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/18/03 **Pages: 51**
Title: On-Demand Strategy Annotations Revisited
Author(s): M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas
Date: 24/07/03
Keywords: Declarative programming, demandedness, lazy evaluation, OBJ, on-demand strategy annotations

Vº Bº
Leader of research Group Author(s)

On-Demand Strategy Annotations Revisited*

M. Alpuente† S. Escobar† B. Gramlich‡ S. Lucas†

Abstract

In functional languages such as OBJ*, CafeOBJ, and Maude, symbols are given strategy annotations that specify (the order in) which subterms are evaluated. Syntactically, strategy annotations 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 prescribes the evaluation of an argument whereas a negative index means "evaluate on-demand". While strategy annotations containing only natural numbers have been implemented and investigated to some extent (regarding, e.g., termination and completeness), fully general annotations (also called *on-demand* strategy annotations), which have been proposed to support laziness in OBJ-like languages, are disappointingly under-explored to date. In this paper, we first point out a number of problems of current proposals for handling on-demand strategy annotations, such as undecidability of the reduction relation or inadequacy of the model. Then, we propose a solution to these problems by keeping an accurate track of annotations along evaluation sequences. This solution is defined as a suitable extension of the E-evaluation strategy of OBJ-like languages (which only considers annotations given as natural numbers) to on-demand strategy annotations. Our strategy overcomes the drawbacks of these previous proposals and also exhibits better computational properties. For instance, we show how to use it for computing (head-)normal forms. We also introduce a transformation for proving termination of the new evaluation strategy by using standard techniques. Finally, we present an interpreter of the new strategy together with some encouraging experiments.

Keywords: Declarative programming, demandedness, lazy evaluation, OBJ, on-demand strategy annotations

1 Introduction

Eager rewriting-based programming languages such as Lisp, OBJ*, CafeOBJ, ELAN, or Maude evaluate expressions by innermost rewriting. Since nonter-

*Work partially supported by CICYT TIC2001-2705-C03-01, Acciones Integradas HA2001-0059 and HU2001-0019.

†DSIC, UPV, Camino de Vera s/n, E-46022 Valencia, Spain. {alpuente,sescobar,slucas}@dsic.upv.es

‡AG Theoretische Informatik und Logik, Institut für Computersprachen, TU Wien, Favoritenstr. 9, E185/2 A-1040 Wien, Austria. gramlich@logic.at

mination is a frequent problem of innermost reduction, *syntactic annotations* (generally specified as sequences of integers associated to function arguments, called *local strategies*) have been used in OBJ2 [FGJM85], OBJ3 [GWM+00], CafeOBJ [FN97], and Maude [CELM96] to improve efficiency and (hopefully) avoid nontermination. Local strategies are used in OBJ programs[1] for guiding the *evaluation strategy* (abbr. *E*-strategy): when considering a function call $f(t_1, \ldots, t_k)$, only the arguments whose indices are present as *positive* integers in the local strategy for f are evaluated (following the specified ordering). If 0 is encountered then evaluation at the root position is attempted The limits of using only positive annotations regarding correctness and completeness of computations are discussed in [AEL02, Luc01a, Luc02b, NO01, OF00]: the obvious problem is that the absence of some indices in the local strategies can have a negative impact on the ability of such strategies to compute normal forms.

Example 1 *Consider the following* OBJ *program (borrowed from [NO01]):*

```
obj Ex1 is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat        [strat (1)] .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat  [strat (1)] .
  op 2nd  : LNat -> Nat       [strat (1 0)] .
  op from : Nat -> LNat       [strat (1 0)] .
  vars X Y : Nat . var Z : LNat .
  eq 2nd(cons(X,cons(Y,Z))) = Y .
  eq from(X) = cons(X,from(s(X))) .
endo
```

The OBJ *evaluation of* `2nd(from(0))` *is given by the following sequence (where we underline the redex reduced in each step):*

$$\texttt{2nd(}\underline{\texttt{from(0)}}\texttt{)} \rightarrow \texttt{2nd(cons(0,from(s(0))))}$$

The evaluation stops here since reductions on the second argument of `cons` *are disallowed (index* `2` *is not included in the strategy for* `cons`*). Note that we cannot apply the rule defining* `2nd` *because the subterm* `from(s(0))` *should be further reduced. Thus, a further step is* demanded *(by the rule of* `2nd`*) in order to obtain the desired outcome:*

$$\texttt{2nd(cons(0,}\underline{\texttt{from(s(0))}}\texttt{))} \rightarrow \texttt{2nd(cons(0,cons(s(0),from(s(s(0))))))}$$

Now, we do not *need to reduce the second argument of the inner occurrence of* `cons` *anymore, since reducing at the root position yields the final value:*

$$\underline{\texttt{2nd(cons(0,cons(s(0),from(s(s(0))))))}} \rightarrow \texttt{s(0)}$$

Therefore, the rather intuitive notion of demanded evaluation of an argument of a function call (see [AL02] for a survey discussing this topic) arises as a possible solution to this problem. In [NO01, OF00], *negative* indices are proposed to indicate those arguments that should be evaluated only 'on-demand',

[1] As in [GWM+00], by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

where the 'demand' is an attempt to match an argument term with the left-hand side of a rewrite rule [Eke00, GWM+00, OF00]. For instance, in [NO01] the authors suggest `(1 -2)` as the apt local strategy for `cons` in Example 1. The inspiration for the local strategies of OBJ comes from *lazy rewriting* (*LR*) [FKW00], a demand-driven technique where syntactic annotations allow the eager evaluation of function arguments, whereas the default strategy is lazy. However, the extended on-demand E-strategy of [NO01, OF00] presents a number of drawbacks, which we formally address in the paper. The following example illustrates that the notion of demandedness which is formalized in [NO01] needs to be refined to be entirely satisfactory in practice.

Example 2 *Consider the following* OBJ *program*[2] *encoding the length function for lists which does not allow any reduction on its argument and which uses an auxiliary symbol* `length'` *including an on-demand strategy annotation for its argument:*

```
obj Ex2 is
  protecting Ex1 .
  op length : LNat -> Nat     [strat (0)] .
  op length' : LNat -> Nat    [strat (-1 0)] .
  var X : Nat . var Z : LNat .
  eq length(nil) = 0 .
  eq length(cons(X,Z)) = s(length'(Z)) .
  eq length'(Z) = length(Z) .
endo
```

The expression `length'(from(0))` *is rewritten (in one step) to the expression* `length(from(0))`*. No evaluation is demanded on the argument of* `length'` *for enabling this step and no further evaluation on* `length(from(0))` *should be performed due to the local strategy* `(0)` *of* `length`*. However, the annotation* `(-1 0)` *of function* `length'` *is treated in such a way that the on-demand evaluation of the expression* `length'(from(0))` *yields an infinite sequence (whether*[3] *we use the operational model in [OF00] or whether we use [NO01]). For instance,* CafeOBJ *ends with a stack overflow*[4]*:*

```
Ex2> red length'(from(0)) .
-- reduce in Ex2 : length'(from(0))
Error: Stack overflow (signal 1000)
```

This is because the negative annotations are implemented as marks on terms which can (inappropriately) initiate reductions later on; see Example 6 below.

Besides the interest of on-demand strategy annotations for avoiding non-termination in OBJ programs while ensuring correctness and completeness of

[2]The reserved words `protecting` and `extending` can be understood as *module reuse* in the OBJ syntax.

[3]Actually, the operational models in [OF00] and [NO01] differ and deliver different computations, see Example 7 below.

[4]Negative annotations are (syntactically) accepted in current OBJ implementations, namely OBJ3, Maude, and CafeOBJ, but they have no effect on the computations of OBJ3 and Maude whereas CafeOBJ manages negative annotations using the model of [OF00, NO01].

the computations, the research on on-demand strategy annotations is quite interesting from the programming point of view.

Example 3 *Consider the following* Maude *program* `pi` *which codifies the well-known infinite series expansion to approximate number* π*:*

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

```
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 seriepos : Nat LNat -> LRecip     [strat (1 2 0)] .
  op serieneg : Nat LNat -> LRecip     [strat (1 2 0)] .
  op pi : Nat -> LRecip                [strat (1 0)] .
  vars N X Y : Nat . var Z : LNat .
  eq from(X) = cons(X,from(s(X))) .
  eq seriepos(0,Z) = rnil .
  eq seriepos(s(N),cons(X,cons(Y,Z))) =
                                 rcons(posrecip(Y),serieneg(N,Z)) .
  eq serieneg(0,Z) = rnil .
  eq serieneg(s(N),cons(X,cons(Y,Z))) =
                                 rcons(negrecip(Y),seriepos(N,Z)) .
  eq pi(X) = seriepos(X,from(0)) .
endo
```

A term[5] `pi(2)` *approximates the number* $\pi/4$ *using* 2 *elements of the series expansion, i.e. the intended behavior is*

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

where `posrecip(`n`)` *denotes the positive reciprocal* $1/n$ *and* `negrecip(`n`)` *denotes* $-1/n$*. The specification uses negative annotations to obtain a terminating and complete program, which can not be obtained using only positive annotations (if index* `-2` *is removed from the local strategy for* `cons`*, then the program becomes incomplete; and if index* `-2` *is replaced by index* `2`*, then the program becomes non-terminating). Indeed, termination of the program with negative annotations (as above) can be formally proved using the technique of Section 6 (see the formal proof in Appendix C).*

[5] Naturals `1`, `2`, ... are used as shorthand to numbers $\mathtt{s}^n(0)$ where $n = 1, 2, \ldots$.

In this paper, after some preliminaries in Section 2, in Section 3 we recall the current proposals for dealing with on-demand E-strategy annotations in OBJ languages and discuss some drawbacks regarding the treatment of demandedness. In Section 4 we (re-)formulate the computational model of *on-demand strategy annotations* by handling demandedness in a different way. We demonstrate that the new on-demand strategy outperforms the original one. In Section 5, we show that our definition behaves better than lazy rewriting (LR) and on-demand rewriting (ODR), the natural extension of context-sensitive rewriting [Luc98] to deal with on-demand strategy annotations, regarding the ability to compute (head-)normal forms. Section 6 introduces a transformation which can be used to formally prove termination of programs that use our computational model for implementing arbitrary local strategy annotations of this type. In order to prove the practicality of our ideas, we present in Section 7 an interpreter of the on-demand evaluation strategy introduced in the paper together with some encouraging experiments. Section 8 concludes and summarizes our contributions.

This paper is a substantially extended and improved version of [AEGL02]. In particular, the definition of the on-demand evaluation strategy has been improved, and Section 5.2, which compares the refined on-demand evaluation strategy of the paper with on-demand rewriting, has been added. The proofs of all technical results are included in Appendix A. Appendix B provides the source code of the benchmarks presented in Section 7, and Appendix C includes the termination proof for the program `pi` used in Example 3 and in Section 7.

2 Preliminaries

In this paper, we follow the standard framework of term rewriting (see [BN98, TeR03]). 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 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 $a\ R^* b$. Throughout the paper, $\mathcal{X}$ denotes a countable set of variables and $\mathcal{F}$ denotes a signature, i.e. a set of function symbols $\{\mathtt{f}, \mathtt{g}, \ldots\}$, each having a fixed arity given by a function $ar : \mathcal{F} \to \mathbb{N}$. 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 labeled trees in the usual way. Positions $p, q, \ldots$ are represented by sequences of positive natural numbers used to address subterms of t. We denote the empty sequence by Λ. By $\mathcal{P}os(t)$ we denote the set of positions of a term t. The set of positions of non-variable symbols in t is denoted by $\mathcal{P}os_{\mathcal{F}}(t)$, and $\mathcal{P}os_{\mathcal{X}}(t)$ are the positions of variables in t. Given positions p, q, we denote their concatenation by $p.q$. Positions are ordered by the standard prefix ordering $\leq$. given a set of positions P, $minimal_{\leq}(P)$ is the set of minimal positions of P w.r.t. $\leq$. For $p \in \mathcal{P}os(t)$, 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 labeling the root of t is denoted by $root(t)$.

A rewrite rule is an ordered pair (l, r), written $l \to r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. 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 $\mathcal{D} = \{root(l) \mid l \to r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. An instance $\sigma(l)$ of a *lhs* $l \in L(\mathcal{R})$ by any substitution σ is called a redex. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to s (at position p), written $t \xrightarrow{p}_{\mathcal{R}} s$ (or just $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for $l \to r \in R$, $p \in \mathcal{P}os(t)$, and substitution σ. A term is a *head-normal form* if it doesn't reduce (in finitely many steps) to a redex.

Given a signature $\mathcal{F}$, a mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a *replacement map* (or $\mathcal{F}$-map) if for all $f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ [Luc98]. The set of *μ-replacing* (or simply *replacing*) positions $\mathcal{P}os^\mu(t)$ of a term t is: $\mathcal{P}os^\mu(t) = \{\Lambda\}$, if $t \in \mathcal{X}$ and $\mathcal{P}os^\mu(t) = \{\Lambda\} \cup \bigcup_{i \in \mu(f)} i.\mathcal{P}os^\mu(t|_i)$, if $root(t) = f$. 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 $f \in \mathcal{F}$, $\mu(f) \subseteq \mu'(f)$. The lattice $(\mathcal{P}(\mathbb{N}), \subseteq, \varnothing, \mathbb{N}, \cup)$ induces a lattice $(M_\mathcal{F}, \sqsubseteq, \mu_\bot, \mu_\top, \sqcup)$: The minimum (maximum) element is $\mu_\bot$ ($\mu_\top$), given by $\mu_\bot(f) = \varnothing$ ($\mu_\top(f) = \{1, \ldots, ar(f)\}$) for all $f \in \mathcal{F}$. The *lub* $\sqcup$ is given by $(\mu \sqcup \mu')(f) = \mu(f) \cup \mu'(f)$ for all $f \in \mathcal{F}$.

3 Rewriting with strategy annotations

A local strategy for a k-ary symbol $f \in \mathcal{F}$ is a sequence $\varphi(f)$ of integers taken from $\{-k, \ldots, -1, 0, 1, \ldots, k\}$ which are given in parentheses. We define an ordering $\sqsubseteq$ between sequences of integers as follows: $nil \sqsubseteq L$, for all sequence L, $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_1\ j_2\ \cdots\ j_n)$ if $i_1 = j_1$ and $(i_2\ \cdots\ i_m) \sqsubseteq (j_2\ \cdots\ j_n)$, or $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_1\ j_2\ \cdots\ j_n)$ if $i_1 \neq j_1$ and $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_2\ \cdots\ j_n)$.

A mapping φ that associates a local strategy $\varphi(f)$ to every $f \in \mathcal{F}$ is called an E-strategy map [Nag99, NO01, OF00]. The extension of ordering $\sqsubseteq$ to strategy maps is defined as follows: $\varphi \sqsubseteq \varphi'$ if for all $f \in \mathcal{F}$, $\varphi(f) \sqsubseteq \varphi'(f)$. In other words, $\varphi \sqsubseteq \varphi'$ means that $\varphi(f)$ is a subsequence of $\varphi'(f)$, that is, $\varphi'(f)$ may contain some additional indices (as compared to $\varphi(f)$).

The semantics of rewriting under a given E-evaluation map φ is usually given by means of a mapping $eval_\varphi : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ from terms to the set of its computed values (technically E-normal forms).

3.1 Rewriting with positive E-strategy maps

Nagaya describes the mapping $eval_\varphi$ for *positive* E-strategy maps φ (i.e., E-strategy maps where negative indices are *not* allowed) by using a reduction relation on pairs $\langle t, p\rangle$ of labeled terms t and positions p [Nag99]. Let $\mathbb{L}$ be the set of all lists consisting of integers and $\mathbb{L}_n$ be the set of all lists of integers

whose absolute value does not exceed $n \in \mathbb{N}$. Given an E-strategy map φ for $\mathcal{F}$, we use the signature $\mathcal{F}^{\mathbb{N}}_{\varphi} = \{f_L \mid f \in \mathcal{F}, L \in \mathbb{L}_{ar(f)}, \text{ and } L \sqsubseteq \varphi(f)\}$ and labeled variables $\mathcal{X}^{\mathbb{N}}_{\varphi} = \{x_{nil} \mid x \in \mathcal{X}\}$. An E-strategy map φ for $\mathcal{F}$ is extended to a mapping from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi},\mathcal{X}^{\mathbb{N}}_{\varphi})$ by introducing the local strategy associated to each symbol as a subscript of the symbol. The mapping $erase : \mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi},\mathcal{X}^{\mathbb{N}}_{\varphi}) \to \mathcal{T}(\mathcal{F},\mathcal{X})$ removes labelings from symbols in the obvious way. Then, given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and a positive E-strategy map φ for $\mathcal{F}$, $eval_{\varphi}$ is defined as $eval_{\varphi}(t) = \{erase(s) \in \mathcal{T}(\mathcal{F},\mathcal{X}) \mid \langle\varphi(t),\Lambda\rangle \xrightarrow{\mathbb{N}}{}^{!}_{\varphi} \langle s,\Lambda\rangle\}$, where the binary relation $\xrightarrow{\mathbb{N}}_{\varphi}$ on $\mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi},\mathcal{X}^{\mathbb{N}}_{\varphi}) \times \mathbb{N}^*_+$ behaves as follows.

