On-Demand Strategy Annotations Revisited: An Improved On-Demand Evaluation Strategy[*]

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 "evaluation on-demand". These on-demand indices have been proposed to support laziness in OBJ-like languages. While strategy annotations containing only natural numbers have been implemented and investigated to some extent (regarding, e.g., termination, confluence, and completeness), fully general annotations (including positive and negative indices) 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. Then, we propose a solution to these problems by keeping an accurate track of annotations along the evaluation sequences. We formalize this solution as a suitable extension of the evaluation strategy of OBJ-like languages (which only consider annotations given as natural numbers) to on-demand strategy annotations. Our *on-demand evaluation strategy* (*ODE*) overcomes the drawbacks of previous proposals and also has better computational properties. For instance, we show how to use this strategy for computing (head-)normal forms. We also introduce a transformation which allows us to prove termination of the new evaluation strategy by using standard rewriting techniques. Finally, we present two interpreters of the new strategy together with some encouraging experiments which demonstrate the usefulness of our approach.

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

[*]This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2007-68093-C02-02, and Generalitat Valenciana GVPRE/2008/113.

[†]DSIC, TU Valencia, Spain. Emails: {alpuente,sescobar,slucas}@dsic.upv.es

[‡]Fakultät für Informatik, TU Wien, Austria. Email: gramlich@logic.at

1 Introduction

Eager rewriting-based programming languages such as Lisp, OBJ*, CafeOBJ, ELAN, or Maude evaluate expressions by innermost rewriting. Since nontermination is a frequent problem of innermost reduction, *syntactic annotations* have been used in OBJ-like programming languages—OBJ2 [FGJM85], OBJ3 [GWM$^+$00], CafeOBJ [FN97], and Maude [CDE$^+$07]—to (hopefully) avoid nontermination. These annotations can also improve the efficiency of computations (e.g., by reducing the number of attempted matchings or avoiding useless or duplicated reductions) [Eke00]. Syntactic annotations have been generally specified as sequences of integers associated to function symbols called *local strategies*. Local strategies are used in OBJ programs 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 order), and when 0 is encountered, then the evaluation at the root position is attempted.

Example 1 *Consider the following* Maude *(functional*[1]*) modules* `NAT` *and* `LIST-NAT` *defining sorts* `Nat` *and* `LNat` *with symbols* `0` *and* `s` *for expressing natural numbers, and symbols* `nil` *(the empty list) and* `_._` *for the construction of lists. Note that* Maude *gives a default strategy* $(1\ 2\ \cdots\ ar(f)\ 0)$ *to each symbol* f*, which is not given an explicit strategy. The reserved word* `protecting` *can be understood as* module reuse. *In this paper, infix symbols such as* `_._`*, are right-associative, and thus written without extra parenthesis.*

```
fmod NAT is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  ops _+_ _-_ _*_ : Nat Nat -> Nat .
  op _^2 : Nat -> Nat .
  vars X Y : Nat .
  eq 0 + Y = Y .                eq 0 - Y = 0 .
  eq s(X) + Y = s(X + Y) .      eq s(X) - 0 = s(X) .
  eq 0 * Y = 0 .                eq s(X) - s(Y) = X - Y .
  eq s(X) * Y = Y + (X * Y) .   eq X ^2 = X * X .
endfm

fmod LIST-NAT is
  protecting NAT .
  sort LNat .
  op nil : -> LNat .
  op _._ : Nat LNat -> LNat [strat (1 0)] .
```

[1]In Maude, local strategies are only considered for *functional* modules, where termination and confluence are assumed. In this paper, we are only interested in termination of the evaluation relation, since we adopt a non-deterministic evaluation strategy and confluence is not necessary.

```
    op from : Nat -> LNat .
    op take : Nat LNat -> LNat .
    vars X N : Nat .   var XS : LNat .
    eq from(X) = X . from(s(X)) .
    eq take(0, XS) = nil .
    eq take(s(N), X . XS) = X . take(N,XS) .
  endfm
```

The strategy `(1 0)` *for symbol* `_._` *guarantees that the resulting program is terminating*[2]. ■

Unfortunately the absence of some indices in the local strategies can jeopardize some computational properties, e.g. the ability to compute normal forms.

Example 2 *The evaluation of the term* `take(s(0),from(0))` *w.r.t. the program in Example 1 using* Maude[3] *yields the following:*

```
  Maude> red take(s(0),from(0)) .
  result Nat: 0 . take(0,from(s(0)))
```

Due to the absence of natural number 2 *in the strategy* `(1 0)` *for the symbol* `_._`*, the contraction of the redex* `take(0,from(s(0)))` *is not possible and the evaluation stops.* ■

The problems related to the correctness and completeness of computations when (only) positive annotations have been used are discussed in [AEL04, Luc01a, NO01, OF00]. A number of solutions have been proposed:

1. Perform a *layered normalization*: when the evaluation stops due to the replacement restrictions imposed by the strategy annotations, it is enabled again over certain inner parts of the resulting term until a normal form is reached (if there exists any) [Luc02a];

2. Transform the program so that values, i.e., constructor ground terms, can be computed [AEL04]; and

3. Extend strategy annotations with *negative* indices, which enable some extra, *on-demand* evaluation [NO01, OF00].

For the term "`0 . take(0,from(s(0)))`" of Example 2, solutions 1 and 2 accomplish the rewriting of the subterm `take(0,from(s(0)))` into `nil`, yielding the final term "`0 . nil`".

[2]The interested reader can prove the termination of this specification automatically, e.g. by using the tool MU-TERM available at `http://www.dsic.upv.es/~slucas/csr/termination/muterm`.

[3]We use the Maude interpreter version 2.1.1 [CDE$^+$07] available at `http://maude.cs.uiuc.edu`.

In this paper, we show how solution 3 above, based on negative (or on-demand) strategy annotations, can improve program properties such as completeness or termination as compared to using (only) positive strategy annotations. We formalize a well-defined computational model for such negative annotations that outperforms previous proposals [OF00, NO01, FKW00, Luc01a]. Note that such a formalization is a difficult error-prone task, as evidenced by the many mistakes and deficiencies of previous proposals. Before going into details, let us motivate in the following subsections how negative indices can improve Maude strategy annotations.

1.1 Motivation for on-demand evaluation

1.1.1 Using negative indices in strategy annotations

Let us consider a concrete example where negative (on-demand) annotations allow us to achieve correctness and completeness, whereas other techniques that only use positive annotations do not.

Example 3 *(Example 1 cont'd) The following modules* `FRAC`*,* `LIST-FRAC`*, and* `PI` *implement the well-known infinite series expansion to approximate $\pi/4$:*

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

```
fmod FRAC is
  protecting NAT .
  sort IntFrac .
  op  1/_ : Nat -> IntFrac .   op -1/_ : Nat -> IntFrac .
endfm

fmod LIST-FRAC is
  protecting FRAC .
  sort LIntFrac .
  op nil : -> LIntFrac .
  op _._ : IntFrac LIntFrac -> LIntFrac .
endfm

fmod PI is
  protecting LIST-NAT .
  protecting LIST-FRAC .
  op pi : Nat -> LIntFrac .
  ops seriesPos seriesNeg : Nat LNat -> LIntFrac .
  vars N X Y : Nat . var XS : LNat .
  eq seriesPos(0,XS) = nil .
  eq seriesPos(s(N),X . Y . XS) =  1/ Y . seriesNeg(N,XS) .
  eq seriesNeg(0,XS) = nil .
  eq seriesNeg(s(N),X . Y . XS) = -1/ Y . seriesPos(N,XS) .
  eq pi(N) = seriesPos(N,from(0)) .
endfm
```

The evaluation of the term `pi(s(s(0)))` *should yield the approximation* $1 - \frac{1}{3}$ *to* $\pi/4$*, denoted by the term "*`1/ s(0) . -1/ s(s(s(0))) . nil`*". However, we get:*

```
Maude> red pi(s(s(0))) .
result LIntFrac: seriesPos(s(s(0)), 0 . from(s(0)))
```

The problem is that the absence of index 2 *in the strategy for symbol* `_._` *of sort* `LNat` *in Example 1 disallows the evaluation of subterm "*`from(s(0))`*" in the term "*`seriesPos(s(s(0)), 0 . from(s(0)))`*", thus disabling the application of the second equation*[4] *of* `seriesPos`*. We can informally say that a rewriting step on subterm "*`from(s(0))`*" in "*`seriesPos(s(s(0)), 0 . from(s(0)))`*" is* demanded *by the above equation because the root symbol* `from` *differs from the root symbol* `_._` *at the same position in the left-hand side of the equation. This demand–driven evaluation mode, which would be specified by including the index* -2 *in the strategy* $(1\ -2\ 0)$ *for symbol* `_._`*, triggers the following rewrite step*[5]*:*

$$
\begin{array}{l}
\texttt{seriesPos(s(s(0)), 0 . }\underline{\texttt{from(s(0))}}\texttt{)} \\
\quad \to \texttt{seriesPos(s(s(0)), 0 . s(0) . from(s(s(0))))}
\end{array}
$$

Then, the second equation of `seriesPos` *can be applied, since no inner subterm is demanded by the equation (according to the previous informal definition):*

$$
\begin{array}{l}
\underline{\texttt{seriesPos(s(s(0)), 0 . s(0) . from(s(s(0))))}} \\
\quad \to \texttt{1/s(0) . seriesNeg(s(0), from(s(s(0))))}
\end{array}
$$

Note that solutions 1 or 2 mentioned above cannot solve this problem, since they can reduce inner non-reduced subterms but can never reduce the whole term at the top. Indeed, both solutions 1 and 2 will enter a loop when trying to reduce every new occurrence of the symbol `from` *(assuming the original strategy* $(1\ 0)$ *for symbol* `_._`*):*

$$
\begin{array}{l}
\texttt{seriesPos(s(s(0)), 0 . }\underline{\texttt{from(s(0))}}\texttt{)} \\
\ \to \texttt{seriesPos(s(s(0)), 0 . s(0) . }\underline{\texttt{from(s(s(0)))}}\texttt{)} \\
\ \to \texttt{seriesPos(s(s(0)), 0 . s(0) . s(s(0)) . }\underline{\texttt{from(s(s(s(0))))}}\texttt{)} \\
\ \to \cdots
\end{array}
$$

■

1.1.2 Problems with previous on-demand evaluation strategies

As a solution to the incompleteness problem under positive strategy annotations, the rather intuitive notion of on-demand evaluation has been investigated in this context (see [AL02] for a survey on demandness in the general context of programming languages). In [NO01, OF00], *negative* indices are proposed to indicate those arguments to be evaluated only 'on-demand'. However, the *on-demand E-strategy* of [NO01, OF00] entails a number of shortages that we discuss in the following example.

