(s(N),cons2(X,cons(Y,Z)))
  = rcons(posrecip(Y),seriesneg(N,Z)) .
eq seriesneg(0,Z) = rnil .
eq seriesneg(s(N),cons(X,Z)) = seriesneg(s(N),cons2(X,Z)) .
eq seriesneg(s(N),cons2(X,cons(Y,Z)))
  = rcons(negrecip(Y),seriespos(N,Z)) .
eq pi(X) = seriespos(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 [17], in order to prove termination of PItr (which only contains positive annotations), we can use the techniques for proving termination of context-sensitive rewriting (see [19] for a survey of these techniques). The application of Zantema's transformation ([25]) to remove positive annotations, yields the following TRS (in a generic syntax not bound to OBJ programs):

```

from(X) → cons(X,n_from(s(X)))
seriespos(0,Z) → rnil
seriespos(s(N),cons(X,Z))
  → seriespos(s(N),cons2(X,activate(Z)))
seriespos(s(N),cons2(X,cons(Y,Z)))
  → rcons(posrecip(Y),seriesneg(N,activate(Z)))
seriesneg(0,Z) → rnil
seriesneg(s(N),cons(X,Z))
  → seriesneg(s(N),cons2(X,activate(Z)))
seriesneg(s(N),cons2(X,cons(Y,Z)))
  → rcons(negrecip(Y),seriespos(N,activate(Z)))
pi(X) → seriespos(X,from(0))
plus(0,Y) → Y
plus(s(X),Y) → s(plus(X,Y))
times(0,Y) → 0
times(s(X),Y) → plus(Y,times(X,Y))
square(X) → times(X,X)
from(X) → n_from(X)
activate(n_from(X)) → from(X)
activate(X) → X

```

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) = X02;
[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;
[seriesneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[seriespos](X0,X1) = 0;
[cons](X0,X1) = 0;
['plus'](X0,X1) = X0;

```

checking component 2 (disjunction of 1 constraints)

```

[rnil] = 0;
[0] = 0;
[activate](X0) = X0;
[n__from](X0) = 0;
[square](X0) = X02;
[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;
[seriesneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[seriespos](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) = X02;
[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;
[seriesneg](X0,X1) = 0;
[rcons](X0,X1) = 0;

```

```
[cons2](X0,X1) = 0;  
[seriespos](X0,X1) = 0;  
[cons](X0,X1) = 0;  
['seriesneg'](X0,X1) = X0;  
['seriespos'](X0,X1) = X0;
```

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