Definition 1 [Nag99, Definition 6.1.3] *Given $\langle t,p\rangle$, where $t \in \mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi},\mathcal{X}^{\mathbb{N}}_{\varphi})$ and $p \in \mathcal{P}os(t)$, $\langle t,p\rangle \xrightarrow{\mathbb{N}}_{\varphi} \langle s,q\rangle$ if and only if $p \in \mathcal{P}os(t)$ and either*

1. *$root(t|_p) = f_{nil}$, $s = t$ and $p = q.i$ for some i; or*
2. *$t|_p = f_{i:L}(t_1,\ldots,t_k)$, with $i > 0$, $s = t[f_L(t_1,\ldots,t_k)]_p$ and $q = p.i$; or*
3. *$t|_p = f_{0:L}(t_1,\ldots,t_k)$, $erase(t|_p)$ is not a redex, $s = t[f_L(t_1,\ldots,t_k)]_p$, $q = p$; or*
4. *$t|_p = f_{0:L}(t_1,\ldots,t_k) = \sigma(l')$, $erase(l') = l$, $s = t[\sigma(\varphi(r))]_p$ for some $l \to r \in R$ and substitution σ, $q = p$.* □

Intuitively, an innermost evaluation is performed, which is restricted to (and follows the order of) those indices included in the E-strategy map. This means that if a positive index $i > 0$ is found in the list labeling the symbol at $t|_p$, then the index is removed from the list, the "target position" is moved from p to $p.i$, and the subterm $t|_{p.i}$ is considered next. If 0 is found, then the evaluation of $t|_p$ is attempted: if possible, a rewriting step is performed; otherwise, the 0 is removed from the list. In both cases, the evaluation continues at the same position p.

Example 4 *Consider the* OBJ *program and the expression* `2nd(from(0))` *of Example 1. Note that this program has only positive annotations. The evaluation of expression* `2nd(from(0))` *produces the following sequence according to Definition 1 (we surround the index involved in the evaluation step by a box):*

$$
\begin{aligned}
&\langle \mathtt{2nd}_{(\boxed{1}\ 0)}(\mathtt{from}_{(1\ 0)}(\mathtt{0}_{nil})),\Lambda\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\mathtt{from}_{(\boxed{1}\ 0)}(\mathtt{0}_{nil})),1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\mathtt{from}_{(0)}(\mathtt{0}_{\boxed{nil}})),1.1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\overline{\mathtt{from}_{(\boxed{0})}(\mathtt{0}_{nil})}),1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\mathtt{cons}_{(\boxed{1})}(\mathtt{0}_{nil},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))),1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\mathtt{cons}_{nil}(\mathtt{0}_{\boxed{nil}},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))),1.1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(0)}(\mathtt{cons}_{\boxed{nil}}(\mathtt{0}_{nil},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))),1\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{(\boxed{0})}(\mathtt{cons}_{nil}(\mathtt{0}_{nil},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))),\Lambda\rangle\\
&\quad\to_{\varphi}\ \langle \mathtt{2nd}_{nil}(\mathtt{cons}_{nil}(\mathtt{0}_{nil},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))),\Lambda\rangle
\end{aligned}
$$

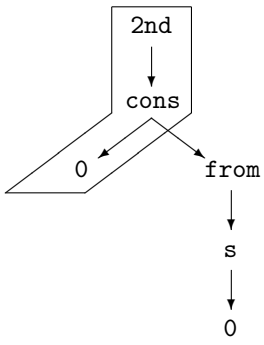

Figure 1: The positive part of the expression `2nd(cons(0,from(0)))`.

The evaluation stops at this point since no further evaluation step can be performed, see Figure 1 where the positive (or reducible) part of the expression `2nd(cons(0,from(0)))` *is marked.*

3.2 The on-demand evaluation strategy

On the one hand, Ogata and Futatsugi [OF00] have provided an operational description of the *on-demand evaluation strategy* $eval_\varphi$ where negative integers are also allowed in local strategies. On the other hand, Nakamura and Ogata [NO01] have described the corresponding evaluation mapping $eval_\varphi$ by using a reduction relation (similarly to [Nag99]). We consider here the latter one since it is more abstract and independent of the CafeOBJ programming language.

Given an E-strategy map φ, we use the signature[6] $\mathcal{F}_\varphi = \{f_L^b \mid f \in \mathcal{F}, L \in \mathbb{L}_{ar(f)}, L \sqsubseteq \varphi(f)$, and $b \in \{0,1\}\}$ and labeled variables $\mathcal{X}_\varphi = \{x_{nil}^0 \mid x \in \mathcal{X}\}$. An on-demand flag $b = 1$ indicates that the term may be reduced if demanded. An E-strategy map φ for $\mathcal{F}$ is extended to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi)$ as follows:

$$\varphi(t) = \begin{cases} x_{nil}^0 & \text{if } t = x \in \mathcal{X} \\ f_{\varphi(f)}^0(\varphi(t_1), \ldots, \varphi(t_k)) & \text{if } t = f(t_1, \ldots, t_k) \end{cases}$$

On the other hand, the mapping $erase : \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ removes labelings from symbols in the obvious way. The (partial) function $flag : \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi) \times \mathbb{N}_+^* \to \{0,1\}$ returns the flag of the function symbol at a position of the term: $flag(t,p) = b$ if $root(t|_p) = f_L^b$. The map $up : \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi) \to \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi)$ (resp. $dn : \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi) \to \mathcal{T}(\mathcal{F}_\varphi, \mathcal{X}_\varphi)$) switches on (resp. switches off) the on-demand flag of each function symbol in a term simply by applying $b = 1$ (resp. $b = 0$), i.e. $up(x_{nil}^0) = dn(x_{nil}^0) = x_{nil}^0$, $up(f_L^b(t_1, \ldots, t_k)) = f_L^1(up(t_1), \ldots, up(t_k))$, and $dn(f_L^b(t_1, \ldots, t_k)) = f_L^0(dn(t_1), \ldots, dn(t_k))$.

[6]Note that Nakamura and Ogata's definition (as well as Nagaya's Definition 1) uses $\mathcal{F}_\mathbb{L}$ and $\mathcal{X}_\mathbb{L}$ instead of $\mathcal{F}_\varphi$ and $\mathcal{X}_\varphi$, where the restriction to $L \sqsubseteq \varphi(f)$ is not considered. However, using terms over $\mathcal{F}_\varphi$ does not cause loss of generality; furthermore, it actually provides a more accurate framework for formalizing and studying the strategy, since these terms are the only class of terms involved in the computations.

When it is being examined whether a term t matches the left-hand side l of a rule, a top-to-bottom and left-to-right pattern matching is performed. Let $\mathcal{P}os_{\neq}(t,l) = \{p \in \mathcal{P}os_{\mathcal{F}}(t) \cap \mathcal{P}os_{\mathcal{F}}(l) \mid root(l|_p) \neq root(t|_p)\}$ be the set of (common) positions of non-variable disagreeing symbols of terms t and l. Then, the map $df_l : \mathcal{T}(\mathcal{F},\mathcal{X}) \to \mathbb{N}_+^* \cup \{\top\}$ returns the first position where the term and the lhs differ (on some non-variable positions of the lhs) or $\top$ if each function symbol of the term coincides with l:

$$df_l(t) = \begin{cases} min_{\leq_{lex}}(\mathcal{P}os_{\neq}(t,l)) & \text{if } \mathcal{P}os_{\neq}(t,l) \neq \varnothing \\ \top & \text{otherwise} \end{cases}$$

where $\leq_{lex}$ is the lexicographic ordering on positions: $p \leq_{lex} q$ iff $p \leq q$ or $p = w.i.p'$, $q = w.j.q'$, $i,j \in \mathbb{N}$, and $i < j$.

Similarly, given a TRS $\mathcal{R}$, the map $DF_{\mathcal{R}} : \mathcal{T}(\mathcal{F},\mathcal{X}) \to \mathbb{N}_+^* \cup \{\top\}$ returns the first position (w.r.t. the inverse of the lexicographic order, i.e. right-to-left and bottom-up) where the term differs w.r.t. all lhs's:

$$DF_{\mathcal{R}}(t) = \begin{cases} \top & \text{if } df_l(t) = \top \text{ for some } l \to r \in \mathcal{R} \\ max_{<_{lex}}\{df_l(t) \mid l \to r \in \mathcal{R}\} & \text{otherwise} \end{cases}$$

Definition 2 [NO01, Definition 4.4] *Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and an arbitrary E-strategy map φ for $\mathcal{F}$, $eval_\varphi : \mathcal{T}(\mathcal{F},\mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F},\mathcal{X}))$ is defined as $eval_\varphi(t) = \{erase(s) \in \mathcal{T}(\mathcal{F},\mathcal{X}) \mid \langle\varphi(t),\Lambda\rangle \to_\varphi^! \langle s,\Lambda\rangle\}$. The binary relation $\to_\varphi$ on $\mathcal{T}(\mathcal{F}_\varphi,\mathcal{X}_\varphi) \times \mathbb{N}_+^*$ is defined as follows: $\langle t,p\rangle \to_\varphi \langle s,q\rangle$ if and only if $p \in \mathcal{P}os(t)$ and either*

1. *$root(t|_p) = f^b_{nil}$, $s = t$ and $p = q.i$ for some i; or*
2. *$t|_p = f^b_{i:L}(t_1,\ldots,t_k)$, $i > 0$, $s = t[f^b_L(t_1,\ldots,t_k)]_p$ and $q = p.i$; or*
3. *$t|_p = f^b_{-i:L}(t_1,\ldots,t_k)$, $i > 0$, $s = t[f^b_L(t_1,\ldots,up(t_i),\ldots,t_k)]_p$ and $q = p$; or*
4. *$t|_p = f^b_{0:L}(t_1,\ldots,t_k)$, $s = t[t']_p$, $q = p$ where t' is a term such that*
   (a) *$t' = \theta(\varphi(r))$ if $DF_{\mathcal{R}}(erase(t|_p)) = \top$, $t|_p = \theta(l')$, $erase(l') = l$ and $l \to r \in \mathcal{R}$.*
   (b) *$t' = f^b_L(t_1,\ldots,t_k)$ if either $DF_{\mathcal{R}}(erase(t|_p)) = \top$ and $erase(t|_p)$ is not a redex, or $DF_{\mathcal{R}}(erase(t|_p)) = \Lambda$, or $DF_{\mathcal{R}}(erase(t|_p)) = p' \neq \Lambda$ and $flag(t,p.p') = 0$;*
   (c) *$t' = f^b_L(t_1,\ldots,t_i[up(s)]_{p''},\ldots,t_k)$ if $DF_{\mathcal{R}}(erase(t|_p)) = p' = i.p''$, $flag(t,p.p') = 1$, $\langle dn(t|_{p.p'}),\Lambda\rangle \to_\varphi^! \langle s,\Lambda\rangle$, and $DF_{\mathcal{R}}(erase(t|_p[s]_{p'})) = p'$;*
   (d) *$t' = t|_p[up(s)]_{p'}$ if $DF_{\mathcal{R}}(erase(t|_p)) = p' \neq \Lambda$, $flag(t,p.p') = 1$, $\langle dn(t|_{p.p'}),\Lambda\rangle \to_\varphi^! \langle s,\Lambda\rangle$, and either $p' <_{lex} DF_{\mathcal{R}}(erase(t|_p[s]_{p'}))$ or $DF_{\mathcal{R}}(erase(t|_p[s]_{p'})) = \top$.* □

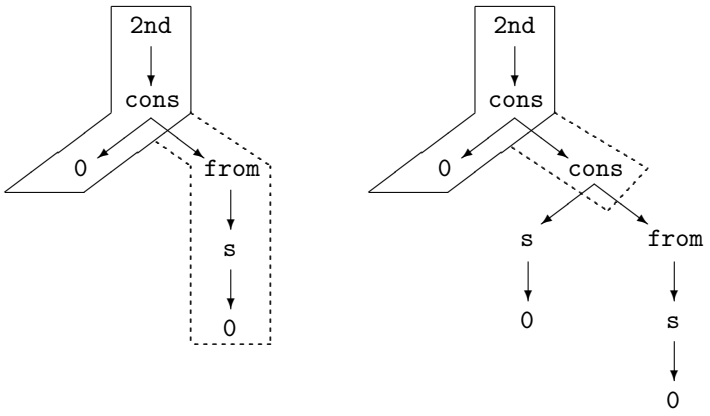

Figure 2: The positive and the on-demand parts of the expressions `2nd(cons(0,from(0)))` and `2nd(cons(0,cons(s(0),from(s(s(0))))))`.

Case 1 means that no more annotations are provided and the evaluation is completed. In case 2, a positive argument index is found and the evaluation proceeds by selecting the subterm at such argument. In case 3, the subterm at the argument indicated by the (absolute value of the) negative index is completely marked with on-demand flags. Case 4 considers the attempt to match the term against the left-hand sides of the program rules. Case 4.*a* applies if the considered (unlabeled) subterm is a redex (which is, then, contracted). If the subterm is not a redex, cases 4.*b*, 4.*c* and 4.*d* are considered, possibly involving some on-demand evaluation steps on some subterm. The selected demanded position for term t (w.r.t. program $\mathcal{R}$) is denoted as $DF_{\mathcal{R}}(t)$ (eventually, symbol $\top$ is returned if t matches the left-hand side of some rule of the TRS). According to $DF_{\mathcal{R}}(t)$, case 4.*b* applies if no demanded evaluation is allowed (or required). Cases 4.*c* and 4.*d* apply if the on-demand evaluation of the subterm $t|_{p.p'}$ is required, i.e. $DF_{\mathcal{R}}(t|_p) = p'$. In both cases, the evaluation is attempted; if it finishes, the evaluation of $t|_p$ continues according to the computed value.

Example 5 *Consider the* OBJ *program of Example 1 but including the local strategy* `(1 -2)` *for symbol* `cons`.

```
obj Ex1A 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 2nd  : LNat -> Nat       [strat (1 0)] .
  op from : Nat -> LNat       [strat (1 0)] .
  vars X Y : Nat . var Z : LNat .
  eq 2nd(cons(X,cons(Y,Z))) = Y .
  eq from(X) = cons(X,from(s(X))) .
endo
```

$$
\begin{array}{ll}
\langle \mathtt{2nd}^0_{(\boxed{1}\ 0)}(\mathtt{from}^0_{(1\ 0)}(\mathtt{0}^0_{nil})), \Lambda\rangle & \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{from}^0_{(\boxed{1}\ 0)}(\mathtt{0}^0_{nil})), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{from}^0_{(0)}(\mathtt{0}^0_{\boxed{nil}})), 1.1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\underline{\mathtt{from}^0_{(\boxed{0})}(\mathtt{0}^0_{nil})}), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{cons}^0_{(\boxed{1}\ -2)}(\mathtt{0}^0_{nil}, \mathtt{from}^0_{(1\ 0)}(\mathtt{s}^0_{(1)}(\mathtt{0}^0_{nil})))), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{cons}^0_{(-2)}(\mathtt{0}^0_{\boxed{nil}}, \mathtt{from}^0_{(1\ 0)}(\mathtt{s}^0_{(1)}(\mathtt{0}^0_{nil})))), 1.1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{cons}^0_{(\boxed{-2})}(\mathtt{0}^0_{nil}, \mathtt{from}^0_{(1\ 0)}(\mathtt{s}^0_{(1)}(\mathtt{0}^0_{nil})))), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(0)}(\mathtt{cons}^0_{\boxed{nil}}(\mathtt{0}^0_{nil}, \mathtt{from}^1_{(1\ 0)}(\mathtt{s}^1_{(1)}(\mathtt{0}^1_{nil})))), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{2nd}^0_{(\boxed{0})}(\mathtt{cons}^0_{nil}(\mathtt{0}^0_{nil}, \underline{\mathtt{from}^{\boxed{1}}_{(1\ 0)}(\mathtt{s}^1_{(1)}(\mathtt{0}^1_{nil}))})), \Lambda\rangle \\
& \quad \left|\begin{array}{ll}
\langle \mathtt{from}^0_{(\boxed{1}\ 0)}(\mathtt{s}^0_{(1)}(\mathtt{0}^0_{nil})), \Lambda\rangle & \\
\quad \to_\varphi & \langle \mathtt{from}^0_{(0)}(\mathtt{s}^0_{(\boxed{1})}(\mathtt{0}^0_{nil})), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{from}^0_{(0)}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{\boxed{nil}})), 1.1\rangle \\
\quad \to_\varphi & \langle \mathtt{from}^0_{(0)}(\mathtt{s}^0_{\boxed{nil}}(\mathtt{0}^0_{nil})), 1\rangle \\
\quad \to_\varphi & \langle \underline{\mathtt{from}^0_{(\boxed{0})}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil}))}, \Lambda\rangle \\
\quad \to_\varphi & \langle \overline{\mathtt{cons}^0_{(\boxed{1}-2)}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil}), \mathtt{from}^0_{(0)}(\mathtt{s}^0_{(1)}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil}))))}, \Lambda\rangle \\
\quad \to_\varphi & \langle \mathtt{cons}^0_{-2)}(\mathtt{s}^0_{\boxed{nil}}(\mathtt{0}^0_{nil}), \mathtt{from}^0_{(0)}(\mathtt{s}^0_{(1)}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil})))), 1\rangle \\
\quad \to_\varphi & \langle \mathtt{cons}^0_{(\boxed{-2})}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil}), \mathtt{from}^0_{(0)}(\mathtt{s}^0_{(1)}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil})))), \Lambda\rangle \\
\quad \to_\varphi & \langle \mathtt{cons}^0_{nil}(\mathtt{s}^0_{nil}(\mathtt{0}^0_{nil}), \mathtt{from}^1_{(0)}(\mathtt{s}^1_{(1)}(\mathtt{s}^1_{nil}(\mathtt{0}^1_{nil})))), \Lambda\rangle
\end{array}\right. \\
\quad \to_\varphi & \langle \underline{\mathtt{2nd}^0_{(\boxed{0})}(\mathtt{cons}^0_{nil}(\mathtt{0}^0_{nil}, \mathtt{cons}^1_{nil}(\mathtt{s}^1_{nil}(\mathtt{0}^1_{nil}), \mathtt{from}^1_{(0)}(\mathtt{s}^1_{(1)}(\mathtt{s}^1_{nil}(\mathtt{0}^1_{nil})))))), \Lambda\rangle} \\
\quad \to_\varphi & \langle \mathtt{s}^1_{nil}(\mathtt{0}^1_{nil}), \Lambda\rangle
\end{array}
$$

Figure 3: On-demand evaluation of term `2nd(from(0))` by Definition 2.