[4]We use the words rule and equation both in the sense of a rewrite rule.

[5]We underline the redex reduced at each evaluation step.

Example 4 *Consider the following program encoding the length function for lists which uses an auxiliary symbol* `length'` *that does not allow any reduction on its argument:*

```
fmod LIST-NAT-LENGTH is
  protecting LIST-NAT .
  op length  : LNat -> Nat    [strat (-1 0)] .
  op length' : LNat -> Nat    [strat (0)] .
  var X : Nat . var XS : LNat .
  eq length(XS) = length'(XS) .
  eq length'(nil) = 0 .
  eq length'(X . XS) = s(length'(XS)) .
endfm
```

The term `length(from(0))` *is rewritten (in one step) to the term* `length'(from(0))`*. No evaluation is demanded on the argument of* `length` *for enabling this step, since the equation for* `length` *does have a variable at its argument position, and no further evaluation on* `length'(from(0))` *should be performed due to the absence of indices* 1 *and* -1 *in the local strategy* `(0)` *of* `length'`*. However, the strategy* `(-1 0)` *of function* `length` *is treated in such a way by the models of [OF00, NO01] that the on-demand evaluation of the term* `length(from(0))` *yields an infinite sequence (whether*[6] *we use the operational model in [OF00] or whether we use [NO01]). For instance,* CafeOBJ[7] *ends with a stack overflow (using* CafeOBJ*'s syntax to represent the program):*

```
LIST-NAT-LENGTH> red length(from(0)) .
Error: Stack overflow (signal 1000)
```

This is because the negative annotations are implemented as marks on terms that are propagated through the evaluation and can (inappropriately) initiate reductions later on; see Example 11 below for further details. In our approach, we substitute such marks on symbols by a list of already processed annotations, local to each symbol. ■

Other proposals in the literature that use on-demand strategy annotations are *lazy rewriting* (*LR*) [FKW00] and on-demand rewriting (*ODR*) [Luc01a], though they are not expressed by using negative annotations. Actually, the inspiration for the (positive) local strategies of OBJ comes from lazy rewriting (*LR*), which uses a strategy to specify the eager evaluation of function arguments, whereas the default strategy is lazy (or on-demand). On-demand rewriting (*ODR*) is the natural extension of context-sensitive rewriting [Luc98] to deal with on-demand strategy annotations. We demonstrate that the on-demand evaluation strategy (*ODE*) introduced in this paper outperforms also these two approaches.

[6]Actually, the operational models in [OF00] and [NO01] differ and deliver different computations, see Example 12 below.

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

1.2 Plan of the paper

After some preliminaries in Section 2, in Section 3 we recall the previous proposals for dealing with *on-demand strategy annotations* in OBJ-like languages, namely the on-demand E-strategy [OF00, NO01], and discuss some drawbacks regarding the management of demandedness. These previous proposals will guide the definition in Section 4 of our on-demand evaluation strategy (*ODE*), which handles demandedness in the right way. We think it is useful to motivate our approach by discussing two previous proposals since the involved problems are quite subtle and have led to various erroneous decisions in the past. In Section 5, we prove some computational properties of *ODE* regarding its ability to compute head-normal forms and normal forms. In Section 6, we compare our reduction model to three representative previous approaches: (i) the on-demand E-strategy, (ii) lazy rewriting (*LR*), and (iii) on-demand rewriting (*ODR*). Since one of the main purposes of strategy annotations is to guarantee and prove termination, Section 7 investigates how to formally prove termination of programs that use our computational model for implementing negative strategy annotations. In order to prove the practicality of our ideas, we present in Section 8 two interpreters of the on-demand evaluation strategy (*ODE*) together with some encouraging experiments. Section 9 concludes and summarizes our contributions. The proofs of all technical results are included in Appendix A.

This paper is a substantially extended and improved version of [AEGL02]. In particular, the definition of the on-demand evaluation strategy (*ODE*) has been improved (it is more restrictive than the previous one while preserving the same properties), and Sections 6.1 and 6.3 that compare *ODE* with the on-demand E-strategy and with the on-demand rewriting (*ODR*), have been added. Moreover, in Section 8, two implementations of the on-demand evaluation strategy (*ODE*) are benchmarked and compared with different OBJ-family systems (dealing with negative annotations or not).

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 $\to \subseteq A \times A$ be an arbitrary binary relation on a set A. We denote the reflexive closure of $\to$ by $\to^=$, its transitive closure by $\to^+$ and its reflexive and transitive closure by $\to^*$. An element $a \in A$ is an $\to$-normal form, if there exists no b such that $a \to b$. We say that b is an $\to$-normal form of a (written $a \to^! b$), if b is an $\to$-normal form and $a \to^* b$. We say that $\to$ is *terminating* iff there is no infinite sequence $a_1 \to a_2 \to a_3 \cdots$. We say that $\to$ is *confluent* if, for every $a, b, c \in A$, whenever $a \to^* b$ and $a \to^* c$, there exists $d \in A$ such that $b \to^* d$ and $c \to^* d$.

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 $\mathcal{T}(\mathcal{F}, \mathcal{X})$. $\mathcal{V}ar(t)$ is the set of variables in t. A term is said to be *linear* if it has no multiple occurrences of a single variable. A k-tuple $t_1, \ldots, t_k$ of terms is written $\bar{t}$. The number k of elements of the tuple $\bar{t}$ will be clear from the context.

We will extensively use labeled terms in this paper: Given a signature $\mathcal{F}$ and a set of labels $\mathcal{L}$, $\mathcal{F} \times \mathcal{L}$ (or $\mathcal{F}_{\mathcal{L}}$) is a new signature of labeled symbols. The labeling of a symbol $f \in \mathcal{F}$ for a given $\lambda \in \mathcal{L}$ is denoted by, e.g., f^{λ} or f_{λ}, rather than $\langle f, \lambda \rangle$; the arity of f^{λ} or f_{λ} is the arity of f.

A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ which homomorphically extends to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. The substitution σ is usually different from the identity substitution id, i.e., $\forall x \in \mathcal{X} : id(x) = x$, for a finite subset $\mathcal{D}om(\sigma) \subseteq \mathcal{X}$, called the *domain* of σ.

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 all positions of a term t. The set of positions of all non-variable symbols in t is denoted by $\mathcal{P}os_{\mathcal{F}}(t)$, and $\mathcal{P}os_{\mathcal{X}}(t)$ is the set of all positions of variables in t. Given positions p, q, we denote their concatenation by $p.q$. Positions are ordered by the standard prefix order $\leq$. Given a set of positions P and a partial order $\leq$, $minimal_{\leq}(P)$ is the set of minimal positions of P w.r.t. order $\leq$. If $\leq$ is a total order, then $min_{\leq}(P)$ (resp. $max_{\leq}(P)$) is the smallest (resp. greatest) position of P w.r.t. order $\leq$. 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 *term rewriting system* (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* (LL) 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 $\mathcal{C} = \mathcal{F} - \mathcal{D}$. 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 $1 \leq i \leq k$.

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 $l \to r \in R$, $p \in \mathcal{P}os_{\mathcal{F}}(t)$, and substitution σ. A term is a *head-normal form* if it does not reduce (in finitely many steps) to a redex.

In this paper, we do not consider AC symbols (neither rewriting modulo any equational theory) nor TRSs that are not left-linear or are not constructor systems, though they are supported by many of the OBJ-like languages. Strategy annotations are explicitly prohibited for AC symbols (see [FR99, GM04]) and the completeness of evaluation with positive (and negative) strategy annotations is known to hold only for left-linear and constructor systems (see [Luc02a]).

3 Rewriting with strategy annotations

In this section, we recall the current proposals for dealing with positive and negative (on-demand) strategy annotations in OBJ-like languages. The acquaintance gained from the discussion will guide the definition of our computational model in Section 4 below.