The evaluation of the expression `2nd(from(0))` *produces the evaluation sequence of Figure 3 according to Definition 2. Roughly speaking, the annotation* `-2` *of symbol* `cons` *switches on the on-demand flag of expression* `from(s(0))` *in the seventh evaluation step (see also Figure 2 where the positive and the on-demand parts of the expression* `2nd(from(0))` *are marked). Then, in the next step, this expression is evaluated to a head-normal form in a separate subsequence. Finally, after the corresponding head-normal form is obtained, the symbol* `2nd` *at the top position is evaluated and the final outcome* `s(0)` *is delivered (see expression* `2nd(cons(0,cons(s(0),from(s(s(s(0)))))))` *in Figure 2 where the subterm* `from(s(s(s(0))))` *is not under a demanded position).*

Note that the computational description of on-demand strategy annotations above involves recursive steps. A single reduction step on a (labeled) term t may involve the application of more than one reduction step on subterms of t (as shown by steps $4(c)$ and $4(d)$). In fact, the definition of a single rewriting

step may depend on the possibility of evaluating some arguments of the considered function call. This implies that the one-step reduction relation proposed by Nakamura and Ogata is in general undecidable. Therefore, associated notions such as the normal form property (w.r.t. their reduction relation) are also undecidable.

Furthermore, as remarked in our introduction, the notion of demandedness formalized in [NO01] needs to be refined in order to be entirely satisfactory in practice.

Example 6 *Following the Example 2. The on-demand evaluation of term* `length'(from(0))` *following Definition 2 yields the following infinite sequence:*

$$
\begin{array}{ll}
\langle \mathtt{length'}^{0}_{(\boxed{\text{-1}}\ 0)}(\mathtt{from}^{0}_{(1\ 0)}(\mathtt{0}^{0}_{nil})), \Lambda\rangle & \\
\quad \to_{\varphi} & \langle \overline{\mathtt{length'}^{0}_{(\boxed{0})}(\mathtt{from}^{1}_{(1\ 0)}(\mathtt{0}^{1}_{nil}))}, \Lambda\rangle \\
\quad \to_{\varphi} & \langle \mathtt{length}^{0}_{(\boxed{0})}(\overline{\mathtt{from}^{\boxed{1}}_{(1\ 0)}(\mathtt{0}^{1}_{nil})}), \Lambda\rangle \\
& \quad \left| \begin{array}{ll}
\langle \mathtt{from}^{0}_{(\boxed{1}\ 0)}(\mathtt{0}^{0}_{nil}), \Lambda\rangle & \\
\quad \to_{\varphi} & \langle \mathtt{from}^{0}_{(0)}(\mathtt{0}^{0}_{\boxed{nil}}), 1\rangle \\
\quad \to_{\varphi} & \langle \overline{\mathtt{from}^{0}_{(\boxed{0})}(\mathtt{0}^{0}_{nil})}, \Lambda\rangle \\
\quad \to_{\varphi} & \langle \overline{\mathtt{cons}^{0}_{(\boxed{1})}(\mathtt{0}^{0}_{nil}, \mathtt{from}^{0}_{(0)}(\mathtt{s}^{0}_{(1)}(\mathtt{0}^{0}_{nil}))}, \Lambda\rangle \\
\quad \to_{\varphi} & \langle \mathtt{cons}^{0}_{nil}(\mathtt{0}^{0}_{\boxed{nil}}, \mathtt{from}^{0}_{(0)}(\mathtt{s}^{0}_{(1)}(\mathtt{0}^{0}_{nil})), 1\rangle \\
\quad \to_{\varphi} & \langle \mathtt{cons}^{0}_{nil}(\mathtt{0}^{0}_{nil}, \mathtt{from}^{0}_{(0)}(\mathtt{s}^{0}_{(1)}(\mathtt{0}^{0}_{nil})), \Lambda\rangle
\end{array} \right. \\
\quad \to_{\varphi} & \langle \overline{\mathtt{length}^{0}_{(\boxed{0})}(\mathtt{cons}^{1}_{nil}(\mathtt{0}^{1}_{nil}, \mathtt{from}^{1}_{(0)}(\mathtt{s}^{1}_{(1)}(\mathtt{0}^{1}_{nil}))))}, \Lambda\rangle \\
\quad \to_{\varphi} & \langle \overline{\mathtt{s}^{0}_{(\boxed{1})}(\mathtt{length'}^{0}_{(-1\ 0)}(\mathtt{from}^{1}_{(1\ 0)}(\mathtt{s}^{1}_{(1)}(\mathtt{0}^{1}_{nil}))))}, \Lambda\rangle \\
\quad \to_{\varphi} & \langle \mathtt{s}^{0}_{nil}(\mathtt{length'}^{0}_{(\boxed{\text{-1}}\ 0)}(\mathtt{from}^{1}_{(1\ 0)}(\mathtt{s}^{1}_{(1)}(\mathtt{0}^{1}_{nil})))), 1\rangle \\
\quad \to_{\varphi} & \cdots
\end{array}
$$

In the first reduction step, annotation -1 *of symbol* `length'` *is consumed according to case* 3 *of Definition 2 and, thus, subterm* $\mathtt{from}^{0}_{(1\ 0)}(\mathtt{0}^{0}_{nil})$ *is marked with the on-demand (superscript) flag. Annotation* `0` *of* `length'` *is reached and the whole term is rewritten using rule* `length'(Z) = length(Z)`*, according to case* $4(a)$ *of Definition 2. Then, annotation* `0` *of* `length` *is reached but the whole term is not a redex, and an on-demand position has to be looked for. Here, the function* $DF_{\mathcal{R}}$ *for calculating demanded positions returns position* 1*, i.e.* $DF_{\mathcal{R}}(\mathtt{length}(\mathtt{from}(\mathtt{0}))) = 1$*, and because this position is marked with the on-demand flag, then, case* $4(c)$ *or* $4(d)$ *is applied and the evaluation of position* 1 *is initiated (using a different subsequence). Thus,* `length` *is rewritten to* `length'` *and the cycle is repeated.*

Note that, within the labeled term $\mathtt{length}^{0}_{(0)}(\mathtt{from}^{1}_{(1\ 0)}(\mathtt{0}^{1}_{nil}))$*, the strategy does not recognize that the (activated) on-demand flags on symbols* `from` *and* `0` *do* not *come from the local annotation for* `length`*. That is, the strategy does not record the origin of on-demand flags. Hence, it (unnecessarily) evaluates the argument of* `length`*. Moreover, at this point, this evaluation does* not *correspond*

to the 'intended' meaning of the strategy annotations that the programmer may have in mind (since the specific annotation `(0)` *for* `length` *forbids reductions on its argument).*

On the other hand, the two existing definitions for the on-demand E-strategy (namely Nakamura and Ogata's [NO01] and Ogata and Futatsugi's [OF00]) sensibly differ. For instance, Nakamura and Ogata select a demanded position for evaluating a given term t by taking the maximum of all positions demanded on t by each rule of the TRS (according to the lexicographic ordering on positions). Yet, in Ogata and Futatsugi's selection of demanded positions, the ordering of the rules in the program is extremely important both in the definition and at the implementation level.

Example 7 *Consider the* OBJ *program of Example 2 with the strategy* `(1 0)` *for* `length` *which makes the evaluation of the call* `length(from(0))` *non-terminating, together with the function* `geq` *defined by the following module:*

```
obj Ex3 is
  extending Ex2 .
  op length : LNat -> Nat     [strat (1 0)] .
  sorts Bool .
  op true  : -> Bool .
  op false : -> Bool .
  op geq   : Nat Nat -> Bool    [strat (-1 -2 0)] .
  vars X Y : Nat .
  eq geq(s(X),s(Y)) = geq(X,Y) .
  eq geq(X,0) = true .
endo
```

Consider the expression `geq(length(from(0)),length(nil))`*. According to Ogata and Futatsugi's definition of on-demand E-strategy, an infinite reduction sequence is started since position* 1 *is the left-most demandable position in the first rule of the program and, thus, it is selected as demanded and its (non-terminating) evaluation attempted. For instance,* CafeOBJ *ends with a stack overflow:*

```
Ex3> red geq(length(from(0)),length(nil)) .
-- reduce in Ex3 : geq(length(from(0)),length(nil))
Error: Stack overflow (signal 1000)
```

However, Nakamura and Ogata's definition of on-demand E-strategy selects position 2 *as demanded (according to the inverse of the lexicographic ordering) and, after the evaluation, the second rule is applied, thus obtaining* `true` *(see case* $4(d)$ *in Definition 2).*

We claim that it is possible to provide a more practical framework for implementing and studying OBJ computations, which may integrate the most interesting features of modern evaluation strategies with on-demand syntactic annotations. This framework is developed in the following.

4 Improving rewriting under on-demand strategy annotations

As discussed at the end of the previous section, the existing operational models for arbitrary strategy annotations have a number of drawbacks: (1) the one-step reduction relation is, in general, undecidable; (2) the mechanization of demandedness by using negative annotations (via the marking of terms with flag 0 or flag 1) allows evaluation steps that shouldn't be possible, since (3) it does not properly keep track of the origin of the marks (lack of memory, see Example 6 above). Here, we want to introduce a further consideration which can be used for improving the previous definitions. Let us illustrate it by means of an example.

Example 8 *Consider the* OBJ *program of Example 7 together with the following function* `lt`*:*

```
obj Ex4 is
  protecting Ex3 .
  op lt : Nat Nat -> Bool    [strat (-1 -2 0)] .
  vars X Y : Nat .
  eq lt(0,s(Y)) = true .
  eq lt(s(X),s(Y)) = lt(X,Y) .
endo
```

Consider the expression $t =$ `lt(length(from(0)),0)`*, which is a* head-normal form *since no possible evaluation could enable the expression to match the left-hand side of a rule due to subterm* `0` *at position* 2*. Neither Nakamura and Ogata's nor Ogata and Futatsugi's formulations are able to avoid evaluations on* t*. For instance,* CafeOBJ *ends with a stack overflow:*

```
Ex4> red lt(length(from(0)),0).
-- reduce in Ex4 : lt(length(from(0)),0)
Error: Stack overflow (signal 1000)
```

Nevertheless, by exploiting the standard distinction between constructor and defined symbols of a signature in the presence of a TRS, it is easy to detect that no rule for `lt` *could ever be applied. Indeed,* `0` *is a constructor symbol in the input term* t *and, hence, it cannot be reduced for improving the matching of* t *against the left-hand side of the rule for* `lt`*. See [AFJV97, AL02, MR92] for a more detailed motivation and formal discussion of the use of these ideas for defining and using demand-driven strategies.*

In the following, we propose a refined (and fixed) definition of the on-demand E-strategy which takes into account all previous considerations. The two important points are the use of two lists of annotations for each symbol (instead of only one for the on-demand evaluation of Definition 2) and a special flag for avoiding recursive definitions of the strategy.

Given a E-strategy map φ, we use the signature[7] $\mathcal{F}^\sharp_\varphi = \cup\{f_{L_1|L_2}, \overline{f}_{L_1|L_2} \mid f \in \mathcal{F} \wedge L_1, L_2 \in \mathbb{L}_{ar(f)}.(L_1{+}{+}L_2 \sqsubseteq \varphi(f))\}$ and labeled variables $\mathcal{X}^\sharp_\varphi = \{x_{nil|nil} \mid$

[7]The function $++$ defines the concatenation of two sequences of integers.

$x \in \mathcal{X}\}$ for marking ordinary terms $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ as terms $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi,\mathcal{X}^\sharp_\varphi)$. 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. We use $f^\sharp$ to denote f or $\overline{f}$ for a symbol $f \in \mathcal{F}$. We define the list of active indices of a labeled symbol $f^\sharp_{L_1|L_2}$ as

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

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

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

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

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:

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

Note that the problem described in Example 8 is solved (along the lines of [MR92]) by restricting the attention to disagreeing positions that correspond to defined symbols (by using $\mathcal{P}os_{\mathcal{D}}(t)$).

Example 9 *Continuing Example 8 where* l_1 = `lt(0,s(Y))` *and* l_2 = `lt(s(X),s(Y))` *are the lhs's of the rules. Let* t_1 = `lt(length(from(0)),0)`*, we have* $DP_{l_1}(t_1) = \varnothing$ *and* $DP_{l_2}(t_1) = \varnothing$*, i.e. no position is demanded by* l_1 *or* l_2 *because of a constructor conflict with subterm* `0` *at position* 2*. Let* t_2 = `lt(length(from(0)),length(nil))`*, we have* $DP_{l_1}(t_2) = \{1,2\}$ *and* $DP_{l_2}(t_2) = \{1,2\}$*, i.e. positions* 1 *and* 2 *are demanded by* l_1 *and* l_2 *because both positions are rooted by defined symbols. Finally, given* t_3 = `lt(0,length(nil))`*, we have* $DP_{l_1}(t_3) = \{2\}$ *but* $DP_{l_2}(t_3) = \varnothing$*, i.e. position* 2 *is demanded by* l_1 *but not by* l_2 *because of a constructor conflict with* l_2*.*

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

$$ADP_l(t) = \begin{cases} DP_l(erase(t)) \cap \mathcal{P}os_A(t) & \text{if } DP_l(erase(t)) \not\subseteq \mathcal{P}os_P(t) \cup \mathcal{P}os_{nil}(t) \\ \varnothing & \text{otherwise} \end{cases}$$

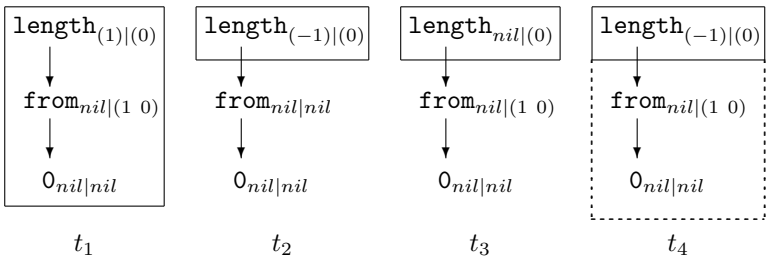

Figure 4: The positive and the on-demand parts of the terms of Example 10.

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

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.

Example 10 *Continuing Example 2 consider* $l = \mathtt{length(nil)}$ *(the lhs of the first rule for* `length`*). For* $t_1 = \mathtt{length}_{(1)|(0)}(\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))$*, we have* $DP_l(erase(t_1)) = \{1\}$ *but* $ADP_l(t_1) = \varnothing$*, i.e. position* 1 *is demanded by* l *but it is a positive position. For* $t_2 = \mathtt{length}_{(-1)|(0)}(\mathtt{from}_{nil|nil}(\mathtt{0}_{nil|nil}))$*, we have* $ADP_l(t_2) = \varnothing$*, i.e. position* 1 *is also demanded by* l *but it is rooted by a symbol with an empty annotation list. For* $t_3 = \mathtt{length}_{nil|(0)}(\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))$*, we have* $ADP_l(t_3) = \varnothing$*, i.e. position* 1 *is demanded by* l *too but it is not an active position. Finally, for* $t_4 = \mathtt{length}_{(-1)|(0)}(\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))$*, we have* $ADP_l(t_4) = \{1\}$*. See Figure 4 for the positive and the on-demand parts of each of these terms.*

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 $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, the total ordering $\leq_s$ between active positions of s is defined as (1) $\Lambda \leq_s p$ for all $p \in \mathcal{P}os_A(s)$; (2) if $i.p, i.q \in \mathcal{P}os_A(s)$ and $p \leq_{s|_i} q$, then $i.p \leq_s i.q$; and (3) if $i.p, j.q \in \mathcal{P}os_A(s)$, $i \neq j$, and i (or $-i$) appears before j (or $-j$) in $active(root(s))$, 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 $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ w.r.t. TRS $\mathcal{R}$ as follows:

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

Example 11 *Continuing Example 9. We have* $OD_\mathcal{R}(t_1) = \varnothing$ *where* $ADP_\mathcal{R}(t_1) = \varnothing$*,* $OD_\mathcal{R}(t_2) = \{1\}$ *where* $ADP_\mathcal{R}(t_2) = \{1, 2\}$*, and* $OD_\mathcal{R}(t_3) = \{2\}$ *where* $ADP_\mathcal{R}(t_3) = \{2\}$*.*

In order to overcome the undecidability of the evaluation strategy of Definition 2, we use symbols $\overline{f}$ 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 $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and a position $p \in \mathcal{P}os(t)$, $mark(t, p)$ is the

term s with all symbols above p (except the root) marked as non-evaluable, in symbols $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t)$, if $\Lambda < q < p$ and $root(t|_q) = f_{L_1|L_2}$, then $root(s|_q) = \overline{f}_{L_1|L_2}$, otherwise $root(s|_q) = root(t|_q)$.

Example 12 *Consider the program of Example 5 and the term* $t = \mathtt{2nd}_{(1)|(0)}(\mathtt{cons}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil}))))$. *We have that*

$$mark(t, 1.2) = \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil}))))$$

Finally, we define a binary relation $\xrightarrow{\sharp}_\varphi$ on the set $\mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \times \mathbb{N}^*_+$, such that a single reduction step on a (labeled) term t does not involve the application of recursive reduction steps on t. In the following definition, the symbol @ denotes appending an element at the end of a list.

Definition 3 *Given a TRS* $\mathcal{R} = (\mathcal{F}, R)$ *and an arbitrary E-strategy map* φ *for* $\mathcal{F}$, $eval_\varphi : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ *is defined as* $eval_\varphi(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle \varphi(t), \Lambda\rangle \xrightarrow{\sharp}{}^!_\varphi \langle s, \Lambda\rangle\}$. *The binary relation* $\xrightarrow{\sharp}_\varphi$ *on* $\mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \times \mathbb{N}^*_+$ *is defined as follows:* $\langle t, p\rangle \xrightarrow{\sharp}_\varphi \langle s, q\rangle$ *if and only if* $p \in \mathcal{P}os(t)$ *and either*

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

Cases 1 and 2 of Definition 3 essentially correspond to cases 1 and 2 of Definitions 1 and 2; that is, (1) no more annotations are provided and the evaluation is completed, or (2) 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

$$
\begin{aligned}
&\langle \mathtt{2nd}_{nil|(\boxed{1}\ 0)}(\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{from}_{nil|(\boxed{1}\ 0)}(\mathtt{0}_{nil|nil})), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{from}_{(1)|(0)}(\mathtt{0}_{\boxed{nil}|nil})), 1.1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\underline{\mathtt{from}_{(1)|(\boxed{0})}(\mathtt{0}_{nil|nil})}), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{cons}_{nil|(\boxed{1}\ -2)}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil})))), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{cons}_{(1)|(\boxed{-2})}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil})))), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{cons}_{(1\ -2)|\boxed{nil}}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil})))), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\mathtt{cons}_{(1\ \boxed{-2})|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil})))), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(\boxed{1}\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{0}_{nil|nil})))), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(0)}(\mathtt{s}_{nil|(\boxed{1})}(\mathtt{0}_{nil|nil})))), 1.2.1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(0)}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|\boxed{nil}})))), 1.2.1.1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{from}_{nil|(0)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{0}_{nil|nil})))), 1.2.1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \underline{\mathtt{from}_{(1)|(\boxed{0})}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}))})), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{nil|(\boxed{1}\ -2)}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \\
&\qquad\qquad \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1)}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})))))), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{(1)|(-2)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{0}_{nil|nil}), \cdots))), 1.2.1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{(1)|(\boxed{-2})}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \cdots))), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{(1\ -2)|\boxed{nil}}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \cdots))), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{2nd}_{(1)|(0)}(\overline{\mathtt{cons}}_{(1\ -2)|\boxed{nil}}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{(1\ -2)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \cdots))), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \underline{\mathtt{2nd}_{(1)|(\boxed{0})}(\mathtt{cons}_{(1\ -2)|nil}(\mathtt{0}_{nil|nil}, \mathtt{cons}_{(1\ -2)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \cdots)))}, \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}), \Lambda\rangle
\end{aligned}
$$

Figure 5: On-demand evaluation of term `2nd(from(0))` by Definition 3.

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 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 $\overline{f}$.

Example 13 *Following the Example 5, the appropriate evaluation sequence for the term* `2nd(from(0))` *via* $eval^{\varphi}_{\mathcal{R}}$ *is depicted in Figure 5, which is similar to that shown in Example 5. The main difference w.r.t. the evaluation sequence of Example 5 is the overline of symbol* `cons` *in order to avoid recursive definitions*

of the evaluation strategy and the use of two annotation list to keep an accurate track of annotations.

Example 14 *Following Examples 2 and 6. The on-demand evaluation of* `length'(from(0))` *under the refined on-demand strategy is the following:*

$$\begin{aligned}
&\langle \texttt{length'}_{nil|(\boxed{\texttt{-1}}\ 0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \texttt{length'}_{(-1)|(\boxed{0})}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \texttt{length}_{nil|(\boxed{0})}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \texttt{length}_{nil|nil}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})), \Lambda\rangle
\end{aligned}$$

In the first step, negative annotation -1 *of* `length'` *is recorded for further use according to case* 3 *of Definition 3. Annotation* `0` *of* `length'` *is processed and the whole term is rewritten using rule* `length'(Z) = length(Z)`*, according to case* 4 *of Definition 3. Then, annotation* `0` *of* `length` *is reached but the whole term cannot be rewritten since it is not a redex and demanded positions have to be looked for. However, no demanded position arises since the memoizing list of strategy annotations for* `length` *is empty (see* $ADP_{\mathcal{R}}(t_3)$ *in Example 10 above). Therefore, we obtain* `length(from(0))` *as the computed value of the evaluation, according to case* 5 *of Definition 3.*

In the following, we study different properties of our on-demand evaluation strategy.

4.1 Properties of the refined on-demand strategy

The following theorem shows that, for positive strategy annotations, each reduction step with $\xrightarrow{\sharp}_{\varphi}$ exactly corresponds to Nagaya's original relation $\xrightarrow{\mathbb{N}}_{\varphi}$ of Section 3.1. For an E-strategy map φ and a term $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, we define $\mathit{positive} : \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi}) \to \mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi}, \mathcal{X}^{\mathbb{N}}_{\varphi})$ as $\mathit{positive}(x_{nil|nil}) = x_{nil}$ for $x \in \mathcal{X}$ and $\mathit{positive}(f^{\sharp}_{L_1|L_2}(t_1, \ldots, t_n)) = f_{L_2'}(\mathit{positive}(t_1), \ldots, \mathit{positive}(t_n))$ where L_2' is L_2 without negative indices.

Theorem 1 *Let* $\mathcal{R}$ *be a TRS and* φ *be a positive* E*-strategy map. Let* $t, s \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ *and* $p \in \mathcal{P}os(t)$*. Then,* $\langle t, p\rangle \xrightarrow{\sharp}_{\varphi} \langle s, q\rangle$ *if and only if* $\langle \mathit{positive}(t), p\rangle \xrightarrow{\mathbb{N}}_{\varphi} \langle \mathit{positive}(s), q\rangle$*.*

Sometimes, it is interesting to get rid of the ordering among indices in local strategies. Then, we use *replacement maps* ($\mu \in M_{\mathcal{F}}$) [Luc98]. Let $\mu^{can}_{\mathcal{R}}$ 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 $\mathcal{R}$: for all $f \in \mathcal{F}$, $i \in \{1, \ldots, ar(f)\}$, $i \in \mu^{can}_{\mathcal{R}}(f)$ iff $\exists l \in L(\mathcal{R}), p \in \mathcal{P}os_{\mathcal{F}}(l), (root(l|_p) = f \wedge p.i \in \mathcal{P}os_{\mathcal{F}}(l))$ [Luc98]. Let $CM_{\mathcal{R}} = \{\mu \in M_{\mathcal{F}} \mid \mu^{can}_{\mathcal{R}} \sqsubseteq \mu\}$ be the set of replacement maps which are less than or equally restrictive as $\mu^{can}_{\mathcal{R}}$ [Luc98]. Given an E-strategy map φ,

let μ^φ be the following replacement map given by $\mu^\varphi(f) = \{|i| \mid i \in \varphi(f) \wedge i \neq 0\}$. We say that φ is a *canonical* E-strategy map (and, slightly abusing notation, we write $\varphi \in CM_\mathcal{R}$) if $\mu^\varphi \in CM_\mathcal{R}$. Given an E-strategy map, let $\varphi^\mathbb{N}$ be the E-strategy map obtained from φ by removing all negative indices for each symbol $f \in \mathcal{F}$. Note that $\varphi^\mathbb{N} \sqsubseteq \varphi$, for all E-strategy map φ.

In the following, we show that for E-strategy maps φ whose positive part (the sublists of positive indices) $\varphi^\mathbb{N}$ is canonical, extra negative annotations can be completely disregarded. This means that negative annotations are only meaningful if the positive indices do not include all indices in the canonical replacement map of the TRS.

Theorem 2 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map such that $\varphi^\mathbb{N} \in CM_\mathcal{R}$. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os(t)$. Then, $\langle t, p\rangle \xrightarrow{\sharp}_\varphi \langle s, q\rangle$ if and only if $\langle positive(t), p\rangle \xrightarrow{\mathbb{N}}_{\varphi^\mathbb{N}} \langle positive(s), q\rangle$.*

Example 14 above shows that restricting the evaluation by using on-demand strategy annotations can deliver terms which are not even head-normal forms w.r.t. $\to_\mathcal{R}$. The following result establishes conditions ensuring that the normal forms computed by our refined on-demand strategy are ordinary head-normal forms (w.r.t. the TRS). A TRS $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ is a constructor system (CS) if for all $f(l_1, \ldots, l_k) \to r \in R$, $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, for $1 \leq i \leq k$.

Theorem 3 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear CS, $\varphi \in CM_\mathcal{R}$ and $\varphi(f)$ end with 0 for all $f \in \mathcal{D}$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $s \in eval_\varphi(t)$, then s is a head-normal form of t.*

Left-linearity and CS conditions cannot be dropped, as [Luc01a] has shown for on-demand rewriting. The following two counterexamples are an adaptation of the ones in [Luc01a].

Example 15 *Consider the following TRS $\mathcal{R}$ from [Luc01a] which is not a CS:*

`f(g(x,a))` $\to$ `a` `g(a,b)` $\to$ `g(b,a)`

Let φ(`f`) = `(-1 0)`, φ(`g`) = `(1 2 0)`, *and* φ(`a`) = φ(`b`) = *nil*. *The term* t = `f(g(a,b))` *is not a head-normal form since* `f(g(a,b))` $\to$ `f(g(b,a))` $\to$ `a`. *However, the head-normal form of t is not computed by* $\xrightarrow{\sharp}_\varphi$ *:*

$$\begin{aligned}
&\langle \mathtt{f}_{nil|(\boxed{\mathtt{-1}}\ 0)}(\mathtt{g}_{nil|(1\ 2\ 0)}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_\varphi \ \langle \mathtt{f}_{(-1)|(\boxed{0})}(\mathtt{g}_{nil|(1\ 2\ 0)}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_\varphi \ \langle \mathtt{f}_{(-1)|nil}(\mathtt{g}_{nil|(1\ 2\ 0)}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil})), \Lambda\rangle
\end{aligned}$$

Note that $1 \notin \mathcal{P}os_{\neq}$(`f(g(a,b))`, `f(g(x,a))`), *i.e. position* 1 *of t is not demanded by lhs* `f(g(x,a))`.

Example 16 *Consider the following TRS $\mathcal{R}$ from [Luc01a] which is not left-linear:*

`f(x,x)` $\to$ `x` `a` $\to$ `b`

Let $\varphi(\mathtt{f}) = \mathtt{(-1\ -2\ 0)}$, $\varphi(\mathtt{a}) = \mathtt{(0)}$, *and* $\varphi(\mathtt{b}) = nil$. *Term* $t = \mathtt{f(a,b)}$ *is not a head-normal form since* $\mathtt{f(\underline{a},b)} \to \underline{\mathtt{f(b,b)}} \to \mathtt{b}$. *However, the head-normal form of* t *is not computed by* $\xrightarrow{\sharp}_\varphi$*:*

$$\begin{aligned}
&\langle \mathtt{f}_{nil|(\boxed{\mathtt{-1}}\ -2\ 0)}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil}), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_\varphi \ \langle \mathtt{f}_{(-1)|(\boxed{\mathtt{-2}}\ 0)}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil}), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_\varphi \ \langle \mathtt{f}_{(-1\ -2)|(\boxed{\mathtt{0}})}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil}), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_\varphi \ \langle \mathtt{f}_{(-1\ -2)|nil}(\mathtt{a}_{nil|nil}, \mathtt{b}_{nil|nil}), \Lambda\rangle
\end{aligned}$$

Note that $1.1 \notin \mathcal{P}os_{\neq}(\mathtt{f(a,b)}, \mathtt{f(x,x)})$*, i.e. position* 1.1 *of* t *is not demanded by lhs* $\mathtt{f(x,x)}$.

Theorem 3 suggests the following *normalization via φ-normalization procedure* to obtain normal forms of a term t: given an E-strategy map φ and $s = f(s_1, \ldots, s_k) \in eval_\varphi(t)$, the evaluation of t proceeds by (recursively) normalizing $s_1, \ldots, s_k$ using $eval_\varphi$. It is not difficult to see that confluence and φ-termination of the TRS guarantee that this procedure actually describes a normalizing strategy (see [Luc01a, Luc02a]).

In the next section, we show that our on-demand strategy improves *lazy rewriting*, a popular demand-driven technique to perform lazy functional computations which inspired the development of local strategies in OBJ, and *on-demand rewriting*, the natural extension of context-sensitive rewriting to deal with on-demand strategy annotations.

5 Comparison with other techniques dealing with on-demand annotations

5.1 Lazy rewriting

In lazy rewriting [FKW00, Luc02b], reductions are performed on a particular kind of *labeled terms*. Nodes (or positions) of a term t are labeled with e for the so-called *eager* positions or with ℓ for the so-called *lazy* ones: let $\mathcal{F}$ be a signature and $\mathcal{L} = \{e, \ell\}$. Then, $\mathcal{F} \times \mathcal{L}$ (or $\mathcal{F}_\mathcal{L}$) is a new signature of labeled symbols. The labeling of a symbol $f \in \mathcal{F}$ is denoted by f^e or f^ℓ rather than $\langle f, e\rangle$ or $\langle f, \ell\rangle$. Labeled terms are terms in $\mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$. Given $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$ and $p \in \mathcal{P}os(t)$, if $root(t|_p) = x^e$ $(= x^\ell)$ or $root(t|_p) = f^e$ $(= f^\ell)$, then we say that p is an *eager* (resp. *lazy*) position of t.

Given a replacement map $\mu \in M_\mathcal{F}$ and $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $label_\mu(s)$ denotes the following *intended* labeling of s: (1) the topmost position Λ of $label_\mu(s)$ is eager; (2) given a position $p \in \mathcal{P}os(label_\mu(s))$ and $i \in \{1, \ldots, ar(root(s|_p))\}$, the position $p.i$ of $label_\mu(s)$ is lazy if $i \notin \mu(root(s|_p))$, or is eager, otherwise.

Example 17 *Consider the program of Example 1 (as a TRS) and the replacement map* μ *given by* $\mu(\mathtt{2nd}) = \mu(\mathtt{from}) = \mu(\mathtt{cons}) = \mu(\mathtt{s}) = \{1\}$. *Then, the labeling of* $s = \mathtt{2nd(cons(0,from(s(0))))}$ *is* $t = label_\mu(s) =$

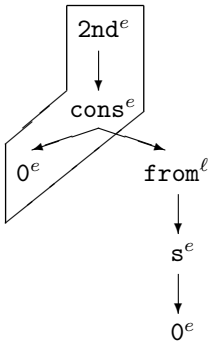

Figure 6: The active part of the expression `2nd(cons(0,from(0)))` w.r.t. lazy rewriting.

$\texttt{2nd}^e(\texttt{cons}^e(\texttt{0}^e,\texttt{from}^\ell(\texttt{s}^e(\texttt{0}^e))))$. *Thus,* $\Lambda, 1, 1.1, 1.2.1$, *and* $1.2.1.1$ *are eager positions; position* 1.2 *is lazy (see Figure 6).*

Given $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$, *erase*$(t)$ is the term in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ that results from removing the labels of t.

As remarked above, given $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$, a position $p \in \mathcal{P}os(t)$ is eager (resp. lazy) if $root(t|_p)$ is labeled with e (resp. ℓ). The so-called *active* positions of a labeled term $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$, denoted by $\mathcal{A}ct(t)$, are those positions which are always reachable from the root of the term via a path of eager positions. For instance, positions $\Lambda, 1$, and 1.1 are active in term t of Example 17; positions $1.2.1$ and $1.2.1.1$ are eager but *not* active, since position 1.2 below is lazy in t. In lazy rewriting, *the set of active nodes may increase as reduction of labeled terms proceeds.* Each lazy reduction step on labeled terms may have two different effects:

1. changing the "activation" status of a given position within a term, or
2. performing a rewriting step (always on an active position).

The *activation* status of a lazy position immediately below an active position within a (labeled) term can be modified if the position is 'essential', i.e. *'its contraction may lead to new redexes at active nodes'* [FKW00].

Definition 4 (Matching modulo laziness [FKW00]) *Let* $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *be linear,* $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$*, and* p *be an active position of* t*. Then,* l *matches modulo laziness* $s = t|_p$ *if either* $l \in \mathcal{X}$*, or* $l = f(l_1, \ldots, l_k)$*,* $s = f^e(s_1, \ldots, s_k)$ *and, for all* $i \in \{1, \ldots, k\}$*, if* $p.i$ *is eager, then* l_i *matches modulo laziness* s_i*. If position* $p.i$ *is lazy and* $l_i \notin \mathcal{X}$*, then position* $p.i$ *is called* essential. □

If p is an active position in $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$ and $l \to r$ is a rewrite rule of a left-linear TRS $\mathcal{R}$ such that l matches modulo laziness $t|_p$ giving rise to an essential position q of t and $t|_q = f^\ell(t_1, \ldots, t_k)$, then we write $t \xrightarrow{\mathsf{A}} t[f^e(t_1, \ldots, t_k)]_q$ for denoting the activation of position p.

Lazy rewriting reduces active positions. Let p be an active position of $t \in \mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$, $u = t|_p$ and $l \to r$ be a rule of a left-linear TRS $\mathcal{R}$ such that l matches $erase(u)$ using substitution σ, then, $t \xrightarrow{\mathsf{R}}_{\mu} s$, where s is obtained from t by replacing $t|_p$ in t by $label_{\mu}(r)$ with all its variables instantiated according to σ but preserving its label according to $label_{\mu}(r)$ (see [Luc02b] for a formal definition).

Example 18 *Consider the program of Example 1 (as a TRS) and the term t of Example 17. The reduction step for t corresponds to:*

$$\begin{aligned}
&\mathtt{2nd}^e(\mathtt{cons}^e(\mathtt{0}^e,\mathtt{from}^\ell(\mathtt{s}^e(\mathtt{0}^e))))\\
&\quad \xrightarrow{\mathsf{A}} \mathtt{2nd}^e(\mathtt{cons}^e(\mathtt{0}^e,\mathtt{from}^e(\mathtt{s}^e(\mathtt{0}^e))))\\
&\quad \xrightarrow{\mathsf{R}}_{\mu} \mathtt{2nd}^e(\mathtt{cons}^e(\mathtt{0}^e,\mathtt{cons}^e(\mathtt{s}^e(\mathtt{0}^e),{}^e\mathtt{from}^\ell(\mathtt{s}^e(\mathtt{s}^e(\mathtt{0}^e)))))))
\end{aligned}$$

Note that this last term is an $\xrightarrow{\mathsf{A}}$-normal form.

The lazy term rewriting relation on labeled *terms* (LR) is $\xrightarrow{\mathsf{LR}}_{\mu} = \xrightarrow{\mathsf{A}} \cup \xrightarrow{\mathsf{R}}_{\mu}$ and the evaluation $LR\text{-}eval_{\mu}(t)$ of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ using LR is given by $LR\text{-}eval_{\mu}(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid label_{\mu}(t) \xrightarrow{\mathsf{LR}!}_{\mu} s\}$. We say that a TRS is $LR(\mu)$-terminating if, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, there is no infinite $\xrightarrow{\mathsf{LR}}_{\mu}$-rewrite sequence starting from $label_{\mu}(t)$ [Luc02b].