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 and put together by juxtaposition. The empty list is denoted by nil. We sometimes write $i : L$ instead of $(i\ L)$ to denote a list composed of an integer i and the rest of the list L. We write $abs(i)$ for the absolute value of an integer i. Note that repeated indices are allowed in local strategies as well as indices with the same absolute value $(i\ {-i})$, e.g. $\varphi(f) = (0\ 1\ {-1}\ 1\ 0\ {-1})$. Let $\mathbb{L}^{\mathbb{Z}}$ be the set of all lists consisting of integers and $\mathbb{L}^{\mathbb{Z}}_n$ be the set of all lists of integers whose absolute values do not exceed $n \in \mathbb{N}$. Similarly, $\mathbb{L}^{\mathbb{N}}$ is the set of all lists consisting of naturals and $\mathbb{L}^{\mathbb{N}}_n$ its restriction to naturals that do not exceed $n \in \mathbb{N}$. We define an order $\sqsubseteq$ between sequences of integers as follows: $L \sqsubseteq L'$ if L is embedded into L'; formally: (i) $nil \sqsubseteq L$, for every L, (ii) $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_1\ j_2\ \cdots\ j_n)$ if $i_1 = j_1$ and $(i_2\ \cdots\ i_m) \sqsubseteq (j_2\ \cdots\ j_n)$, and (iii) $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_1\ j_2\ \cdots\ j_n)$ if $i_1 \neq j_1$ and $(i_1\ i_2\ \cdots\ i_m) \sqsubseteq (j_2\ \cdots\ j_n)$. For instance, $(1\ 4) \sqsubseteq (2\ 1\ 3\ 4)$ but $(4\ 1) \not\sqsubseteq (2\ 1\ 3\ 4)$.

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]. In this paper, we assume that the default strategy (i.e., when no explicit strategy is provided) given to a symbol f is $(1\ 2\ \cdots ar(f)\ 0)$. For simplification, the default strategy given to a constant c (i.e., $ar(c) = 0$) is nil if c is a constructor symbol, and (0) if c is a defined symbol. The extension of an order $\sqsubseteq$ to a strategy map 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, for all symbols $f \in \mathcal{F}$, $\varphi(f)$ is embedded into $\varphi'(f)$.

The semantics of rewriting under a given E-strategy 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

In [Nag99], Nagaya describes the mapping $eval^{\mathbb{N}}_\varphi$ 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. Given an E-strategy map φ for $\mathcal{F}$, we use the signature[8] $\mathcal{F}^{\mathbb{N}}_\varphi = \{f_L \mid f \in \mathcal{F}, L \in \mathbb{L}^{\mathbb{N}}_{ar(f)},$ 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

[8]Note that Nagaya (as well as Nakamura and Ogata) use $\mathcal{F}_{\mathbb{L}^{\mathbb{N}}}$ and $\mathcal{X}_{\mathbb{L}^{\mathbb{N}}}$ instead of $\mathcal{F}^{\mathbb{N}}_\varphi$ and $\mathcal{X}^{\mathbb{N}}_\varphi$ (or $\mathcal{F}^{\mathbb{Z}}_\varphi$ and $\mathcal{X}^{\mathbb{Z}}_\varphi$), i.e., they do not consider the restriction to $L \sqsubseteq \varphi(f)$ as we do. Using terms over $\mathcal{F}^{\mathbb{N}}_\varphi$ ($\mathcal{F}^{\mathbb{Z}}_\varphi$) does not cause loss of generality and 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.

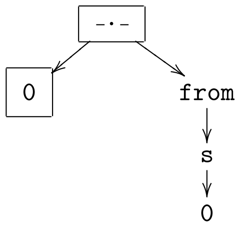


Figure 1: The positive part (full frame) of the term `0 . from(s(0))` according to the term with strategy annotations $\mathtt{0}_{nil}\ \cdot_{(1\ 0)}\ \mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1\ 0)}(\mathtt{0}_{nil}))$.

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 labels from symbols in the obvious way. We define the set of *positive* positions of a term $s \in \mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi}, \mathcal{X}^{\mathbb{N}}_{\varphi})$ as $\mathcal{P}os^{\mathbb{N}}_{P}(s) = \{\Lambda\} \cup \{i.\mathcal{P}os^{\mathbb{N}}_{P}(s|_i) \mid root(s) = f_L, i \in L, \text{ and } i > 0\}$.

Example 5 *For the* `OBJ` *program and the E-strategy map φ of Example 1, we have*

$$s = \varphi(\mathtt{0\ .\ from(0)}) = \mathtt{0}_{nil}\ \cdot_{(1\ 0)}\ \mathtt{from}_{(1\ 0)}(\mathtt{0}_{nil}).$$

The annotated term is depicted in Figure 1, where the symbols at positive positions $\mathcal{P}os^{\mathbb{N}}_{P}(s)$ are framed. ■

Definition 1 [Nag99, Definition 6.1.3] *Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and a positive E-strategy map φ for $\mathcal{F}$, $eval^{\mathbb{N}}_{\varphi} : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ is defined as $eval^{\mathbb{N}}_{\varphi}(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle\varphi(t), \Lambda\rangle \overset{\mathbb{N}}{\to}^{!}_{\varphi} \langle s, \Lambda\rangle\}$. The binary relation $\overset{\mathbb{N}}{\to}_{\varphi}$ on $\mathcal{T}(\mathcal{F}^{\mathbb{N}}_{\varphi}, \mathcal{X}^{\mathbb{N}}_{\varphi}) \times \mathbb{N}^{*}_{+}$ is defined as follows: $\langle t, p\rangle \overset{\mathbb{N}}{\to}_{\varphi} \langle s, q\rangle$ if and only if $p \in \mathcal{P}os^{\mathbb{N}}_{P}(t)$ and either*

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

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 (note that annotated

$$\begin{aligned}
\langle\ \texttt{from}_{(\boxed{1}\ 0)}(\texttt{0}_{nil}),\ \Lambda\ \rangle &\xrightarrow{\mathbb{N}}_{\varphi\ (2)} \langle\ \texttt{from}_{(0)}(\texttt{0}_{\boxed{nil}}),\ 1\ \rangle \\
&\xrightarrow{\mathbb{N}}_{\varphi\ (1)} \langle\ \underline{\texttt{from}_{(\boxed{0})}(\texttt{0}_{nil})},\ \Lambda\ \rangle \\
&\xrightarrow{\mathbb{N}}_{\varphi\ (4)} \langle\ \texttt{0}_{nil}\ \cdot_{(\boxed{1}\ 0)}\ \texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})),\ \Lambda\ \rangle \\
&\xrightarrow{\mathbb{N}}_{\varphi\ (2)} \langle\ \texttt{0}_{\boxed{nil}}\ \cdot_{(0)}\ \texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})),\ 1\ \rangle \\
&\xrightarrow{\mathbb{N}}_{\varphi\ (1)} \langle\ \texttt{0}_{nil}\ \cdot_{(\boxed{0})}\ \texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})),\ \Lambda\ \rangle \\
&\xrightarrow{\mathbb{N}}_{\varphi\ (3)} \langle\ \texttt{0}_{nil}\ \cdot_{nil}\ \texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})),\ \Lambda\ \rangle
\end{aligned}$$

Figure 2: Execution of term `from(0)` by Definition 1.

subterms are propagated through the matching substitution); otherwise, the 0 is removed from the list. In both cases, the evaluation continues at the same position p.

Example 6 *Consider the* OBJ *program of Example 1. Note that this program has only positive annotations. The evaluation of term* `from(0)` *produces the sequence shown in Figure 2 according to Definition 1 (for each step, we indicate which case of Definition 1 has been considered, frame the index involved, and underline the redex reduced, if any). The evaluation stops at term "*`0 . from(s(0))`*", since no further evaluation step can be performed. Recall that Figure 1 shows the term "*`0 . from(s(0))`*" with the symbols at positive (or reducible) positions framed.* ■

3.2 The on-demand E-strategy

Ogata and Futatsugi [OF00] proposed the use of negative integers in local strategies. Following Nagaya's style of description, Nakamura and Ogata [NO01] have formalized the corresponding evaluation mapping $eval_{\varphi}^{\mathbb{Z}}$ by using a reduction relation. We recall 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 $\mathcal{F}_{\varphi}^{\mathbb{Z}} = \{f_L^b \mid f \in \mathcal{F}, L \in \mathbb{L}_{ar(f)}^{\mathbb{Z}}, L \sqsubseteq \varphi(f), \text{and } b \in \{0,1\}\}$ and labeled variables $\mathcal{X}_{\varphi}^{\mathbb{Z}} = \{x_{nil}^0 \mid x \in \mathcal{X}\}$. 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 $\mathcal{T}(\mathcal{F}_{\varphi}^{\mathbb{Z}},\mathcal{X}_{\varphi}^{\mathbb{Z}})$ as follows:

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

Given a term $s \in \mathcal{T}(\mathcal{F}_{\varphi}^{\mathbb{Z}},\mathcal{X}_{\varphi}^{\mathbb{Z}})$, we define the set of its *positive* positions as $\mathcal{P}os_P^{\mathbb{Z}}(s) = \{\Lambda\} \cup \{i.\mathcal{P}os_P^{\mathbb{Z}}(s|_i) \mid root(s) = f_L^b, i \in L, \text{ and } i > 0\}$, the set of its *active* positions as $\mathcal{P}os_A^{\mathbb{Z}}(s) = \{\Lambda\} \cup \{abs(i).\mathcal{P}os_A^{\mathbb{Z}}(s|_i) \mid root(s) = f_L^b$ and either $b =$

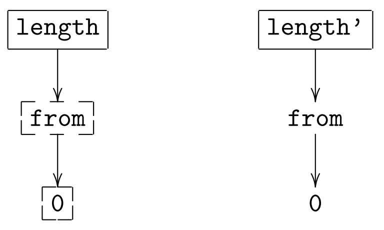


Figure 3: The positive and negative positions (full and dashed frame, respectively) of terms $\texttt{length}^0_{(-1\ 0)}(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil}))$ and $\texttt{length'}^0_{(0)}(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil}))$.