In the following, we show that each evaluation step of our refined on-demand strategy is included into some evaluation steps of lazy rewriting. First, we give some auxiliary definitions and results for Theorem 4. Given a term $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ and $p \in \mathcal{P}os(t)$, we translate the labeling of terms in $\mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ into the labeling of $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ by considering only positive annotations and transforming overlined symbols and the symbols at the position under consideration into eager symbols, as follows:

$lazy^p_{\varphi}(t) = \rho''_p(\rho'_t(label_{\mu^{\varphi^{\mathbb{N}}}}(erase(t))))$ where

(1) $\rho'_t(f^b(t_1,\ldots,t_n)) = f^e(\rho'_{s_1}(t_1),\ldots,\rho'_{s_n}(t_n))$ if $t = \overline{f}_{L_1|L_2}(s_1,\ldots,s_n)$,

(2) $\rho'_t(f^b(t_1,\ldots,t_n)) = f^b(\rho'_{s_1}(t_1),\ldots,\rho'_{s_n}(t_n))$ if $t = f_{L_1|L_2}(s_1,\ldots,s_n)$,

(3) $\rho''_p(s) = s[f^e(s_1,\ldots,s_k)]_p$ for $s|_p = f^b(s_1,\ldots,s_k)$

We define the ordering $\leq_{lazy}$ between terms $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ by extending the ordering $f^e \leq_{lazy} f^e$ and $f^\ell \leq_{lazy} f^e$, for all $f \in \mathcal{F}$, to terms in the obvious way.

The following theorem shows that each evaluation step of our refined on-demand strategy corresponds to some evaluation steps of lazy rewriting. Also, it shows that lazy rewriting (potentially) activates as many symbols (within a term) as our strategy does (we use the ordering $\leq_{lazy}$ for expressing this fact). Note that lazy rewriting is defined only for left-linear TRSs (see Definition 4).

Theorem 4 *Let $\mathcal{R}$ be a left-linear TRS and φ be an E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, $p \in \mathcal{P}os(t)$ and $\mu = \mu^{\varphi^{\mathbb{N}}}$. If $\langle t, p\rangle \xrightarrow{\sharp}_{\varphi} \langle s, q\rangle$ and $p \in \mathcal{A}ct(lazy^p_{\varphi}(t))$,*

then $q \in \mathcal{A}ct(lazy^q_\varphi(s))$ *and* $lazy^p_\varphi(t) \xrightarrow{\mathsf{LR}}{}^*_\mu s'$ *for* $s' \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$ *such that* $lazy^q_\varphi(s) \leq_{lazy} s'$.

In general, our strategy is strictly more restrictive than *LR* as the following example shows.

Example 19 *Consider the program* $\mathcal{R}$ *(as a TRS) and the E-strategy map* φ *of Example 2. Consider the replacement map* $\mu = \mu^\varphi$. *In Example 25 below, we prove that* $\mathcal{R}$ *is* φ*-terminating. However,* LR *enters an infinite reduction sequence starting with the expression* $label_\mu(\texttt{length'(from(0))})$*:*

$$\begin{aligned}
&\texttt{length'}^e(\texttt{from}^\ell(\texttt{0}^e)) \\
&\quad \xrightarrow{\mathsf{R}}_\mu \texttt{length}^e(\texttt{from}^\ell(\texttt{0}^e)) \\
&\quad \xrightarrow{\mathsf{A}} \texttt{length}^e(\texttt{from}^e(\texttt{0}^e)) \\
&\quad \xrightarrow{\mathsf{R}}_\mu \texttt{length}^e(\texttt{cons}^e(\texttt{0}^e, \texttt{from}^\ell(\texttt{s}^e(\texttt{0}^e)))) \\
&\quad \xrightarrow{\mathsf{R}}_\mu \texttt{s}^e(\texttt{length'}^e(\texttt{from}^\ell(\texttt{s}^e(\texttt{0}^e)))) \\
&\quad \xrightarrow{\mathsf{LR}}_\mu \cdots
\end{aligned}$$

Note that if no positive annotation is provided for an argument of a symbol, then LR *freely demands this argument. Then, in contrast to* φ *(where* $\varphi(\texttt{length}) = (0)$*),* LR *can evaluate position* 1 *in the expression* `length(from(0))`.

5.2 On-demand rewriting

A replacement map $\mu \in M_\mathcal{F}$ specifies which arguments of symbols in $\mathcal{F}$ may be reduced. In *context-sensitive rewriting* (*CSR* [Luc98]), we (only) rewrite subterms at *replacing* positions: t μ-rewrites to s, written $t \hookrightarrow_{\mathcal{R}(\mu)} s$ (or simply $t \hookrightarrow_\mu s$ or $t \hookrightarrow s$), if $t \xrightarrow{p}_\mathcal{R} s$ and $p \in \mathcal{P}os^\mu(t)$.

Example 20 *Consider* $\mathcal{R}$ *in Example 1 and the replacement map* $\mu(\texttt{s}) = \mu(\texttt{2nd}) = \mu(\texttt{from}) = \mu(\texttt{cons}) = \{1\}$ *(which corresponds to the strategy map* φ *of Example 1). Then, we have:*

`2nd(from(0))` $\hookrightarrow_\mu$ `2nd(cons(0,from(s(0))))`

where, since $\mu(\texttt{cons}) = \{1\}$, `2nd(cons(0,from(s(0))))` *cannot be further* μ*-rewritten.*

$\hookrightarrow_\mu$-normal forms are called μ-normal forms. A TRS $\mathcal{R}$ is μ-terminating if $\hookrightarrow_\mu$ is terminating (see [Luc01a]).

The *non-replacing* positions of a term t are denoted by $\overline{\mathcal{P}os^\mu}(t) = \mathcal{P}os(t) - \mathcal{P}os^\mu(t)$; we also use $\mathcal{L}azy_\mu(t) = minimal_\leq(\overline{\mathcal{P}os^\mu}(t))$ which covers the non-replacing positions of t, i.e., for all $p \in \overline{\mathcal{P}os^\mu}(t)$, there exists $q \in \mathcal{L}azy_\mu(t)$ such that $q \leq p$. Given a pair $\langle \mu, \mu_D \rangle$ of replacement maps μ and μ_D, *on-demand rewriting* (*ODR*) is defined as an extension of *CSR* (under μ), where *on-demand* reductions are also permitted according to μ_D. Given $f \in \mathcal{F}$, indices $j \in \mu_D(f)$ aim at enabling reductions on a subterm t_j of a function call $f(t_1, \ldots, t_j, \ldots, t_k)$

if they can eventually lead to match a pattern of a rule defining f (i.e., $l \to r \in R$ such that $root(t) = f$). After its formal definition, we will explain the notion and give an example. The chain of symbols lying on positions above/on $p \in \mathcal{P}os(t)$ is $prefix_t(\Lambda) = root(t)$, $prefix_t(i.p) = root(t).prefix_{t|_i}(p)$. The strict prefix *sprefix* is $sprefix_t(\Lambda) = \Lambda$, $sprefix_t(p.i) = prefix_t(p)$.

Definition 5 (On-demand rewriting) [Luc01a] *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS and $\mu, \mu_D \in M_\mathcal{F}$. Then, $t \overset{p}{\hookrightarrow}_{\langle\mu,\mu_D\rangle} s$ (or simply $t \hookrightarrow_{\langle\mu,\mu_D\rangle} s$), if $t \overset{p}{\to} s$ and either*

1. *$p \in \mathcal{P}os^\mu(t)$, or*
2. *$p \in \mathcal{P}os^{\mu\sqcup\mu_D}(t) - \mathcal{P}os^\mu(t)$ and there exist $e \in \mathcal{P}os^\mu(t)$, $p_1, \ldots, p_n \in \mathcal{L}azy_{\langle\mu,\mu_D\rangle}(t)$, $r_1, \ldots, r_n, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \to r \in R$, and substitution σ such that*
   - *(1) $e \leq p$, $t' = t[r_1]_{p_1} \cdots [r_n]_{p_n}$, $t'|_e = \sigma(l)$ and*
   - *(2) for all $q \in \mathcal{P}os(l)$ s.t. $sprefix_{t|_e}(q) = sprefix_l(q)$, whenever $e.q \leq p$, we have that $l|_q \notin \mathcal{X}$.*

Here, $\mathcal{L}azy_{\langle\mu,\mu_D\rangle}(t) = \mathcal{L}azy_\mu(t) \cap \mathcal{P}os^{\mu\sqcup\mu_D}(t)$. □

Therefore, given a term t, a rewriting step $t \overset{p}{\to} s$ is *on-demand* (w.r.t. μ and μ_D) if either

(i) $t \overset{p}{\hookrightarrow}_\mu s$, or

(ii) $t \overset{p}{\hookrightarrow}_{\mu\sqcup\mu_D} s$ and reducing $t|_p$ may contribute to a future μ-rewriting step at μ-replacing position e, using some rule $l \to r$.

Such a contribution is approximated by checking whether the replacement of some non-μ-replacing maximal subterms of t would eventually make the matching possible (condition 2(1) of Definition 5). On-demand indices in μ_D determine the positions (in $\mathcal{L}azy_{\langle\mu,\mu_D\rangle}(t)$) of the subterms of t that can be refined. Note that the position p on which the rewriting step is performed is always covered by some position $p_i \in \mathcal{L}azy_{\langle\mu,\mu_D\rangle}(t)$, i.e., $p_i \leq p$ and p_i is a position demanded by the lhs l, which is (possibly) applicable at position e. On the other hand, the position p is constrained to having no variable position of l covering p (condition 2(2) of Definition 5); otherwise, the reduction at $t|_p$ would not improve the matching.

Example 21 *Consider the TRS $\mathcal{R}$ of Example 5, the replacement map μ of Example 20 and the following on-demand replace map $\mu_D(\texttt{cons}) = \{2\}$ and $\mu(\texttt{s}) = \mu(\texttt{2nd}) = \mu(\texttt{from}) = \varnothing$ (where the union of μ and μ_D corresponds to the apt strategy map φ of Example 5). Now we have:*

$$\begin{aligned}
\texttt{2nd(}\underline{\texttt{from(0)}}\texttt{)} &\hookrightarrow_{\langle\mu,\mu_D\rangle} \texttt{2nd(cons(0,}\underline{\texttt{from(s(0))}}\texttt{))} \\
&\hookrightarrow_{\langle\mu,\mu_D\rangle} \underline{\texttt{2nd(cons(0,s(0):from(s(s(0)))))}} \\
&\hookrightarrow_{\langle\mu,\mu_D\rangle} \texttt{s(0)}
\end{aligned}$$

but

$$\texttt{2nd(cons(0,cons(s(0),}\underline{\texttt{from(s(s(0)))}}\texttt{)))} \not\rightarrow_{\langle\mu,\mu_D\rangle} \texttt{2nd(cons(0,cons(s(0),cons(s(s(0)),from(s(s(s(0))))))))}$$

since $1.2.2 \in \mathcal{P}os(l)$ *and* $sprefix_t(1.2.2) = sprefix_l(1.2.2)$*, but* $l|_{1.2.2} \in \mathcal{X}$ *(where* $l = \texttt{2nd(x:y:z)}$*).*

In the following, we show that each evaluation step of our refined on-demand strategy is included in (at most) one evaluation step of on-demand rewriting. Given a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and a position $p \in \mathcal{P}os_A(t)$, we say the tuple $\langle t, p\rangle$ is *consistent* w.r.t. TRS $\mathcal{R}$ and strategy map φ (or simply *consistent*) if there exists $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\langle \varphi(s), \Lambda\rangle \xrightarrow{\sharp}{}^*_\varphi \langle t, p\rangle$.

Theorem 5 *Let* $\mathcal{R}$ *be a left-linear CS and* φ *be an E-strategy map such that* $\mu^{\varphi^\mathbb{N}}(c) = \varnothing$ *for* $c \in \mathcal{C}$*. Let* $\mu, \mu_D \in M_\mathcal{F}$ *be such that* $\mu = \mu^{\varphi^\mathbb{N}}$ *and* $\mu \sqcup \mu_D = \mu^\varphi$*. Let* $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ *and* $p \in \mathcal{P}os_A(t)$*. If* $\langle t, p\rangle \xrightarrow{\sharp}_\varphi \langle s, q\rangle$ *and* $\langle t, p\rangle$ *is consistent, then* $erase(t) \overset{p}{\hookrightarrow}_{=\langle\mu,\mu_D\rangle} erase(s)$*.*

Similarly to the *LR* case, our refined on-demand strategy is strictly more restrictive than *ODR* as the following example shows.

Example 22 *Consider the following* OBJ *program and its strategy map* φ*:*

```
obj Ex5 is
  sorts Nat LNat .
  op 0    : -> Nat .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat  [strat (-2)] .
  op from : Nat -> LNat       [strat (-1 0)] .
  op f    : LNat -> Nat       [strat (-1 0)] .
  op g    : Nat -> Nat        [strat (-1 0)] .
  vars X : Nat .
  eq from(X) = cons(X,from(s(X))) .
  eq f(X) = 0 .
  eq g(0) = 0 .
endo
```

Consider the TRS underlying this program and the replacement maps $\mu(\texttt{g}) = \mu(\texttt{f}) = \mu(\texttt{from}) = \mu(\texttt{cons}) = \varnothing$*,* $\mu_D(\texttt{g}) = \mu_D(\texttt{f}) = \mu_D(\texttt{from}) = \{1\}$*, and* $\mu_D(\texttt{cons}) = \{2\}$*; whose union corresponds to strategy map* φ*. The expression* $t = \texttt{g(f(from(0)))}$ *has a terminating evaluation sequence using* $\xrightarrow{\sharp}$*:*

$$\begin{aligned}
&\langle \texttt{g}_{nil|(\boxed{\texttt{-1}}\ 0)}(\texttt{f}_{nil|(-1\ 0)}(\texttt{from}_{nil|(-1\ 0)}(\texttt{0}_{nil|nil}))), \Lambda\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \texttt{g}_{(-1)|(\boxed{0})}(\texttt{f}_{nil|(-1\ 0)}(\texttt{from}_{nil|(-1\ 0)}(\texttt{0}_{nil|nil}))), \Lambda\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \texttt{g}_{(-1)|(0)}(\texttt{f}_{nil|(\boxed{\texttt{-1}}\ 0)}(\texttt{from}_{nil|(-1\ 0)}(\texttt{0}_{nil|nil}))), 1\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \texttt{g}_{(-1)|(0)}(\underline{\texttt{f}_{(-1)|(\boxed{0})}(\texttt{from}_{nil|(-1\ 0)}(\texttt{0}_{nil|nil}))}), 1\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \texttt{g}_{(-1)|(0)}(\texttt{0}_{nil|\boxed{nil}}), 1\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \underline{\texttt{g}_{(-1)|(\boxed{0})}(\texttt{0}_{nil|nil})}, \Lambda\rangle \\
&\quad\xrightarrow{\sharp}_\varphi \langle \texttt{0}_{nil|nil}, \Lambda\rangle
\end{aligned}$$

However, even if ODR *is able to reproduce the previous terminating reduction sequence:*

$$\texttt{g(}\underline{\texttt{f(from(0))}}\texttt{)} \hookrightarrow_{\langle\mu,\mu_D\rangle} \underline{\texttt{g(0)}} \hookrightarrow_{\langle\mu,\mu_D\rangle} \texttt{0}$$

the following non-terminating reduction sequence is also possible:

$$\texttt{g(f(}\underline{\texttt{from(0)}}\texttt{))} \hookrightarrow_{\langle\mu,\mu_D\rangle} \texttt{g(f(cons(0,}\underline{\texttt{from(s(0))}}\texttt{)))} \hookrightarrow_{\langle\mu,\mu_D\rangle} \cdots$$

Note that Λ *is a positive position which is a eventual redex of* `g(0)` *and positions* $1.1, 1.1.2, \ldots$ *are demanded by the lhs* `g(0)`*.*

Moreover, the condition in Theorem 5 that φ be an E-strategy map such that $\mu^{\varphi^{\mathbb{N}}}(c) = \varnothing$ for $c \in \mathcal{C}$ cannot be dropped.

Example 23 *Consider the following* OBJ *program and its strategy map* φ*:*

```
obj Ex6 is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat  [strat (2)] .
  op from : Nat -> LNat       [strat (-1 0)] .
  op f    : LNat -> Nat       [strat (-1 0)] .
  var X : Nat . var Z : LNat .
  eq from(X) = cons(X,from(s(X))) .
  eq f(cons(X,Z)) = 0 .
endo
```

Consider the TRS underlying this program and the replacement maps $\mu(\texttt{f}) = \mu(\texttt{from}) = \varnothing$*,* $\mu(\texttt{cons}) = \{2\}$*, and* $\mu_D(\texttt{f}) = \mu_D(\texttt{from}) = \{1\}$*, whose union corresponds to strategy map* φ*. Now the term* $t =$ `f(from(0))` *has a unique normalizing evaluation sequence under* ODR*:*

$$\texttt{f(}\underline{\texttt{from(0)}}\texttt{)} \hookrightarrow_{\langle\mu,\mu_D\rangle} \underline{\texttt{f(cons(0,from(s(0))))}} \hookrightarrow_{\langle\mu,\mu_D\rangle} \texttt{0}$$

Position 1.2 *of* `f(cons(0,from(s(0))))` *is under a variable of lhs* `f(cons(X,Z))` *and the subterm* `from(s(0))` *is not further evaluated. However, the evaluation sequence for* $\xrightarrow{\sharp}$ *is non-terminating:*

$$\begin{aligned}
&\langle \mathtt{f}_{nil|(\boxed{\texttt{-1}}\ 0)}(\mathtt{from}_{nil|(-1\ 0)}(\mathtt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{f}_{(-1)|(\boxed{0})}(\mathtt{from}_{nil|(-1\ 0)}(\mathtt{0}_{nil|nil})), \Lambda\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{f}_{(-1)|(0)}(\mathtt{from}_{nil|(\boxed{\texttt{-1}}\ 0)}(\mathtt{0}_{nil|nil})), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{f}_{(-1)|(0)}(\underline{\mathtt{from}_{(-1)|(\boxed{0})}(\mathtt{0}_{nil|nil})}), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{f}_{(-1)|(0)}(\mathtt{cons}_{nil|(\boxed{2})}(\mathtt{0}_{nil|nil}, \mathtt{from}_{(-1)|(0)}(\mathtt{s}_{nil|nil}(\mathtt{0}_{nil|nil})))), 1\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \langle \mathtt{f}_{(-1)|(0)}(\mathtt{cons}_{(2)|nil}(\mathtt{0}_{nil|nil}, \underline{\mathtt{from}_{(-1)|(\boxed{0})}(\mathtt{s}_{nil|nil}(\mathtt{0}_{nil|nil}))})), 1.2\rangle \\
&\quad \xrightarrow{\sharp}_{\varphi} \cdots
\end{aligned}$$

Note that since $\varphi(\texttt{cons}) =$ `(2)`*, each term* `from(`w`)` *has a non-terminating reduction sequence.*

In Theorem 5, the condition of $\mathcal{R}$ being a *CS* cannot be dropped. An argument similar to the one in Example 23 can be used if we consider a defined symbol in a non-root position of a lhs whose strategy map records a positive argument.