$1, i \in \{1, \ldots, ar(f)\}$ or $i \in L, i \neq 0\}$, and the set of its *on-demand* (or *negative*) positions as $\mathcal{P}os^{\mathbb{Z}}_N(s) = \mathcal{P}os^{\mathbb{Z}}_A(s) - \mathcal{P}os^{\mathbb{Z}}_P(s)$.

Example 7 *For the* OBJ *program and the map* φ *of Example 4, we have*

$$s_1 = \varphi(\texttt{length(from(0))}) = \texttt{length}^0_{(-1\ 0)}(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil})),$$

and

$$s_2 = \varphi(\texttt{length'(from(0))}) = \texttt{length'}^0_{(0)}(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil})).$$

Figure 3 shows them with the symbols at positive positions $\mathcal{P}os^{\mathbb{Z}}_P(s_1)$ *and* $\mathcal{P}os^{\mathbb{Z}}_P(s_2)$, *and on-demand positions* $\mathcal{P}os^{\mathbb{Z}}_N(s_1)$ *and* $\mathcal{P}os^{\mathbb{Z}}_N(s_2)$ *framed with a full or dashed line, respectively.* ■

The mapping *erase* : $\mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ removes labels from symbols in the obvious way. The (partial) function $flag : \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi) \times \mathbb{N}^*_+ \to \{0,1\}$ returns the flag of the function symbol at a position of the term: $flag(t,p) = b$ if $root(t|_p) = f^b_L$. The map $up : \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi) \to \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi)$ (resp. $dn : \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi) \to \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi)$) 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. $up(x^0_{nil}) = dn(x^0_{nil}) = x^0_{nil}$, $up(f^b_L(t_1, \ldots, t_k)) = f^1_L(up(t_1), \ldots, up(t_k))$, and $dn(f^b_L(t_1, \ldots, t_k)) = f^0_L(dn(t_1), \ldots, dn(t_k))$.

Example 8 *For the* OBJ *program and the map* φ *of Example 4, we have*

$$\texttt{length}^0_{(-1\ 0)}(up(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil}))) = \texttt{length}^0_{(-1\ 0)}(\texttt{from}^1_{(1\ 0)}(\texttt{0}^1_{nil}))$$

and

$$up(\texttt{length}^0_{(-1\ 0)}(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil}))) = \texttt{length}^1_{(-1\ 0)}(\texttt{from}^1_{(1\ 0)}(\texttt{0}^1_{nil})).$$

■

In [NO01], the matching of a term t with the left-hand side l of a rule is attempted following the top-down and left-to-right order. 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. Note that variables in both terms t and l are treated as constants by the operator $\mathcal{P}os_{\neq}$, and thus no substitution is actually computed. Therefore it may happen that $\mathcal{P}os_{\neq}(t,l) = \varnothing$ but t is not an instance of l due to some repeated variable in l. Then, the map $df_l : \mathcal{T}(\mathcal{F},\mathcal{X}) \to \mathbb{N}_+^* \cup \{\top\}$ returns the first position where the term t and the lhs l 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 order 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 $i < j$.

Example 9 *For the* OBJ *program and the map φ of Example 4, we have $l_1 =$* `length(XS)`, *$l_2 =$* `length'(nil)` *and $l_3 =$* `length'(X . XS)` *with*

$$df_{l_1}(\texttt{length(from(0))}) = \top \quad df_{l_2}(\texttt{length(from(0))}) = \Lambda$$
$$df_{l_3}(\texttt{length(from(0))}) = \Lambda$$

and

$$df_{l_1}(\texttt{length'(from(0))}) = \Lambda \quad df_{l_2}(\texttt{length'(from(0))}) = 1$$
$$df_{l_3}(\texttt{length'(from(0))}) = 1$$

■

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:

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

Example 10 *(Example 9 cont'd) We have $DF_{\mathcal{R}}(\texttt{length(from(0))}) = \top$, since $df_{l_1}(\texttt{length(from(0))}) = \top$, and $DF_{\mathcal{R}}(\texttt{length'(from(0))}) = max_{\leq_{lex}}(\{\Lambda, 1\}) =$ 1.* ■

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

1. *$root(t|_p) = f_{nil}^b$, $s = t$, and $p = q.i$ for some i;*
2. *$t|_p = f_{i:L}^b(t_1,\ldots,t_k)$, $i > 0$, $s = t[f_L^b(t_1,\ldots,t_k)]_p$, and $q = p.i$;*
3. *$t|_p = f_{-i:L}^b(t_1,\ldots,t_k)$, $i > 0$, $s = t[f_L^b(t_1,\ldots,up(t_i),\ldots,t_k)]_p$, and $q = p$;*

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 (i)* $DF_{\mathcal{R}}(erase(t|_p)) = \top$ *and* $erase(t|_p)$ *is not a redex, (ii)* $DF_{\mathcal{R}}(erase(t|_p)) = \Lambda$, *or (iii)* $DF_{\mathcal{R}}(erase(t|_p)) = p' \neq \Lambda$ *and* $flag(t, p.p') = 0$;

   (c) $t' = f^b_L(t_1, \ldots, t_i[up(s)]_{p''}, \ldots, t_k)$ *if* $DF_{\mathcal{R}}(erase(t|_p)) = p' = i.p''$, $flag(t, p.p') = 1$, $\langle dn(t|_{p.p'}), \Lambda\rangle \overset{\mathbb{Z}}{\to}^!_{\varphi} \langle s, \Lambda\rangle$, *and* $DF_{\mathcal{R}}(erase(t|_p[s]_{p'})) = p'$;

   (d) $t' = t|_p[up(s)]_{p'}$ *if* $DF_{\mathcal{R}}(erase(t|_p)) = p' \neq \Lambda$, $flag(t, p.p') = 1$, $\langle dn(t|_{p.p'}), \Lambda\rangle \overset{\mathbb{Z}}{\to}^!_{\varphi} \langle s, \Lambda\rangle$, *and either* $p' <_{lex} DF_{\mathcal{R}}(erase(t|_p[s]_{p'}))$ *or* $DF_{\mathcal{R}}(erase(t|_p[s]_{p'})) = \top$. ■

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 that 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 evaluation steps on demanded positions. 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 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, then the evaluation of $t|_p$ continues according to the computed value. In case $4(c)$, after the evaluation of subterm $t|_{p.p'}$, position $p.p'$ is still demanded, which implies that more evaluation is necessary but unfeasible, and thus the index 0 is removed. In case $4(d)$, after the evaluation of subterm $t|_{p.p'}$, position $p.p'$ is no longer demanded, and thus more evaluation is needed at another (possibly demanded) position.

3.2.1 Unexpected behavior of the on-demand E-strategy

In the following example, we illustrate in detail why the notion of demandedness of [NO01], given in Definition 2, does not work for Example 4.

Example 11 *(Example 4 cont'd) The on-demand evaluation of term* `length(from(0))` *following Definition 2 is shown in Figure 4 (at each step, we indicate which case of Definition 2 has been considered, frame the indices involved in such steps, and underline the selected redex, if any). In the first reduction step, annotation* -1 *of symbol* `length` *is consumed according to case*

3 of Definition 2 and, thus, the subterm $\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil})$ *is marked with the on-demand (superscript) flag, i.e.,* $up(\texttt{from}^0_{(1\ 0)}(\texttt{0}^0_{nil})) = \texttt{from}^1_{(1\ 0)}(\texttt{0}^1_{nil})$*. Annotation* `0` *of* `length` *is considered 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 processed and the whole term* `length'(from(0))` *is not a redex, thus an on-demand position is sought. Here, the function* $DF_{\mathcal{R}}$*, which calculates demanded positions, returns position* 1 *as shown in Example 10. Since this position is marked with the on-demand flag, case* $4(c)$ *or* $4(d)$ *is applied and the evaluation of term* `from(0)` *at position* 1 *is initiated (using a different rewriting subsequence). The term* `from(0)` *is evaluated into "*`0 . from(s(0))`*" and the term "*`length'(0 . from(s(0)))`*" is re-evaluated. The cycle of demanding the evaluation of the first argument of* `length'` *is repeated again and again.*

The point is that, within the labeled term $\texttt{length'}^0_{(0)}(\texttt{from}^1_{(1\ 0)}(\texttt{0}^1_{nil}))$*, the strategy does not recognize that the (activated) on-demand flag on symbol* `from` *does* not *stem 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, the 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).* ■

The problem is that the on-demand E-strategy does not keep track of the origin of the on-demand flags, i.e., the strategy does not recognize whether an on-demand flag stems from the annotations of symbols. This could be fixed either by resetting the on-demand flags w.r.t. the current annotations of symbols after each rewriting step or by keeping track of the origin of on-demand flags. In this paper, we solve the problem by keeping a sort of memory of each processed index, i.e., by keeping track of the origin of on-demand marks.

3.2.2 Differences between the two models of the on-demand E–strategy

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 (according to the lexicographic order on positions) of all positions demanded on t by each rule of the TRS. However, in the selection of demanded positions of Ogata and Futatsugi's definition, the order of the rules in the program is extremely important.