In the following section, we consider other aspects of the definition of a suitable on-demand evaluation strategy and formulate methods for proving termination of our on-demand strategy.

6 Proving termination of programs with negative annotations by transformation

In [Luc02b] a method for proving termination of *LR* as termination of *context-sensitive rewriting* (*CSR* [Luc98]) is described. In contrast to *LR*, context-sensitive rewriting forbids *every* reduction on the arguments not included into $\mu(f)$ for a given function call $f(t_1, \ldots, t_k)$. A TRS $\mathcal{R}$ is μ-terminating if the context-sensitive rewrite relation associated to $\mathcal{R}$ and μ is terminating. The idea of the aforementioned method is simple: given a TRS $\mathcal{R}$ and a replacement map μ, a new TRS $\mathcal{R}'$ and replacement map μ' is obtained in such a way that μ'-termination of $\mathcal{R}'$ implies $LR(\mu)$-termination of $\mathcal{R}$. Fortunately, there are a number of different techniques for proving termination of *CSR* (see [GM03, Luc02c] for recent surveys) which provide a formal framework for proving termination of lazy rewriting. A simple modification of such transformation provides a sound technique for proving φ-termination of TRSs for arbitrary strategy annotations φ by taking into account that only those symbols which have associated a negative index may be activated by demandedness. Here, as in [Luc01a, Luc02b], by φ-termination of a TRS $\mathcal{R}$ we mean the absence of infinite $\xrightarrow{\sharp}_{\varphi}$-sequences of terms starting from $\langle \varphi(t), \Lambda \rangle$.

As for the transformation in [Luc02b], the idea is to encode the demandedness information expressed by the rules of the TRS $\mathcal{R}$ together with the (negative) annotations of the E-strategy map φ as new symbols and rules (together with the appropriate modification/extension of φ) in such a way that φ-termination is preserved in the new TRS and E-strategy map, but the negative indices are finally suppressed (by removing from the lhs of the rules the parts that introduce on-demand computations). We iterate on these basic transformation steps until obtaining a canonical E-strategy map. In this case, we can stop the transformation and use the existing methods for proving termination of *CSR*. Let φ be an arbitrary E-strategy map. Given $l \to r \in R$ and $p \in \mathcal{P}os(l)$, we let

$$\mathcal{I}(l,p) = \{i > 0 \mid p.i \in \mathcal{P}os_{\mathcal{F}}(l) \text{ and } -i \in \varphi(root(l|_p))\}$$

Assume that $\mathcal{I}(l,p) = \{i_1, \ldots, i_n\}$ for some $n > 0$ (i.e., $\mathcal{I}(l,p) \neq \varnothing$) and let $f = root(l|_p)$. Then, $\mathcal{R}^\diamond = (\mathcal{F}^\diamond, R^\diamond)$ and $\varphi^\diamond$ are as follows: $\mathcal{F}^\diamond = \mathcal{F} \cup \{f_j \mid 1 \leq j \leq n\}$, where each f_j is a new symbol of arity $ar(f_j) = ar(f)$, and

$$R^\diamond = R - \{l \to r\} \cup \{l'_j \to r, l[x]_{p.i_j} \to l'_j[x]_{p.i_j} \mid 1 \leq j \leq n\}$$

where $l'_j = l[f_j(l|_{p.1}, \ldots, l|_{p.k})]_p$ if $ar(f) = k$, and x is a new variable. We let $\varphi^\diamond(f_j) = (i_j\ 0)$ for $1 \leq j \leq n$ and $f \in \mathcal{D}$, $\varphi^\diamond(f_j) = (i_j)$ for $1 \leq j \leq n$ and $f \in \mathcal{C}$, and $\varphi^\diamond(g) = \varphi(g)$ for all $g \in \mathcal{F}.(g \neq f)$. Moreover, we let $\varphi^\diamond(f) = \varphi^{\mathbb{N}}(f)$ if $\mu^{can}_{\mathcal{R}^\diamond}(f) \subseteq \mu^{\varphi^{\mathbb{N}}}(f)$, and $\varphi^\diamond(f) = \varphi(f)$ otherwise. Informally, if p is a position in a lhs l with a symbol f with a negative annotation $-i$ and position $p.i$ is a non-variable position in l, then we transform the rule $l \to r$ into $l[x]_p \to l'[x]_p$ and $l' \to r$; where l' is l with a new symbol f' at position p such that the annotation $-i$ is converted to i in the strategy for f' and removed from the strategy for f.

The transformation proceeds in this way (starting from $\mathcal{R}^\diamond$ and $\mu^\diamond$) until obtaining $\mathcal{R}^\natural = (\mathcal{F}^\natural, R^\natural)$ and $\varphi^\natural$ such that $\varphi^\natural = \varphi^{\natural^{\mathbb{N}}}$. If $\varphi = \varphi^{\mathbb{N}}$, then $\mathcal{R}^\natural = \mathcal{R}$ and $\varphi^\natural = \varphi$. Finally, we can state a sufficient condition for φ-termination as termination of *CSR* for the transformed TRS.

Theorem 6 (Termination) *Let $\mathcal{R}$ be a TRS, φ be an E-strategy map. If $\mathcal{R}^\natural$ is $\mu^{\varphi^\natural}$-terminating, then $\mathcal{R}$ is φ-terminating.*

It is well-known that *CSR* does not completely capture the φ-termination property of an OBJ program with only positive strategy annotations (see [Luc01b]). Thus, the technique proposed in the paper does not completely capture the φ-termination of an OBJ program with on-demand strategy annotations.

In the following, we show how some examples along the paper can be proved terminating by this technique.

Example 24 *Consider the TRS $\mathcal{R}$ associated to Example 5 but changing $\varphi(\texttt{cons}) = (1\ -2)$. Then, $\mathcal{R}^\natural$ is:*

```
2nd(cons'(x,cons(y,z))) → y
2nd(cons(x,y))          → 2nd(cons'(x,y))
from(x)                 → cons(x,from(s(x)))
```

and $\varphi^\natural$ is given by $\varphi^\natural(\texttt{2nd}) = \varphi^\natural(\texttt{from}) = (1\ 0)$, $\varphi^\natural(\texttt{cons}) = (1)$, and $\varphi^\natural(\texttt{cons'}) = (2)$. The $\mu^{\varphi^\natural}$-termination of $\mathcal{R}^\natural$ is proved by using Zantema's transformation for proving termination of CSR *[Zan97]: the TRS*

```
2nd(cons'(x,cons(y,z))) → y
2nd(cons(x,y))          → 2nd(cons'(x,activate(y)))
from(x)                 → cons(x,from'(s(x)))
activate(from'(x))      → from(x)
from(x)                 → from'(x)
activate(x)             → x
```

which is obtained in this way (where `activate` *and* `from'` *are new symbols introduced by Zantema's transformation) is terminating*[8].

Example 25 *Consider the TRS $\mathcal{R}$ and the E-strategy map φ that correspond to the* OBJ *program of Example 2. Our transformation returns the original TRS, i.e., $\mathcal{R}^\natural$ is:*

[8] This can be proven using the C*i*ME 2.0 system [EC96] (available at `http://cime.lri.fr`).

ms./rewrites	`pi`
OnDemandOBJ	25/364
CafeOBJ	30/364
OBJ3	*unavailable*
Maude	*unavailable*

Table 1: Execution of call `pi(square(square(3)))`.

ms./rewrites	`msquare_eager`	`msquare_apt`	`msquare_neg`
OnDemandOBJ	33/ 715	62/ 1640	0/ 1
	40/ 914	78/ 1992	80/ 1992
CafeOBJ	40/ 715	50/ 715	0/ 1
	50/ 914	60/ 914	60/ 914
OBJ3	20/ 715	*overflow*	*unavailable*
	30/ 914	*overflow*	*unavailable*
Maude	0/ 715	0/ 1640	*unavailable*
	0/ 914	3/ 1992	*unavailable*

Table 2: Execution of terms `minus(0,square(square(5)))` and `minus(square(square(5)),square(square(3)))`.

```
from(x)           → cons(x,from(s(x)))
length(nil)       → 0
length(cons(x,z)) → s(length'(z))
length'(z)        → length(z)
```

together with the simplified E-strategy $\varphi^\natural(\mathtt{s}) = \varphi^\natural(\mathtt{cons}) = (1)$, $\varphi^\natural(\mathtt{from}) = (1\ 0)$ *and* $\varphi^\natural(\mathtt{length}) = \varphi^\natural(\mathtt{length'}) = (0)$. *The* $\mu^{\varphi^\natural}$*-termination of* $\mathcal{R}$ *can be automatically proved by splitting up the rules of the program into two modules* $\mathcal{R}_1$ *(consisting of the rule for* `from`*) and* $\mathcal{R}_2$ *(consisting of the rules for* `length` *and* `length'`*). The* $\mu^{\varphi^\natural}$*-termination of* $\mathcal{R}_1$ *can easily be proved by using Zantema's transformation (in fact, the proof can be extracted from that of Example 24). The* $\mu^{\varphi^\natural}$*-termination of* $\mathcal{R}_2$ *is easily proved: in fact,* $\mathcal{R}_2$ *can be proved terminating (regarding standard rewriting) by using a polynomial ordering*[9]*. Now,* $\mu^{\varphi^\natural}$*-termination of* $\mathcal{R}$ *follows by applying the modularity results of [GL02].*

7 Experiments

In order to demonstrate the practicality of the on-demand evaluation strategy proposed in this paper, an interpreter has been implemented in Haskell (using ghc 5.04.2). The system is called OnDemandOBJ and is publicly available at `http://www.dsic.upv.es/users/elp/soft.html`.

Tables 1, 2, and 3 show the runtimes[10] and the number of rewrite steps

[9] *C*i*ME 2.0 [EC96] can also be used for achieving this proof.*

[10] The average of 10 executions measured in a Pentium III machine running RedHat 7.2.

ms./rewrites	quicksort	minsort	mod	mod'	average
OnDemandOBJ	55/1373	87/1649	540/13661	135/3117	70/1399
CafeOBJ	42/ 658	*overflow*	180/ 3117	175/3117	130/1399

Table 3: Comparison of CafeOBJ and OnDemandOBJ.

of the benchmarks for the different OBJ-family systems (source programs are included in Appendix B). CafeOBJ[11] (we use version 1.4.6) is developed in Lisp at the Japan Advanced Inst. of Science and Technology (JAIST); OBJ3[12] (we use version 2.0), also written in Lisp, is maintained by the University of California at San Diego; Maude[13] (we use version 1.0.5) is developed in C++ and maintained by the University of Illinois at 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 [NO01, OF00]. OnDemandOBJ computes with negative annotations using the on-demand evaluation strategy provided in this paper. Note that CafeOBJ and OBJ3 implement sharing of variables whereas Maude and OnDemandOBJ do not. It is worth noting that the mark *overflow* in Tables 2 and 3 indicates that the execution raised a memory overflow and normal form was not achieved whereas the mark *unavailable* in Tables 1 and 2 indicates that the program can not be executed in such OBJ implementation. Note that since Maude is implemented in C++, typical execution times are nearly 0 milliseconds.

The benchmark `pi` encodes the well-known infinite serie expansion to approximate number π of Example 3. Table 1 compares the evaluation of expression `pi(square(square(3)))` using existing OBJ implementations. It witnesses that negative annotations are actually useful in practice and that the implementation of the on-demand evaluation strategy in other systems is quite promising.

On the other hand, Table 2 illustrates the interest of using negative annotations to improve the behavior of programs: the benchmark `msquare_eager` encodes the functions `square`, `minus`, `times`, and `plus` over natural numbers using only positive annotations. Every k-ary symbol f is given a strategy $(1\ 2\ \cdots k\ 0)$ (this corresponds to *default* strategies in Maude). Note that the program is terminating as a TRS (i.e., without any 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 [AL02]). The benchmark `msquare_neg` is similar to `msquare_apt`, 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 $-i$

[11] Available at `http://www.ldl.jaist.ac.jp/Research/CafeOBJ/system.html`.
[12] Available at `http://www.kindsoftware.com/products/opensource/obj3/OBJ3/`.
[13] Available at `http://maude.cs.uiuc.edu/`.

is given to f (see [AL02]). Then, for instance, program `msquare_neg` runs in less time and requires a smaller number of rewrite steps than `msquare_eager` or `msquare_apt`, which do not include negative annotations.

Finally, Table 3 compares the execution of typical functional programs with canonical arbitrary strategies in OnDemandOBJ and in CafeOBJ, and demonstrates that there are clear advantages in using our implementation of the on-demand evaluation. We have used benchmarks `quicksort`, `minsort`, `mod`, and `average` which are borrowed from [AG01], and use canonical arbitrary strategies. Benchmark `mod'` is similar to `mod` but extra annotations are provided in order to avoid differences due to sharing. These experimental results, together with the OBJ source programs, are available at

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

8 Conclusions

We have provided a suitable extension of the positive E-evaluation strategy of OBJ-like languages to general (positive as well as negative) annotations. Such an extension is conservative, i.e., programs which only use positive strategy annotations and that are executed under our strategy behave exactly as if they were executed under the standard OBJ evaluation strategy (Theorems 1, 2, and 5). The main contributions of the paper are:

(a) the definition of a suitable and well-defined approach to demandedness via E-strategies (see Examples 2, 6, 14, and 19 for motivation regarding the undecidability of the reduction relation or the inadequacy of the model in previous proposals),

(b) the demonstration of the sound computational properties associated to such a new on-demand strategy (Theorems 3 and 4),

(c) the definition of techniques for analyzing computational properties such as termination under annotated strategies (Theorem 6),

(d) the experimental results of Section 7 demonstrate that our approach is better suited for implementation.

We also show that our on-demand strategy improves the two most important evaluation strategies dealing with on-demand annotations:

- *lazy rewriting* (LR) [FKW00], a popular, demand-driven technique to perform lazy functional computations which inspired the development of on-demand strategies in OBJ, and
- *on-demand rewriting* (ODR) [Luc01a], which extends the context sensitive rewriting of [Luc98] by also considering "negative annotations" and which does not directly apply to OBJ and is not comparable to LR.

As future work, we plan to extend the program transformation developed in [AEL02], which provides completeness of the evaluation strategy for positive strategy annotations, to the case of on-demand strategy annotations.

References

[AEGL02] 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.

[AEL02] M. Alpuente, S. Escobar, and S. Lucas. Correct and complete (positive) strategy annotations for OBJ. In F. Gadducci, editor, *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.

[AFJV97] 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.

[AG01] T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical report, AIB-2001-09, RWTH Aachen, Germany, 2001.

[AL02] 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.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[CELM96] 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, RWLW 96*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89. Elsevier Sciences Publisher, 1996.

[EC96] C. Marché . RTA 1996: 416-419 E. Contejean. CiME: Completion Modulo E. In H. Ganzinger, editor, *Proc. of 7th International Conference on Rewriting Techniques and Applications, RTA'96*, vol-

ume 1103 of *Lecture Notes in Computer Science*, pages 416–419. Springer-Verlag, Berlin, 1996.

[Eke00] 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.

[FGJM85] 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.

[FKW00] W. Fokkink, J. Kamperman, and P. Walters. Lazy rewriting on eager machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45–86, 2000.

[FN97] 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.

[GL02] B. Gramlich and S. Lucas. Modular termination of context-sensitive rewriting. In C. Kirchner, editor, *Proc. of 4th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'02*. ACM Press, New York, 2002.

[GM03] Jürgen Giesl and Aart Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 2003. To appear.

[GWM+00] 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.

[Luc98] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1–61, 1998.

[Luc01a] 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.

[Luc01b] S. Lucas. Termination of Rewriting With Strategy Annotations. In R. Nieuwenhuis and A. Voronkov, editors, *Proc. of 8th International*

Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'01, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 669–684. Springer-Verlag, Berlin, 2001.

[Luc02a] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[Luc02b] 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.

[Luc02c] S. Lucas. Termination of (Canonical) Context-Sensitive Rewriting. In S. Tison, 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.

[MR92] 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.

[Nag99] T. Nagaya. *Reduction Strategies for Term Rewriting Systems*. PhD thesis, School of Information Science, Japan Advanced Institute of Science and Technology, March 1999.

[NO01] 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.

[OF00] 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.

[TeR03] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.

[Zan97] 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

A.1 Proofs of Section 4.1

Theorem 1 *Let $\mathcal{R}$ be a TRS and φ be a positive E-strategy map. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os(t)$. Then, $\langle t, p\rangle \xrightarrow{\sharp}_\varphi \langle s, q\rangle$ if and only if $\langle positive(t), p\rangle \xrightarrow{\mathbb{N}}_\varphi \langle positive(s), q\rangle$.*

Proof. Straightforward according to Definition 1. □

We introduce some auxiliary notation. Let φ_1, φ_2 be two E-strategy maps such that $\varphi_1 \sqsubseteq \varphi_2$ and $t \in \mathcal{T}(\mathcal{F}^\sharp_{\varphi_2}, \mathcal{X}^\sharp_{\varphi_2})$. We define $<t>_{\varphi_1} = t' \in \mathcal{T}(\mathcal{F}^\sharp_{\varphi_1}, \mathcal{X}^\sharp_{\varphi_1})$ such that for all $p \in \mathcal{P}os(t)$, $root(t|_p) = f^\sharp_{L_1|L_2}$ implies that $root(t'|_p) = f^\sharp_{L'_1|L'_2}$ and that L'_1 and L'_2 are the maximal sequences such that $L'_1 \sqsubseteq L_1$, and $L'_2 \sqsubseteq L_2$.

Lemma 1 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map such that $\varphi^\mathbb{N} \in CM_\mathcal{R}$. If $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, then $OD_\mathcal{R}(t) = \varnothing$.*

Proof. Immediate. From $\varphi^\mathbb{N} \in CM_\mathcal{R}$, we get $DP_\mathcal{R}(erase(t)) \subseteq Paths_P(<t>_{\varphi^\mathbb{N}}) \subseteq Paths_P(t)$. Thus, $OD_\mathcal{R}(t) = OD_\mathcal{R}(<t>_{\varphi^\mathbb{N}}) = \varnothing$. □

Theorem 2 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map such that $\varphi^\mathbb{N} \in CM_\mathcal{R}$. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os(t)$. Then, $\langle t, p\rangle \xrightarrow{\sharp}_\varphi \langle s, q\rangle$ if and only if $\langle positive(t), p\rangle \xrightarrow{\mathbb{N}}_{\varphi^\mathbb{N}} \langle positive(s), q\rangle$.*

Proof. Immediate, since the term $positive(t)$ does not contain negative indices and, hence, by Lemma 1, $OD_\mathcal{R}(t) = \varnothing$. □

Theorem 3 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear CS, $\varphi \in CM_\mathcal{R}$ and $\varphi(f)$ end with 0 for all $f \in \mathcal{D}$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $s \in eval_\varphi(t)$, then s is a head-normal form of t.*

Proof. First, note that it is not possible to have that $root(s) = \overline{f}_{L|nil}$ for $f \in \mathcal{F}$ since non-evaluable flags are raised only when a position is demanded and only for those symbols occurring at positions between the root and the considered demanded position (excluding both).

We prove the claim by structural induction on s. If $s \in \mathcal{C}$ or $s \in \mathcal{X}$, we are trivially done. Consider $s = f \in \mathcal{D}$, with $ar(f) = 0$. By assumption, $\varphi(f)$ ends with 0, thus the last rewriting step was $\langle f_{nil|(0)}, \Lambda\rangle \xrightarrow{\sharp}_\varphi \langle f_{nil|nil}, \Lambda\rangle$. The only case when this can happen is when $erase(f_{nil|(0)}) = f$ is not a redex and $OD_\mathcal{R}(f_{nil|(0)}) = \varnothing$. But this case can only occur if there is no $l \in L(\mathcal{R}).root(l) = f$. Hence, s is a head-normal form.

For the induction case, we omit the case $root(s) \in \mathcal{C}$ which is trivial. Consider $root(s) = f \in \mathcal{D}$. By assumption, $\varphi(f)$ ends with 0, thus, there are terms $t', s' \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ such that the last rewriting step was $\langle t', \Lambda\rangle \xrightarrow{\sharp}_\varphi \langle s', \Lambda\rangle$, $s = erase(s')$, $root(t') = f_{L|(0)}$ and $s' = f_{L|nil}(t'|_1, \ldots, t'|_{ar(f)})$. This can happen only when $erase(t')$ is not a redex and $OD_\mathcal{R}(t') = \varnothing$.

If $erase(t')$ is not a redex, then s is also not a redex and, by left-linearity, $\nexists l \in L(\mathcal{R})$ and $\sigma \in Subst(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ such that $s = \sigma(l)$, i.e. $Pos_{\neq}(s, l) \neq \varnothing$ for all $l \in L(\mathcal{R})$. Moreover, if $ADP_\mathcal{R}(s) = \varnothing$, then s is a head-normal form because either there is no $l \in L(\mathcal{R}).root(s) = root(l)$ (and then there is no rule which can be applied to s) or there is $p \in \mathcal{P}os(s)$ such that for all $l \in L(\mathcal{R})$, $p \in \mathcal{P}os_{\neq}(s, l)$ and $root(s|_p) \notin \mathcal{D}$ (and then, by the CS property, we know that the symbol at this position will never change by further reductions on the term). On the contrary, if there is $p \in DP_\mathcal{R}(s)$, then there exists $l \in L(\mathcal{R})$ such that $p \in DP_l(s)$. In this case, since $OD_\mathcal{R}(t') = \varnothing$, either $p \notin Paths_A(t')$ or $p \in Paths_P(t')$. But, since $\varphi \in CM_\mathcal{R}$, all symbols in l at positions which are above p have an index ($-i$ or i) in φ; thus, we have $p \in Paths_A(t')$ and $p \in Paths_P(t')$. Then, since all indices from Λ down to p are positive, position p was previously reduced and now, $root(t'|_p) = g_{L'|nil}$ for $g \in \mathcal{F} \cup \mathcal{X}$. Thus, by induction hypothesis, $s|_p$ is a head-normal form and, by CS property, the symbol g at this position will never disappear. Hence, s is also a head-normal form. □

A.2 Proofs of Section 5.1

Lemma 2 *Let $\mathcal{R}$ be a left-linear TRS and φ be an E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. If $OD_\mathcal{R}(t) = \{p\}$, then $\exists l \in L(\mathcal{R})$ such that l matches modulo laziness $label_{\mu^{\varphi^\mathbb{N}}}(erase(t))$; $root(erase(t)|_p) \neq root(l|_p) \notin \mathcal{X}$; for all $p' : \Lambda \leq p' < p$, $root(erase(t)|_{p'}) = root(l|_{p'})$; and at least one position $p' : \Lambda < p' \leq p$ is declared essential.*

Proof. By definition, there exists $l \in L(\mathcal{R})$ such that $p \in \mathcal{P}os_{\neq}(erase(t), l)$ and $root(erase(t)|_p) \neq root(l|_p) \notin \mathcal{X}$. By minimality, for all $p' : \Lambda \leq p' < p$, $root(erase(t)|_{p'}) = root(l|_{p'})$. Moreover, we have that $OD_\mathcal{R}(erase(t)) \cap Paths_P(t) = \varnothing$. That is, for all $q \in DP_\mathcal{R}(erase(t))$, $q \notin Paths_P(t)$ and then, there is $q' \leq q$ such that $q' \in Paths_A(t)$, $root(label_{\mu^{\varphi^\mathbb{N}}}(erase(t))|_{q'}) = f^\ell$ and for all $q'' : \Lambda \leq q'' < q'$, symbol $root(label_{\mu^{\varphi^\mathbb{N}}}(erase(t))|_{q''})$ is marked as eager. Hence, the conclusion follows and l matches modulo laziness t. □

Lemma 3 *Let $\mathcal{R}$ be a left-linear TRS and φ be an E-strategy map. If $t, l', r' \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $l \to r \in \mathcal{R}$ such that $t = \sigma(l')$, $erase(l') = l$, and $r' = \sigma(\varphi(r))$, then l matches $erase(t)$ and, let $t^\varphi = label_{\mu^{\varphi^\mathbb{N}}}(erase(t))$, there exists θ for* LR *such that $label_{\mu^{\varphi^\mathbb{N}}}(erase(r')) = \theta(label_{\mu^{\varphi^\mathbb{N}}}(r))$.*

Proof. Note that all variables of l' have the same labelling, i.e. $\mathcal{V}ar(l') = \{x_{nil|nil} \mid x \in \mathcal{V}ar(l)\}$. Note also that, if $t = \sigma(l')$, then l matches $erase(t)$

and there exists θ for LR such that $erase(\sigma(x')) = erase(\theta(x''))$ for $erase(x') = erase(x'') = x \in \mathcal{V}ar(l)$. Finally, $\theta(label_{\mu^{\varphi^{\mathbb{N}}}}(r)) = label_{\mu^{\varphi^{\mathbb{N}}}}(erase(\sigma(r)))$, i.e. $label_{\mu^{\varphi^{\mathbb{N}}}}(erase(r')) = \theta(label_{\mu^{\varphi^{\mathbb{N}}}}(r))$. □

Theorem 4 *Let $\mathcal{R}$ be a left-linear TRS and φ be an E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, $p \in \mathcal{P}os(t)$ and $\mu = \mu^{\varphi^{\mathbb{N}}}$. If $\langle t, p\rangle \xrightarrow{\sharp}_{\varphi} \langle s, q\rangle$ and $p \in \mathcal{A}ct(lazy^p_{\varphi}(t))$, then $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$ and $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{*}_{\mu} s'$ for $s' \in \mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ such that $lazy^q_{\varphi}(s) \leq_{lazy} s'$.*

Proof. We consider the different cases of Definition 3 separately.

1. If $t|_p = f_{L|nil}(t_1, \ldots, t_k)$, $s = t$ and $p = q.i$ for some i, then $lazy^q_{\varphi}(s) \leq_{lazy} lazy^p_{\varphi}(t)$ and $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^p_{\varphi}(t)$. Note that $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$ since $p = q.i \in \mathcal{A}ct(lazy^p_{\varphi}(t))$.
2. If $t|_p = f_{L_1|i:L_2}(t_1, \ldots, t_k)$ with $i > 0$, $s = t[f_{L_1@i|L_2}(t_1, \ldots, t_k)]_p$ and $q = p.i$, then $lazy^p_{\varphi}(t) = lazy^q_{\varphi}(s)$ and, since $i \in \mu(f)$, $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$, we get $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^q_{\varphi}(s)$.
3. If $t|_p = f_{L_1|-i:L_2}(t_1, \ldots, t_k)$ with $i > 0$, $s = t[f_{L_1@-i|L_2}(t_1, \ldots, t_k)]_p$ and $q = p$, then $lazy^p_{\varphi}(t) = lazy^q_{\varphi}(s)$, $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$ and $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^q_{\varphi}(s)$.
4. If $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k) = \sigma(l')$, $erase(l') = l$, $s = t[\sigma(\varphi(r))]_p$ for some $l \to r \in R$ and substitution σ, and $q = p$, then by Lemma 3, there exists θ for LR such that $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{R}}_{\mu} lazy^p_{\varphi}(t)[\theta(label_{\mu}(r))]_p$. By Lemma 3, we also have that $lazy^p_{\varphi}(t)[\theta(label_{\mu}(r))]_p = lazy^p_{\varphi}(t)[label_{\mu}(erase(\sigma(\varphi(r))))]_p$. Since $t|_p$ contains no overlined symbol and $p \in \mathcal{A}ct(lazy^p_{\varphi}(t))$, then $label_{\mu}(erase(\sigma(\varphi(r)))) = lazy^p_{\varphi}(\sigma(\varphi(r)))$, and we finally get $lazy^p_{\varphi}(t)[\theta(label_{\mu}(r))]_p = lazy^p_{\varphi}(t)[lazy^p_{\varphi}(\sigma(\varphi(r)))]_p = lazy^q_{\varphi}(s)$.
5. If $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $erase(t|_p)$ is not a redex, $OD_{\mathcal{R}}(t|_p) = \varnothing$, $s = t[f_{L_1|L_2}(t_1, \ldots, t_k)]_p$, and $q = p$, then $lazy^p_{\varphi}(t) = lazy^q_{\varphi}(s)$, $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$ and $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^q_{\varphi}(s)$.
6. If $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $erase(t|_p)$ is not a redex, $OD_{\mathcal{R}}(t|_p) = \{p'\}$, $s = t[mark(t|_p, p')]_p$, and $q = p.p'$, then, for all $p'' : p \leq p'' \leq p'$, $root(lazy^q_{\varphi}(s)|_{p''}) = f^e$ and $q \in \mathcal{A}ct(lazy^q_{\varphi}(s))$. Thus, it is not difficult to see that, since $root(erase(t)|_{p''}) = root(l|_{p''})$ for all $p'' : \Lambda \leq p'' < p'$, appropriate activation steps of lazy rewriting are available (by successive applications of Lemma 2) in order to activate the necessary symbols from the root down to q, i.e. we obtain $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{A}}{}^{*} lazy^q_{\varphi}(s)$.
7. If $t|_p = \overline{f}_{L_1|L_2}(t_1, \ldots, t_k)$, $s = t[f_{L_1|L_2}(t_1, \ldots, t_k)]_p$, and $p = q.i$ for some i, then, since $lazy^q_{\varphi}(s) \leq_{lazy} lazy^p_{\varphi}(t)$, we have $lazy^p_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^p_{\varphi}(t)$. Note that $q \in \mathcal{A}ct(lazy^q_{\varphi}(t))$ since $q.i \in \mathcal{A}ct(lazy^p_{\varphi}(t))$. □

A.3 Proofs of Section 5.2

Given a term $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ and a position $p \in \mathcal{P}os_A(t)$, we say p is a *stop* position if there is no sequence $\langle t, p\rangle \xrightarrow{\sharp}{}^{*}_{\varphi} \langle t', q\rangle$ such that $p \leq q$, $erase(t) = erase(t')$, and $erase(t'|_q)$ is a redex.

Lemma 4 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map such that $\mu^{\varphi^{\mathbb{N}}}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ and $p \in \mathcal{P}os_A(t)$. If $root(erase(t|_p)) \in \mathcal{C}$, then p is a stop position.*

Proof. Immediate since $\mu^{\varphi^{\mathbb{N}}}(c) = \varnothing$ for $root(erase(t|_p)) = c$ and then it has no effect, even if annotation $\mathtt{0}$ is included into $\varphi(c)$. □

Lemma 5 *Let $\mathcal{R}$ be a left-linear CS and φ be an E-strategy map such that $\mu^{\varphi^{\mathbb{N}}}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ and $p \in \mathcal{P}os_A(t)$. If $\langle t, p\rangle$ is consistent, then either $p \in \mathcal{P}os_P(t)$ or $\exists q \in \mathcal{P}os_P(t)$ s.t. $q < p$ and either $OD_{\mathcal{R}}(t|_q) \neq \varnothing$ or, otherwise, if $erase(t|_q)$ is a redex, then $\forall q < w \leq p$, w is a stop position.*

Proof. By induction on the length n of the evaluation sequence $\langle \varphi(s), \Lambda\rangle \xrightarrow{\sharp}{}^{n}_{\varphi} \langle t, p\rangle$ for $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

- $(n = 0)$ It is immediate to see that $p = \Lambda$ and $\Lambda \in \mathcal{P}os_P(t)$.
- $(n > 0)$ Let us consider $\langle \varphi(s), \Lambda\rangle \xrightarrow{\sharp}{}^{n-1}_{\varphi} \langle t', p'\rangle \xrightarrow{\sharp}_{\varphi} \langle t, p\rangle$. The induction hypothesis is: $p' \in \mathcal{P}os_P(t')$ or $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$ and $\exists q' \in \mathcal{P}os_P(t')$ s.t. $q' < p'$ and either $OD_{\mathcal{R}}(t'|_{q'}) \neq \varnothing$ or, otherwise, if $erase(t'|_{q'})$ is a redex, then for all q' s.t. $q' < w \leq p'$, w is a stop position. We consider the different cases of Definition 3 separately:

  1. Let $t'|_{p'} = f_{L|nil}(t_1, \ldots, t_k)$, $t = t'$ and $p' = p.i$ for some i. If $p' \in \mathcal{P}os_P(t')$, then $p \in \mathcal{P}os_P(t)$. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$ and $q' = p$, then $p \in \mathcal{P}os_P(t)$. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, and $OD_{\mathcal{R}}(t|_{q'}) \neq \varnothing$, then the conclusion follows since no symbol occurring above or at position p' has been changed. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, and $OD_{\mathcal{R}}(t|_{q'}) = \varnothing$, then the conclusion follows by induction since p is also a stop position.
  2. Let $t'|_{p'} = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $t = t'[f_{L_1@i|L_2}(t_1, \ldots, t_k)]_{p'}$ and $p = p'.i$. If $p' \in \mathcal{P}os_P(t')$, then $p \in \mathcal{P}os_P(t)$. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, and $OD_{\mathcal{R}}(t'|_{q'}) \neq \varnothing$, then the conclusion follows since no symbol occurring above or at position p has been changed. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, $OD_{\mathcal{R}}(t'|_{q'}) = \varnothing$, and $erase(t'|_{q'})$ is a redex, then p is a stop position if p' is.

3. Let $t'|_{p'} = f_{L_1|-i:L_2}(t_1,\ldots,t_k)$, $i > 0$, $t = t'[f_{L_1@-i|L_2}(t_1,\ldots,t_k)]_{p'}$ and $p = p'$. This case is straightforward since the symbols above and at position p' are unchanged.
4. Let $t'|_{p'} = f_{L_1|0:L_2}(t_1,\ldots,t_k) = \sigma(l')$, $erase(l') = l$, $t = t'[\sigma(\varphi(r))]_{p'}$ for some $l \to r \in R$ and substitution σ, and $p = p'$. If $p' \in \mathcal{P}os_P(t')$, then $p \in \mathcal{P}os_P(t)$. Otherwise, $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$ and $q' < p$. If $OD_{\mathcal{R}}(t|_{q'}) \neq \varnothing$ or $OD_{\mathcal{R}}(t|_{q'}) = \varnothing$ and $erase(t|_{q'})$ is not a redex, then the conclusion follows. Otherwise, $erase(t|_{q'})$ is a redex.