Example 12 *Consider the following module with the function greater than or equal to between natural numbers, denoted by* `geq`*, and the (auxiliary) non-terminating*[9] *function* `foo`*:*

```
    fmod NAT-FOO is
```

[9]Note that the function `foo` may be replaced by a terminating but computationally expensive expression and, thus, we use a non-terminating function only for motivational purposes.

$$
\begin{array}{l}
\langle\ \texttt{length}^{0}_{(\boxed{\text{-1}}\ 0)}(\texttt{from}^{0}_{(1\ 0)}(\texttt{0}^{0}_{nil})),\ \Lambda\ \rangle \\
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (3)} \langle\ \underline{\texttt{length}^{0}_{(\boxed{0})}(\texttt{from}^{1}_{(1\ 0)}(\texttt{0}^{1}_{nil})),}\ \Lambda\ \rangle \\
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (4a)} \langle\ \texttt{length'}^{0}_{(\boxed{0})}(\texttt{from}^{1}_{(1\ 0)}(\texttt{0}^{1}_{nil})),\ \Lambda\ \rangle
\end{array}
$$

$$
\begin{array}{|l}
\langle\ \texttt{from}^{0}_{(\boxed{1}\ 0)}(\texttt{0}^{0}_{nil}),\ \Lambda\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (2)} \langle\ \texttt{from}^{0}_{(0)}(\texttt{0}^{0}_{\boxed{nil}}),\ 1\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (1)} \langle\ \underline{\texttt{from}^{0}_{(\boxed{0})}(\texttt{0}^{0}_{nil}),}\ \Lambda\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (4a)} \langle\ \texttt{0}^{0}_{nil}\ \cdot^{0}_{(\boxed{1}\ 0)}\ \texttt{from}^{0}_{(1\ 0)}(\texttt{s}^{0}_{(1\ 0)}(\texttt{0}^{0}_{nil})),\ \Lambda\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (2)} \langle\ \texttt{0}^{0}_{\boxed{nil}}\ \cdot^{0}_{(0)}\ \texttt{from}^{0}_{(1\ 0)}(\texttt{s}^{0}_{(1\ 0)}(\texttt{0}^{0}_{nil})),\ 1\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (1)} \langle\ \texttt{0}^{0}_{nil}\ \cdot^{0}_{(\boxed{0})}\ \texttt{from}^{0}_{(1\ 0)}(\texttt{s}^{0}_{(1\ 0)}(\texttt{0}^{0}_{nil})),\ \Lambda\ \rangle \\
\quad\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (4b)} \langle\ \texttt{0}^{0}_{nil}\ \cdot^{0}_{nil}\ \texttt{from}^{0}_{(1\ 0)}(\texttt{s}^{0}_{(1\ 0)}(\texttt{0}^{0}_{nil})),\ \Lambda\ \rangle
\end{array}
$$

$$
\begin{array}{l}
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (4d)} \langle\ \underline{\texttt{length'}^{0}_{(\boxed{0})}(\texttt{0}^{1}_{nil}\ \cdot^{1}_{nil}\ \texttt{from}^{1}_{(1\ 0)}(\texttt{s}^{1}_{(1\ 0)}(\texttt{0}^{1}_{nil}))),}\ \Lambda\ \rangle \\
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (4a)} \langle\ \texttt{s}^{0}_{(\boxed{1}\ 0)}(\texttt{length'}^{0}_{(0)}(\texttt{from}^{1}_{(1\ 0)}(\texttt{s}^{1}_{(1\ 0)}(\texttt{0}^{1}_{nil})))),\ \Lambda\ \rangle \\
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi\ (2)} \langle\ \texttt{s}^{0}_{(0)}(\texttt{length'}^{0}_{(\boxed{0})}(\texttt{from}^{1}_{(1\ 0)}(\texttt{s}^{1}_{(1\ 0)}(\texttt{0}^{1}_{nil})))),\ 1\ \rangle \\
\overset{\mathbb{Z}}{\longrightarrow}_{\varphi}\ \cdots
\end{array}
$$

Figure 4: Execution of term `length(from(0))` by Definition 2.

```
  protecting NAT .
  op foo : -> Nat .
  eq foo = foo .
endfm
fmod NAT-GEQ is
  protecting NAT .
  protecting BOOL .
  protecting NAT-FOO .
  vars X Y : Nat .
  op geq : Nat Nat -> Bool  [strat (-2 -1 0)] .
  eq geq(s(X),s(Y)) = geq(X,Y) .
  eq geq(X,0) = true .
endfm
```

Consider the term `geq(foo,0 + 0)`*. According to Ogata and Futatsugi's definition of the on-demand E-strategy, an infinite reduction sequence is started, since position* 1 *is the leftmost demanded position in the first rule of the program, and thus it is selected, which triggers a non-terminating evaluation sequence. For instance,* CafeOBJ *ends with a stack overflow:*

```
    NAT-GEQ> red geq(foo,0 + 0) .
    Error: Stack overflow (signal 1000)
```

However, Nakamura and Ogata's definition of on-demand E-strategy (Definition 2) selects position 2 as demanded (according to the inverse of the lexicographic order) and, after the evaluation, the second rule is applied, thus obtaining `true` *(see case 4(d) in Definition 2). Note that exactly the inverse behavior can be obtained by adding equation*[10] *"*`eq geq(0,X) = true .`*" before the other two and using term* `geq(0 + 0,foo)`*, i.e., Nakamura and Ogata's definition does not terminate whereas Ogata and Futatsugi's definition terminates. Thus, both definitions produce different E-normal forms.* ■

In this paper, we follow the strategies fixed by the user instead, and thus proceed according to the strategy $(-2\ -1\ 0)$ for `geq` that determines that the second argument must be selected when both the first and the second arguments are demanded, thus the evaluation of term `geq(foo, 0 + 0)` terminates, whereas it does not terminate for term `geq(0 + 0,foo)`, which is the intended behavior associated to the strategy $(-2\ -1\ 0)$. See Remark 1 below for further details about the impact of the order of evaluation.

3.2.3 Inconsistency of the on-demand E-strategy

The computational description of on-demand strategy annotations in Definition 2 involves recursive steps, as shown in Figure 4. A single reduction step on a (labeled) term t may involve the application of more than one reduction step on subterms of t (as it is defined in steps $4(c)$ and $4(d)$ of Definition 2). In fact, the definition of a single rewriting step depends on testing whether an expression is irreducible, which is just negation of reducibility, as shown in the following example.

Example 13 *Consider the following program:*

```
    fmod INCONSISTENCY is
      sort S .
      ops a g : -> S .
      op f : S -> S [strat (-1 0)] .
      eq f(a) = a .
      eq g = f(g) .
    endfm
```

Let us consider the term $t =$ `f(g)` *in its annotated form $\varphi(t) = \mathtt{f}^0_{(-1\ 0)}(\mathtt{g}^0_{(0)})$. According to Definition 2, we have the following rewriting sequence where index -1 of the strategy list of* `f` *activates superscript 1 in symbol* `g`*:*

$$\langle\ \mathtt{f}^0_{(-1\ 0)}(\mathtt{g}^0_{(0)}),\ \Lambda\ \rangle \overset{\mathbb{Z}}{\to}_{\varphi} \langle\ \mathtt{f}^0_{(0)}(\mathtt{g}^1_{(0)}),\ \Lambda\ \rangle$$

[10]This equation does not correspond to the *greater than or equal to* function but we use it only for motivational purposes.

There is no more single (or simple) step for the resulting term $s = \mathtt{f}^0_{(0)}(\mathtt{g}^1_{(0)})$ *and only cases* $4(c)$ *or* $4(d)$ *of Definition 2 could be applied to* s*. However, the application of cases* $4(c)$ *or* $4(d)$ *implies testing whether* $\mathtt{g}^0_{(0)}$ *has a normal form w.r.t.* $\overset{\mathbb{Z}}{\to}_\varphi$ *or not, and this raises a contradiction:*

1. *If* $\mathtt{g}^0_{(0)}$ *had a normal form, then it would be of the form* $\mathtt{f}^0_{(0)}(\cdots \mathtt{f}^0_{(0)}((\mathtt{g}^1_{(0)})\cdots)$*. But by Definition 2, there is a possible step reducing* $\mathtt{g}^1_{(0)}$ *in such normal form. Thus, we have a contradiction.*

2. *If* $\mathtt{g}^0_{(0)}$ *did not have a normal form, then* $\mathtt{f}^0_{(-1\ 0)}((\mathtt{g}^0_{(0)})$ *would not have one either, since* $\langle\ \mathtt{g}^0_{(0)},\ \Lambda\ \rangle \overset{\mathbb{Z}}{\to}_\varphi \langle\ \mathtt{f}^0_{(-1\ 0)}(\mathtt{g}^0_{(0)}),\ \Lambda\ \rangle$*. But if* $\mathtt{f}^0_{(-1\ 0)}(\mathtt{g}^0_{(0)})$ *does not have a normal form, then there must exist a next rewriting step and this step must reduce* $\mathtt{g}^0_{(0)}$*. However, there is no next step for* $\langle\ \mathtt{f}^0_{(-1\ 0)}(\mathtt{g}^0_{(0)}), \Lambda\ \rangle$ *according to Definition 2 (because* $\mathtt{g}^0_{(0)}$ *does not have a normal form by assumption) and then we have a contradiction.*

■

The problem is that the strategy is defined recursively and it makes use of "negative information" associated to its own definition. In this paper, we do not give a recursive definition of the evaluation strategy but put some sort of "negative information" explicitly as a mark on top of symbols.

In the following, we provide a correct and practical framework for implementing and studying OBJ computations, which may integrate the most interesting features of modern evaluation strategies with on-demand syntactic annotations.

4 Improving rewriting under on-demand strategy annotations: the on-demand evaluation strategy (*ODE*)

The drawbacks of existing operational models for arbitrary strategy annotations discussed so far can be summarized as follows:

1. the one-step reduction relation is, in general, contradictory (see Example 13);

2. the mechanization of demandedness by using negative annotations (via the marking of terms with flag 0 or flag 1) enables evaluation steps that should not be allowed (see Example 11), since

3. it does not properly keep track of the origin of the marks (lack of memory, see Example 11); and

4. the order between on-demand annotations fixed by the user is not used in the selection of on-demand positions (see Example 12).

Here, we want to discuss an extra drawback that further motivates our improved definition. Let us illustrate it by means of an example.

Example 14 *Consider the following* OBJ *program defining the function lower-than or equal-to between natural numbers, denoted by* `lt`*, and the non-terminating function* `foo` *defined in Example 12:*

```
fmod NAT-LT is
  protecting NAT .
  protecting BOOL .
  protecting NAT-FOO .
  op lt : Nat Nat -> Bool    [strat (-2 -1 0)] .
  vars X Y : Nat .
  eq lt(0,s(Y)) = true .
  eq lt(s(X),s(Y)) = lt(X,Y) .
endfm
```

Consider the term $t =$ `lt(foo,0)`*, which is a* head-normal form*, since no possible evaluation could enable the term 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:*

```
    NAT-LT> red lt(foo,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. That is,* `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.

4.1 Labeling terms

Two important points in our formulation are the use of two lists of annotations for each symbol (instead of only one as in the on-demand E-strategy of Definition 2) and a special flag for avoiding recursive definitions of the strategy.

The function $\oplus$ defines the concatenation of two sequences of numbers. Given a E-strategy map φ, we use the signature

$$\mathcal{F}^\sharp_\varphi = \cup\{f_{L_1|L_2}, \overline{f}_{L_1|L_2} \mid f \in \mathcal{F} \text{ and } L_1, L_2 \in \mathbb{L}^{\mathbb{Z}}_{ar(f)} \text{ s.t. } (L_1 \oplus L_2 \sqsubseteq \varphi(f))\}$$

and labeled variables $\mathcal{X}^\sharp_\varphi = \{x_{nil|nil} \mid x \in \mathcal{X}\}$ for marking ordinary terms $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ as terms $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. 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; indeed, we will call it the *memory list*. We use $f^\sharp$ to denote f or $\overline{f}$ for a given symbol $f \in \mathcal{F}$. 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 labels from terms.

Example 15 *For the* OBJ *program and the map* φ *of Example 4, we have*

$$\varphi(\texttt{length(from(0))}) = \texttt{length}_{nil|(-1\ 0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})).$$

■

4.2 On-demand matching

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 we exploit the standard distinction between constructor and defined symbols of a signature by restricting the attention only to disagreeing positions that correspond to defined symbols (by using $\mathcal{P}os_{\mathcal{D}}(t)$). Note that the use of $minimal_\leq$ in $DP_l(t)$ only considers the topmost different positions between a term t and a lhs l, not even the demanded positions, and is not related to the use of $min_{<_{lex}}$ in Section 3.2, which selects only one position among all the demanded positions.

Example 16 *(Example 14 cont'd) Consider the lhs's* $l_1 = \texttt{lt(0,s(Y))}$ *and* $l_2 = \texttt{lt(s(X),s(Y))}$*. For the term*

$$t_1 = \texttt{lt(foo,0)},$$

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*. For the term*

$$t_2 = \texttt{lt(foo,0 + 0)},$$

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, for*

$$t_3 = \texttt{lt(0,foo)},$$

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*.* ■

The following notion is auxiliary. We define the list of *lookout* indices of a labeled symbol $f^\sharp_{L_1|L_2}$ as

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

Intuitively, the lookout indices of a symbol f are those already processed, or those in the sequence $\varphi(f)$, if no annotation has been processed yet. We define the set of *positive* positions of a term $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ as

$$\mathcal{P}os_P(s) = \{\Lambda\} \cup \{i.\mathcal{P}os_P(s|_i) \mid i > 0 \text{ and } \mathit{lookout}(\mathit{root}(s)) \text{ contains } i\},$$

the set of *active* positions as

$$\mathcal{P}os_A(s) = \{\Lambda\} \cup \{i.\mathcal{P}os_A(s|_i) \mid i > 0 \text{ and } \mathit{lookout}(\mathit{root}(s)) \text{ contains } i \text{ or } -i\},$$

and the set of *on-demand* (or *negative*) positions as $\mathcal{P}os_N(s) = \mathcal{P}os_A(s) - \mathcal{P}os_P(s)$. Note that $\mathcal{P}os_P(s) \subseteq \mathcal{P}os_A(s)$ for all $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$; moreover, $\mathcal{P}os_P(s) = \mathcal{P}os_A(s)$ if the labels in s contain no negative number. By abuse, symbols rooting subterms at positive and on-demand positions are called positive and on-demand symbols, respectively. We also define the set of positions with *empty* annotation list as

$$\mathcal{P}os_{nil}(s) = \{p \in \mathcal{P}os(s) \mid \mathit{root}(s|_p) = f_{L|nil}\}.$$

Example 17 *For the annotated term*

$$t_1 = \texttt{length'}_{nil|(1\ 0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),$$

we have $\mathcal{P}os_P(t_1) = \mathcal{P}os_A(t_1) = \{\Lambda, 1, 1.1\}$*, and* $\mathcal{P}os_{nil}(t_1) = \{1.1\}$*. For the annotated term*

$$t_2 = \texttt{length'}_{nil|(-1\ 0)}(\texttt{from}_{nil|nil}(\texttt{0}_{nil|nil})),$$

we have $\mathcal{P}os_P(t_2) = \{\Lambda\}$*,* $\mathcal{P}os_A(t_2) = \{\Lambda, 1\}$*, and* $\mathcal{P}os_{nil}(t_2) = \{1, 1.1\}$*. For the annotated term*

$$t_3 = \texttt{length'}_{nil|(0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),$$

we have $\mathcal{P}os_P(t_3) = \{\Lambda\}$*,* $\mathcal{P}os_A(t_3) = \{\Lambda\}$*, and* $\mathcal{P}os_{nil}(t_3) = \{1.1\}$ *Finally, for the annotated term*

$$t_4 = \texttt{length'}_{(-1)|(0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),$$

we have $\mathcal{P}os_P(t_4) = \{\Lambda\}$*,* $\mathcal{P}os_A(t_4) = \{\Lambda, 1\}$*, and* $\mathcal{P}os_{nil}(t_4) = \{1.1\}$*. Figure 5 shows these four terms with the positive and on-demand symbols framed with a full or dashed line, respectively.* ■

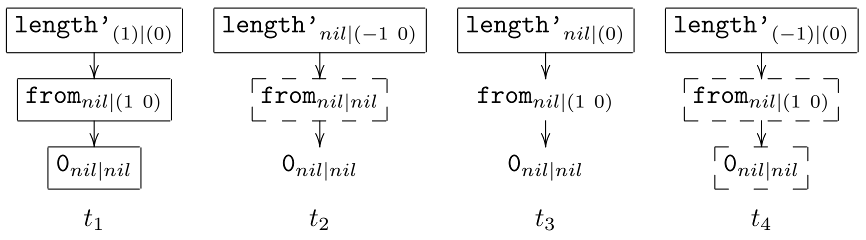


Figure 5: The positive and the on-demand symbols (full and dashed frame, respectively) of the terms of Example 17.

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:

$$ADP_l(t) = \begin{cases} D \cap \mathcal{P}os_A(t) & \text{if } D \cap (\mathcal{P}os_P(t) \cup \mathcal{P}os_{nil}(t)) = \varnothing \\ & \text{ where } D = DP_l(erase(t)) \\ \varnothing & \text{otherwise} \end{cases}$$

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 with the intended meaning of strategy annotations, since empty positions would have been evaluated before and positive (but not empty) positions will be evaluated later.

Example 18 *(Example 17 cont'd) Consider the lhs $l =$ `length'(nil)`. For the annotated term t_1, we have $DP_l(erase(t_1)) = \{1\}$ but $ADP_l(t_1) = \varnothing$, since position 1 is demanded by l but it is a positive position, as shown in Example 17. For the annotated term t_2, we have $ADP_l(t_2) = \varnothing$, since position 1 is again demanded by l but it is rooted by a symbol with an empty annotation list. For the annotated term t_3, we have $ADP_l(t_3) = \varnothing$, since position 1 is again demanded by l but it is not an active position, as shown in Example 17, because no index 1 or -1 appears in the memory list of symbol `length'`. Finally, for the annotated term t_4, we have $ADP_l(t_4) = \{1\}$.* ■

4.3 Selection of the demanded redex

When $ADP_l(s)$ contains more than one active demanded position, we use an order $\leq_s$ to select the redex position, where $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. This is related to the $min_{\leq_{lex}}$ and $max_{\leq_{lex}}$ functions used in Section 3.2. In contrast to Section 3.2, however, the order $\leq_s$ is based on the annotations fixed by the user. The intuitive idea is that every time we have either $\varphi(f) = (\cdots\ i\ \cdots\ j\ \cdots)$, $\varphi(f) = (\cdots\ -i\ \cdots\ j\ \cdots)$, $\varphi(f) = (\cdots\ i\ \cdots\ -j\ \cdots)$, or $\varphi(f) = (\cdots\ -i\ \cdots\ -j\ \cdots)$ for $i, j \in \mathbb{N}$ and $i \neq j$, then we can say that any position $p \in i.\mathcal{P}os(s|_i)$ is preferable (for the user) to any position $q \in j.\mathcal{P}os(s|_j)$, for a term s such that $root(s) = f$;

in symbols, $p \leq_s q$. Given a term $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, the total order[11] $\leq_s$ 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 the leftmost occurrence of i or $-i$ appears before the leftmost occurrence of j or $-j$ in the list $lookout(root(s))$, then $i.p \leq_s j.q$.

Example 19 *(Example 16 cont'd) Recall that* $\varphi(\texttt{lt}) = (-2\ -1\ 0)$ *and* $\varphi(\texttt{\_+\_}) = (1\ 2\ 0)$. *For the term* $t_2 = \texttt{lt(foo,0 + 0)}$ *and its annotated version*

$$\varphi(t_2) = \texttt{lt}_{nil|(-2\ -1\ 0)}(\texttt{foo}_{nil|(0)}, \texttt{0}_{nil|nil}\ \texttt{+}_{nil|(1\ 2\ 0)}\ \texttt{0}_{nil|nil})$$

we have that $\mathcal{P}os_P(\varphi(t_2)) = \{\Lambda\}$ *and* $\mathcal{P}os_A(\varphi(t_2)) = \{\Lambda, 1, 2, 2.1, 2.2\}$. *Then, since the user specified that the second argument of* `lt` *is preferable to the first one and the first argument of* `_+_` *is preferable against the second one, we have that*

$$\Lambda <_{\varphi(t_2)} 2 <_{\varphi(t_2)} 2.1 <_{\varphi(t_2)} 2.2 <_{\varphi(t_2)} 1.$$

■

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:

$$OD_\mathcal{R}(s) = \begin{cases} \varnothing & \text{if } ADP_\mathcal{R}(s) = \varnothing \\ \{min_{\leq_s}(ADP_\mathcal{R}(s))\} & \text{otherwise} \end{cases}$$

Note that the $OD_\mathcal{R}$ is a deterministic strategy due to $min_{\leq_s}$.

Example 20 *(Example 19 cont'd) For* $t_2 = \texttt{lt(foo,0 + 0)}$, *we have* $OD_\mathcal{R}(\varphi(t_2)) = \{2\}$ *with* $ADP_\mathcal{R}(\varphi(t_2)) = \{1, 2\}$. ■

Remark 1 *Note that the order relation provided by the user is important in our computational model and can determine program properties; see [Eke00] for the same point w.r.t. positive annotations. Recall Example 12 with*