   Here, note that it is impossible that $OD_{\mathcal{R}}(t'|_{q'}) = \varnothing$ because in that case, either $erase(t'|_{q'})$ would not be a redex and, since $\mathcal{R}$ is a left-linear CS, it is impossible that $erase(t|_{q'})$ becomes a redex; or $erase(t'|_{q'})$ would be a redex and p' a stop position, but, then, no reduction could be performed at position p'. Thus, $OD_{\mathcal{R}}(t'|_{q'}) \neq \varnothing$, $OD_{\mathcal{R}}(t|_{q'}) = \varnothing$, and $erase(t|_{q'})$ is a redex. Now, since $\mathcal{R}$ is a CS, for all q' s.t. $q' < w \leq p$, $root(erase(t|_w)) \in \mathcal{C}$ and, by Lemma 4, w is a stop position.
5. Let $t'|_{p'} = f_{L_1|0:L_2}(t_1,\ldots,t_k)$, $erase(t'|_{p'})$ is not a redex, $OD_{\mathcal{R}}(t'|_{p'}) = \varnothing$, $t = t'[f_{L_1|L_2}(t_1,\ldots,t_k)]_{p'}$, and $p = p'$. Then, it is straightforward (see Case 3).
6. Let $t'|_{p'} = f_{L_1|0:L_2}(t_1,\ldots,t_k)$, $erase(t'|_{p'})$ is not a redex, $OD_{\mathcal{R}}(t'|_{p'}) = \{p''\}$, $t = t'[mark(t'|_{p'}, p'')]_{p'}$, and $p = p'.p''$. If $p' \in \mathcal{P}os_P(t')$, then $p' < p$, $OD_{\mathcal{R}}(t|_{p'}) \neq \varnothing$, and the conclusion follows. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, and $OD_{\mathcal{R}}(t'|_{q'}) \neq \varnothing$, then the conclusion follows since no symbol above p has been changed. If $p' \in \mathcal{P}os_A(t') - \mathcal{P}os_P(t')$, $q' < p$, $OD_{\mathcal{R}}(t'|_{q'}) = \varnothing$, and $erase(t'|_{q'})$ is a redex, then p is a stop position if p' is.
7. Let $t'|_{p'} = \overline{f}_{L_1|L_2}(t_1,\ldots,t_k)$, $t = t'[f_{L_1|L_2}(t_1,\ldots,t_k)]_{p'}$ and $p' = p.i$ for some i. This case is similar to case 1 above. □

Theorem 5 *Let $\mathcal{R}$ be a left-linear CS and φ be an E-strategy map such that $\mu^{\varphi^{\mathbb{N}}}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $\mu, \mu_D \in M_{\mathcal{F}}$ be such that $\mu = \mu^{\varphi^{\mathbb{N}}}$ and $\mu \sqcup \mu_D = \mu^{\varphi}$. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ and $p \in \mathcal{P}os_A(t)$. If $\langle t, p\rangle \xrightarrow{\sharp}_{\varphi} \langle s, q\rangle$ and $\langle t, p\rangle$ is consistent, then $erase(t) \overset{p}{\hookrightarrow}_{\overline{=}\langle \mu, \mu_D\rangle} erase(s)$.*

Proof. We consider only case 4 of Definition 3 since the other cases only manipulate annotations on symbols or compute the next position q to be considered, which implies a reflexive on-demand rewriting step. Then, by Lemma 5, case 4 can only occur under the following conditions:

1. If $p \in \mathcal{P}os_P(t)$, then we are trivially done.
2. If $p \in \mathcal{P}os_A(t) - \mathcal{P}os_P(t)$, $\exists q \in \mathcal{P}os_P(t)$ s.t. $q < p$ and $OD_{\mathcal{R}}(t|_q) \neq \varnothing$, then it is easy to prove that there exist $p_1,\ldots,p_n \in \mathcal{L}azy_{\langle \mu, \mu_D\rangle}(erase(t))$,

$r_1, \ldots, r_n, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \to r \in R$, and a substitution σ such that $t' = \mathit{erase}(t)[r_1]_{p_1} \cdots [r_n]_{p_n}$, $t'|_q = \sigma(l)$ and, for all $w \in \mathcal{P}os(l)$ s.t. $\mathit{sprefix}_{\mathit{erase}(t)|_q}(w) = \mathit{sprefix}_l(w)$ whenever $q.w \leq p$, hence we have that $l|_w \notin \mathcal{X}$.

3. Otherwise, $p \in \mathcal{P}os_A(t) - \mathcal{P}os_P(t)$, and there is no $q \in \mathcal{P}os_P(t)$ s.t. $q < p$, $OD_{\mathcal{R}}(t|_q) = \varnothing$, and $\mathit{erase}(t|_q)$ is a redex. Note that, by Definition 3, there exist $q \in \mathcal{P}os_P(t)$ and $l \in L(\mathcal{R})$ s.t. $q < p$ and $\mathcal{P}os_{\neq}(\mathit{erase}(t|_q), l) \neq \varnothing$ because, otherwise, it is impossible that position $p \notin \mathcal{P}os_P(t)$ is used for reduction. Thus, it is easy to prove that there exist $p_1, \ldots, p_n \in \mathcal{L}azy_{\langle \mu, \mu_D \rangle}(\mathit{erase}(t))$, $r_1, \ldots, r_n, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \to r \in R$, and substitution σ such that $t' = \mathit{erase}(t)[r_1]_{p_1} \cdots [r_n]_{p_n}$, $t'|_q = \sigma(l)$ and for all $w \in \mathcal{P}os(l)$ s.t. $\mathit{sprefix}_{\mathit{erase}(t)|_q}(w) = \mathit{sprefix}_l(w)$, whenever $q.w \leq p$, we have that $l|_w \notin \mathcal{X}$. □

A.4 Proofs of Section 6

Theorem 6 *Let $\mathcal{R}$ be a TRS, φ be an E-strategy map. If $\mathcal{R}^\sharp$ is $\mu^{\varphi^\sharp}$-terminating, then $\mathcal{R}$ is φ-terminating.*

Proof. By induction on the number n of transformation steps $\langle \mathcal{R}_0, \varphi_0 \rangle, \ldots, \langle \mathcal{R}_n, \varphi_n \rangle$, where $\mathcal{R}_0 = \mathcal{R}$ and $\varphi_0 = \varphi$. If $n = 0$, then $\varphi^\sharp = \varphi^{\sharp^{\mathbb{N}}}$ and, since $\mathcal{R}$ is $\mu^{\varphi^\sharp}$-terminating, by Theorem 2 of [Luc01b], $\mathcal{R}$ is φ-terminating. If $n > 0$, then, it is not difficult to see that for all $t, s \in \mathcal{T}(\mathcal{F}^\sharp_{\varphi_0}, \mathcal{X}^\sharp_{\varphi_0})$ and $p \in \mathcal{P}os(t)$, if $\langle t, p \rangle \xrightarrow{\sharp}_{\varphi_0} \langle s, q \rangle$, then $\langle <t>_{\varphi_1}, p \rangle \xrightarrow{\sharp}{}^*_{\varphi_1} \langle <s>_{\varphi_1}, q \rangle$. Hence, since $\mathcal{R}_1$ is φ_{n-1}-terminating, $\mathcal{R}_0$ is φ_n-terminating. □

B Code of benchmarks

B.1 Program `pi`

This program encodes the well-known infinite series expansion to approximate the π number of Example 3: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$. To make the program terminating and complete, the strategy for symbol `cons` must include the on-demand annotation `-2`. The remainder strategy annotations are positive since termination of the whole program is ensured (see termination proof in Section C). 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 2ndspos : Nat LNat -> LRecip                [strat (1 2 0)] .
  op 2ndsneg : Nat LNat -> LRecip                [strat (1 2 0)] .
  op pi : Nat -> LRecip                          [strat (1 0)] .
  op plus : Nat Nat -> Nat                       [strat (1 2 0)] .
  op times : Nat Nat -> Nat                      [strat (1 2 0)] .
  op square : Nat -> Nat                         [strat (1 0)] .
  vars N X Y : Nat . var Z : LNat .
  eq from(X) = cons(X,from(s(X))) .
  eq 2ndspos(0,Z) = rnil .
  eq 2ndspos(s(N),cons(X,cons(Y,Z))) = rcons(posrecip(Y),2ndsneg(N,Z)) .
  eq 2ndsneg(0,Z) = rnil .
  eq 2ndsneg(s(N),cons(X,cons(Y,Z))) = rcons(negrecip(Y),2ndspos(N,Z)) .
  eq pi(X) = 2ndspos(X,from(0)) .
  eq plus(0,Y) = Y .
  eq plus(s(X),Y) = s(plus(X,Y)) .
  eq times(0,Y) = 0 .
  eq times(s(X),Y) = plus(Y,times(X,Y)) .
  eq square(X) = times(X,X) .
endo
```

B.2 Program `msquare_eager`

This program uses functions `minus`, `square`, `times`, and `plus` over natural numbers; they are common to several examples included in the 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.3 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.4 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.5 Program quicksort

This program is borrowed from Example 3.11 of [AG01]. Note that auxiliary functions from and take for constructing lists are included, as well as two predicates nfLNat and nfNat to normalize terms, and the connective and. The term used for evaluation is: nfLNat(quicksort(take(10,from(0))))

```
obj Quicksort is
   sorts Nat LNat Bool2 .
   op 0 :  -> Nat .
   op s : Nat -> Nat                   [strat (1)] .
   op nil :  -> LNat .
   op cons : Nat LNat -> LNat          [strat (1)] .
   op true2 :  -> Bool2 .
   op false2 :  -> Bool2 .
   op le : Nat Nat -> Bool2            [strat (1 -2 0)] .
   op app : LNat LNat -> LNat          [strat (1 0)] .
   op low : Nat LNat -> LNat           [strat (2 0)] .
   op high : Nat LNat -> LNat          [strat (2 0)] .
   op ifLNat : Bool2 LNat LNat -> LNat [strat (1 0)] .
   op quicksort : LNat -> LNat         [strat (1 0)] .
   op and : Bool2 Bool2 -> Bool2       [strat (1 0)] .
   op nfLNat : LNat -> Bool2           [strat (1 0)] .
   op nfNat : Nat -> Bool2             [strat (1 0)] .
   op from : Nat -> LNat               [strat (0)] .
   op take : Nat LNat -> LNat          [strat (1 -2 0)] .
   vars X Y : Nat . vars Z W : LNat . vars A B : Bool2 .
```

```
   eq le(0,Y) = true2 .
   eq le(s(X),0) = false2 .
   eq le(s(X),s(Y)) = le(X,Y) .
   eq app(nil,Z) = Z .
   eq app(cons(X,Z),W) = cons(X,app(Z,W)) .
   eq low(X,nil) = nil .
   eq low(X,cons(Y,Z)) = ifLNat(le(Y,X),cons(Y,low(X,Z)),low(X,Z)) .
   eq high(X,nil) = nil .
   eq high(X,cons(Y,Z)) = ifLNat(le(Y,X),high(X,Z),cons(Y,high(X,Z))) .
   eq ifLNat(true2,Z,W) = Z .
   eq ifLNat(false2,Z,W) = W .
   eq quicksort(nil) = nil .
   eq quicksort(cons(X,Z)) = app(quicksort(low(X,Z)),
                                 cons(X,quicksort(high(X,Z)))) .
   eq from(X) = cons(X,from(s(X))) .
   eq take(0,Z) = nil .
   eq take(s(X),cons(Y,Z)) = cons(Y,take(X,Z)) .
   eq nfLNat(nil) = true2 .
   eq nfLNat(cons(X,Z)) = and(nfNat(X),nfLNat(Z)) .
   eq nfNat(0) = true2 .
   eq nfNat(s(X)) = nfNat(X) .
   eq and(true2,A) = A .
   eq and(false2,A) = false2 .
endo
```

B.6 Program `minsort`

This program is borrowed from Example 3.10 of [AG01]. The call considered for evaluation is: `nfLNat(minsort(take(10,from(0)),nil))`

```
obj Minsort is
   sorts Nat LNat Bool2 .
   op 0 :  -> Nat .
   op s : Nat -> Nat                    [strat (1)] .
   op nil :  -> LNat .
   op cons : Nat LNat -> LNat           [strat (1 -2)] .
   op true2 :  -> Bool2 .
   op false2 :  -> Bool2 .
   op le : Nat Nat -> Bool2             [strat (1 2 0)] .
   op app : LNat LNat -> LNat           [strat (1 0)] .
   op rm : Nat LNat -> LNat             [strat (2 0)] .
   op min : LNat -> Nat                 [strat (1 0)] .
   op eqNat : Nat Nat -> Bool2          [strat (1 2 0)] .
   op ifNat : Bool2 Nat Nat -> Nat      [strat (1 0)] .
   op ifLNat : Bool2 LNat LNat -> LNat [strat (1 0)] .
   op and : Bool2 Bool2 -> Bool2        [strat (1 0)] .
   op minsort : LNat LNat -> LNat       [strat (1 2 0)] .
   op nfLNat : LNat -> Bool2            [strat (1 0)] .
   op nfNat : Nat -> Bool2              [strat (1 0)] .
   op take : Nat LNat -> LNat           [strat (1 2 0)] .
```

```
   op from : Nat -> LNat               [strat (0)] .
   vars X Y : Nat . vars Z W : LNat . vars A B : Bool2 .
   eq le(0,Y) = true2 .
   eq le(s(X),0) = false2 .
   eq le(s(X),s(Y)) = le(X,Y) .
   eq app(nil,Z) = Z .
   eq app(cons(X,Z),W) = cons(X,app(Z,W)) .
   eq rm(X,nil) = nil .
   eq rm(X,cons(Y,Z)) = ifLNat(eqNat(X,Y),rm(X,Z),cons(Y,rm(X,Z))) .
   eq min(cons(X,nil)) = X .
   eq min(cons(X,cons(Y,Z))) = ifNat(le(Y,X),min(cons(Y,Z)),min(cons(X,Z))) .
   eq eqNat(0,0) = true2 .
   eq eqNat(s(X),0) = false2 .
   eq eqNat(0,s(X)) = false2 .
   eq eqNat(s(X),s(Y)) = eqNat(X,Y) .
   eq ifLNat(true2,Z,W) = Z .
   eq ifLNat(false2,Z,W) = W .
   eq ifNat(true2,X,Y) = X .
   eq ifNat(false2,X,Y) = Y .
   eq minsort(nil,nil) = nil .
   eq minsort(cons(X,Z),W) = ifLNat(eqNat(X,min(cons(X,Z))),
                                    cons(X,minsort(app(rm(X,Z),W),nil)),
                                    minsort(Z,cons(X,W))) .
   eq from(X) = cons(X,from(s(X))) .
   eq take(0,W) = nil .
   eq take(s(X),cons(Y,Z)) = cons(Y,take(X,Z)) .
   eq nfLNat(nil) = true2 .
   eq nfLNat(cons(X,Z)) = and(nfNat(X),nfLNat(Z)) .
   eq nfNat(0) = true2 .
   eq nfNat(s(X)) = nfNat(X) .
   eq and(true2,A) = A .
   eq and(false2,A) = false2 .
endo
```

B.7 Program `mod`

This program is borrowed from Example 3.5 of [AG01]. Auxiliary functions for natural numbers are included, namely `fact`, `times`, and `plus`. The call considered for evaluation is: `mod(fact(fact(3)),2)`

```
obj MOD is
   sorts Nat Bool2 .
   op 0 :  -> Nat .
   op s : Nat -> Nat               [strat (1)] .
   op true2 : -> Bool2 .
   op false2 : -> Bool2 .
   op minus : Nat Nat -> Nat       [strat (1 -2 0)] .
   op mod : Nat Nat -> Nat         [strat (1 -2 0)] .
   op le : Nat Nat -> Bool2        [strat (1 -2 0)] .
   op ifNat : Bool2 Nat Nat -> Nat [strat (1 0)] .
```

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

B.8 Program `mod'`

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

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

```
   eq mod(s(N),0) = 0 .
   eq mod(s(N),s(M)) = ifNat(le(M,N),mod(minus(N,M),s(M)),s(N)) .
   eq ifNat(true2,N,M) = N .
   eq ifNat(false2,N,M) = M .
   eq plus(0,N) = N .
   eq plus(s(M),N) = s(plus(M,N)) .
   eq times(0,N) = 0 .
   eq times(s(M),N) = plus(N,times(M,N)) .
   eq fact(0) = s(0) .
   eq fact(s(N)) = times(s(N),fact(N)) .
endo
```

B.9 Program `average`

This program is borrowed from Example 3.15 of [AG01]. Auxiliary functions for natural numbers are included, namely `fact`, `times`, and `plus`. The call considered for evaluation is: `average(square(square(4)),square(square(4)))`

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

C Proof of termination of `pi` program

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

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

Following [Luc01a], in order to prove termination of `PI4tr` (which only contains positive annotations), we can use the techniques for proving termination of context-sensitive rewriting (see [Luc02c] for a survey of these techniques). The application of Zantema's transformation ([Zan97]) to remove positive annotations, yields the following TRS (in a generic syntax not adhering to the OBJ syntax):

```
from(X) → cons(X,n__from(s(X)))
2ndspos(0,Z) → rnil
2ndspos(s(N),cons(X,Z)) → 2ndspos(s(N),cons2(X,activate(Z)))
2ndspos(s(N),cons2(X,cons(Y,Z))) → rcons(posrecip(Y),2ndsneg(N,activate(Z)))
2ndsneg(0,Z) → rnil
```

```
2ndsneg(s(N),cons(X,Z)) → 2ndsneg(s(N),cons2(X,activate(Z)))
2ndsneg(s(N),cons2(X,cons(Y,Z))) → rcons(negrecip(Y),2ndspos(N,activate(Z)))
pi(X) → 2ndspos(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 [EC96] (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;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](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;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](X0,X1) = 0;
[cons](X0,X1) = 0;
['times'](X0,X1) = X0;

checking component 3 (disjunction of 2 constraints)
[rnil] = 0;
[0] = 0;
[activate](X0) = X0;
[n__from](X0) = 0;
[square](X0) = 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;
[2ndsneg](X0,X1) = 0;
[rcons](X0,X1) = 0;
[cons2](X0,X1) = 0;
[2ndspos](X0,X1) = 0;
[cons](X0,X1) = 0;
['2ndsneg'](X0,X1) = X0;
['2ndspos'](X0,X1) = X0;

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