```
eq geq(s(X),s(Y)) = geq(X,Y) .
eq geq(X,0) = true .
```

For each strategy $(1\ 2\ 0)$, $(2\ 1\ 0)$, $(1\ -2\ 0)$, $(-2\ 1\ 0)$, *and* $(-1\ -2\ 0)$ *for symbol* `geq`, *calls* `geq(foo,0 + 0)` *and* `geq(0 + 0,foo)` *do not terminate. However, for each strategy* $(-1\ 2\ 0)$, $(2\ -1\ 0)$ *and* $(-2\ -1\ 0)$, *call* `geq(foo,0 + 0)` *does terminate but call* `geq(0 + 0,foo)` *does not.*

Intuitively, there is an implicit evaluation order in the rules for symbol `geq` *that suggests reducing first the second argument and then, possibly, the first*

[11] A more general notion that captures this idea is given in [Ohl02, Def. A.1.4].

argument. Thus, we should include index 2 in the strategy for `geq` *because its evaluation is necessary, and index -1 (instead of index 1) because its evaluation is (only) sometimes necessary. Alternatively, we can include indices -1 and -2, but imposing an explicit order between them, i.e., with $(-2\ -1\ 0)$.*

It is worth noting that some sophisticated lazy strategies try to find out this implicit evaluation order[12] *automatically from the rules, see for instance [Esc03, EMT05]. This is an interesting line of research which is outside the scope of the paper. Here we adopt a simpler user-guided approach, which is coherent with the use of strategies in* OBJ*-like languages.*

4.4 A new reduction model for on-demand evaluation strategies

In order to avoid the generation of different rewriting subsequences in Definition 2, we use symbols $\overline{f}$ to mark non-evaluable positions. This helps the evaluation of a demanded position to come back to the position which demanded a particular 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 every symbol f between p and the root (excluding them) marked as $\overline{f}$, 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 21 *Consider the program of Example 3 and the term*

$$t = \texttt{seriesPos}_{(1\ 2)|(0)}(\texttt{s}_{(1)|nil}(\texttt{s}_{(1)|nil}(\texttt{0}_{nil|nil})),\\ \texttt{0}_{nil|nil}\ \cdot_{(1\ -2)|nil}\ \texttt{from}_{nil|(1\ 0)}(\texttt{s}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})))$$

When we mark the position 2.2 in t, we have that symbol $\_ \cdot \_$ at position 2, i.e., "$\cdot_{(1\ -2)|nil}$", gets marked with a bar, i.e., "$\overline{\cdot}_{(1\ -2)|nil}$", to denote that this symbol is in the middle of a demanded computation and is non-evaluable (i.e., when the execution strategy considers this subterm, then it must jump to the position above it):

$$mark(t, 2.2)\\ = \texttt{seriesPos}_{(1\ 2)|(0)}(\texttt{s}_{(1)|nil}(\texttt{s}_{(1)|nil}(\texttt{0}_{nil|nil})),\\ \texttt{0}_{nil|nil}\ \overline{\cdot}_{(1\ -2)|nil}\ \texttt{from}_{nil|(1\ 0)}(\texttt{s}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})))$$

■

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.

Definition 3 (On-demand evaluation strategy (*ODE*)) *Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and an arbitrary E-strategy map φ for $\mathcal{F}$, $eval^\sharp_\varphi : \mathcal{T}(\mathcal{F},\mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F},\mathcal{X}))$ is defined as $eval^\sharp_\varphi(t) = \{erase(s) \in \mathcal{T}(\mathcal{F},\mathcal{X}) \mid \langle\varphi(t),\Lambda\rangle \xrightarrow{\sharp}{}^!_\varphi \langle s,\Lambda\rangle\}$.*

[12]This idea is formally expressed with the notion of *neededness* of a computation. See [AL02] for a relation of neededness and demandness in programming languages.

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

1. $t|_p = f_{L|nil}(t_1, \ldots, t_k)$, $s = t$, *and* $p = q.i$ *for some* i*; or*
2. $t|_p = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $s = t[f_{L_1 \oplus i|L_2}(t_1, \ldots, t_k)]_p$, *and* $q = p.i$*; or*
3. $t|_p = f_{L_1|-i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $s = t[f_{L_1 \oplus -i|L_2}(t_1, \ldots, t_k)]_p$, *and* $q = p$*; or*
4. $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k) = \sigma(l')$, $erase(l') = l$, $s = t[\sigma(\varphi(r))]_p$, *and* $q = p$ *for some* $l \rightarrow r \in R$ *and substitution* σ*; 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*, $OD_{\mathcal{R}}(t|_p) = \{p'\}$, $s = t[mark(t|_p, p')]_p$, *and* $q = p.p'$*; 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 position (note that the index is stored because, in the future, there can be negative indices under this positive one). Case 3 only stores the negative index for further use. Cases 4, 5, and 6 consider the attempt to match the term against the left-hand sides of the program rules. Case 4 applies if the considered (unlabeled) subterm is a redex (which is, then, contracted). If the subterm is not a redex, cases 5 and 6 are considered (possibly involving some on-demand evaluation). We use the lists of indices labeling the symbols for fixing the concrete positions on which it is safe to allow on-demand evaluations; in particular, the first (memoizing) list is crucial for achieving this (by means of the function *lookout* 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 22 *Consider the modules of Example 3 with the strategy* `(1 -2 0)` *for symbol* `_._` *of sort* `LNat`*. Figure 6 shows the first steps of the evaluation sequence of the term* `pi(s(s(0)))` *via* $eval^{\sharp}_{\varphi}$ *(for each step, we indicate which case of Definition 3 has been used, we frame the indices involved in that step, and we underline the redex reduced, if any). We continue until the symbol* `_._` *is obtained at the top. That evaluation sequence corresponds to the following (much simpler) general rewriting sequence:*

$$\begin{array}{l}\underline{\texttt{pi(s(s(0)))}}\\ \quad\to \texttt{seriesPos(s(s(0)), }\underline{\texttt{from(0)}}\texttt{)}\\ \quad\to \texttt{seriesPos(s(s(0)), 0 . }\underline{\texttt{from(s(0))}}\texttt{)}\\ \quad\to \underline{\texttt{seriesPos(s(s(0)), 0 . s(0) . from(s(s(0))))}}\\ \quad\to \texttt{1/s(0) . seriesNeg(s(0), from(s(s(0))))}\end{array}$$

Note that in the step $(*)$ *of Figure 6, i.e., when an on-demand evaluation starts and the evaluation moves suddenly from position* Λ *to position* 2.2*, the symbol* `_._` *is overlined, as explained in Example 21.* ■

In the following example, we illustrate in detail why our notion of demandedness does work for Example 4.

Example 23 *Consider Examples 4 and 11. The on-demand evaluation of* `length(from(0))` *under* ODE *is the following:*

$$\begin{array}{l}\langle\ \texttt{length}_{nil|(\boxed{\text{-1}}\ 0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),\ \Lambda\ \rangle\\ \quad\overset{\sharp}{\to}_{\varphi}\ \langle\ \underline{\texttt{length}_{(-1)|(\boxed{0})}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil}))},\ \Lambda\ \rangle\\ \quad\overset{\sharp}{\to}_{\varphi}\ \langle\ \texttt{length'}_{nil|(\boxed{0})}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),\ \Lambda\ \rangle\\ \quad\overset{\sharp}{\to}_{\varphi}\ \langle\ \texttt{length'}_{nil|nil}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})),\ \Lambda\ \rangle\end{array}$$

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 are computed. However, there are no demanded positions, since the memoizing list of strategy annotations for* `length'` *is empty (see* $ADP_l(t_3)$ *in Example 18 above). Therefore, we obtain* `length'(from(0))` *as the computed value of the evaluation, according to case* 5 *of Definition 3.* ■

In the following Section, we study different properties of the on-demand evaluation strategy (*ODE*).

5 Properties of the on-demand evaluation strategy (*ODE*)

We first give a general property stating the uselessness of repeated indices in a strategy map by defining a mapping $\|\_\| : \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi}) \to \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ that removes redundant non-zero indices.

$$\langle\ \mathtt{pi}_{nil|(\boxed{1}\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{nil|(\boxed{1}\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{s}_{nil|(\boxed{1}\ 0)}(\mathtt{0}_{nil|nil}))),\ 1.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{0}_{nil|\boxed{nil}}))),\ 1.1.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{s}_{(1)|(\boxed{0})}(\mathtt{0}_{nil|nil}))),\ 1.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{0}_{nil|nil}))),\ 1.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|(\boxed{0})}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}))),\ 1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{pi}_{(1)|(0)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}))),\ 1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \underline{\mathtt{pi}_{(1)|(\boxed{0})}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})))},\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (4)} \langle\ \mathtt{seriesPos}_{nil|(\boxed{1}\ 2\ 0)}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil})),\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1)|(2\ 0)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil})),\ 1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1)|(\boxed{2}\ 0)}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\mathtt{from}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil})),\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\mathtt{from}_{nil|(\boxed{1}\ 0)}(\mathtt{0}_{nil|nil})),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\mathtt{from}_{(1)|(0)}(\mathtt{0}_{nil|\boxed{nil}})),\ 2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\mathtt{s}_{(1)|nil}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})),\underline{\mathtt{from}_{(1)|(\boxed{0})}(\mathtt{0}_{nil|nil})}),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (4)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{nil|(\boxed{1}\ -2\ 0)}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|\boxed{nil}}\ \cdot_{(1)|(-2\ 0)}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{(1)|(\boxed{-2}\ 0)}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (3)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|(\boxed{0})}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|\boxed{nil}}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(\boxed{0})}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|nil}\ \mathtt{from}_{nil|(1\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (6)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{from}_{nil|(\boxed{1}\ 0)}(\mathtt{s}_{nil|(1\ 0)}(\mathtt{0}_{nil|nil}))),\ 2.2\ \rangle\ (*)$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{from}_{(1)|(0)}(\mathtt{s}_{nil|(\boxed{1}\ 0)}(\mathtt{0}_{nil|nil}))),\ 2.2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{from}_{(1)|(0)}(\mathtt{s}_{(1)|(0)}(\mathtt{0}_{nil|\boxed{nil}}))),\ 2.2.1.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{from}_{(1)|(0)}(\mathtt{s}_{(1)|(\boxed{0})}(\mathtt{0}_{nil|nil}))),\ 2.2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{from}_{(1)|(0)}(\mathtt{s}_{(1)|\boxed{nil}}(\mathtt{0}_{nil|nil}))),\ 2.2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \underline{\mathtt{from}_{(1)|(\boxed{0})}(\mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil}))}),\ 2.2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{nil|(\boxed{1}\ -2\ 0)}\ \cdots),\ 2.2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (2)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|\boxed{nil}}(\mathtt{0}_{nil|nil})\ \cdot_{(1)|(-2\ 0)}\ \cdots),\ 2.2.1\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{(1)|(\boxed{-2}\ 0)}\ \cdots),\ 2.2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (3)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{(1\ -2)|(\boxed{0})}\ \cdots),\ 2.2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (5)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \bar{\cdot}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{(1\ -2)|\boxed{nil}}\ \cdots),\ 2.2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (1)} \langle\ \mathtt{seriesPos}_{(1\ 2)|(0)}(\ldots,\mathtt{0}_{nil|nil}\ \boxed{\bar{\cdot}}_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{(1\ -2)|nil}\ \cdots),\ 2\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (7)} \langle\ \underline{\mathtt{seriesPos}_{(1\ 2)|(\boxed{0})}(\ldots,\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|nil}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{(1\ -2)|nil}\ \cdots)},\ \Lambda\ \rangle$$

$$\xrightarrow{\sharp}_{\varphi\ (4)} \langle\ \mathtt{1/}_{nil|(1\ 0)}\ \mathtt{s}_{(1)|nil}(\mathtt{0}_{nil|nil})\ \cdot_{nil|(1\ -2\ 0)}\ \cdots\ ,\ \Lambda\ \rangle$$

Figure 6: On-demand evaluation of term `pi(s(s(0)))` by Definition 3.

Given a list L of integers, let $\|L\|$ be the list without repeated non-zero indices, i.e., we keep only the leftmost occurrence of each index $i \in \mathbb{Z} - \{0\}$ appearing in L. Given an E-strategy map, let $\|\varphi\|$ be the E-strategy map obtained from φ by removing repeated non-zero indices for each symbol $f \in \mathcal{F}$. Note that $\|\varphi\| \sqsubseteq \varphi$, for every E-strategy map φ. For an E-strategy map φ and a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, we define $\|\_\| : \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \to \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ as $\|x_{nil|nil}\| = x_{nil|nil}$ for $x \in \mathcal{X}$ and $\|f^\sharp_{L_1|L_2}(t_1, \ldots, t_n)\| = f^\sharp_{L'_1|L'_2}(\|t_1\|, \ldots, \|t_n\|)$ such that $L'_1 \oplus L'_2 = \|L_1 \oplus L_2\|$, $L'_1 \sqsubseteq L_1$, and $L'_2 \sqsubseteq L_2$.

Theorem 1 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. Then, $\langle t, \Lambda\rangle \overset{\sharp}{\to}{}^*_\varphi \langle s, \Lambda\rangle$ if and only if $\langle \|t\|, \Lambda\rangle \overset{\sharp}{\to}{}^*_{\|\varphi\|} \langle \|s\|, \Lambda\rangle$.*

5.1 Comparison with Nagaya's model

Now, we compare our *ODE* with the Nagaya's evaluation strategy using only positive indices to show that ours is a sound extension. We define a mapping $\lfloor\_\rfloor_{\mathbb{N}} : \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \to \mathcal{T}(\mathcal{F}^{\mathbb{N}}_\varphi, \mathcal{X}^{\mathbb{N}}_\varphi)$ that removes negative indices and transforms terms $\mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, used by our *ODE* strategy, into terms $\mathcal{T}(\mathcal{F}^{\mathbb{N}}_\varphi, \mathcal{X}^{\mathbb{N}}_\varphi)$, used by Nagaya's strategy.

Given a list L of integers, let $\lfloor L\rfloor$ be the list without negative indices. Given an E-strategy map, let $\lfloor\varphi\rfloor$ be the E-strategy map obtained from φ by removing all negative indices for each symbol $f \in \mathcal{F}$. Note that $\lfloor\varphi\rfloor \sqsubseteq \varphi$, for all E-strategy map φ. For an E-strategy map φ and a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, we define $\lfloor\_\rfloor_{\mathbb{N}} : \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \to \mathcal{T}(\mathcal{F}^{\mathbb{N}}_\varphi, \mathcal{X}^{\mathbb{N}}_\varphi)$ as $\lfloor x_{nil|nil}\rfloor_{\mathbb{N}} = x_{nil}$ for $x \in \mathcal{X}$ and $\lfloor f^\sharp_{L_1|L_2}(t_1, \ldots, t_n)\rfloor_{\mathbb{N}} = f_{\lfloor L_2\rfloor}(\lfloor t_1\rfloor_{\mathbb{N}}, \ldots, \lfloor t_n\rfloor_{\mathbb{N}})$.

The following theorem shows that, for positive strategy annotations, each reduction step with $\overset{\sharp}{\to}_\varphi$ exactly corresponds to Nagaya's original relation $\overset{\mathbb{N}}{\to}_\varphi$ of Section 3.1.

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

Proof. Straightforward according to Definitions 1 and 3. □

5.2 Meaningful negative annotations

In the following, we show that for E-strategy maps φ whose positive part (the sublists of positive indices) $\lfloor\varphi\rfloor$ is *canonical*, extra negative annotations can be completely disregarded. We first introduce the notion of canonical strategy maps, then a mapping $\lfloor\_\rfloor : \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \to \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ that removes negative indices (note that the mapping $\lfloor\_\rfloor$ does not transform terms between different domains, as the mapping $\lfloor\_\rfloor_{\mathbb{N}}$ does), and finally the notion of an alternating strategy map.

Sometimes, it is interesting to get rid of the order among indices in local strategies and, then, we use *replacement maps* [Luc98]. 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 order $\sqsubseteq$ on $M_{\mathcal{F}}$, i.e., 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 $f \in \mathcal{F}$.

Given a TRS $\mathcal{R}$, $\mu^{can}_{\mathcal{R}}$ is 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.* Note that $\mu^{can}_{\mathcal{R}}$ is easily obtained from $\mathcal{R}$: for all $f \in \mathcal{F}$, for all $i \in \{1, \ldots, ar(f)\}$, $i \in \mu^{can}_{\mathcal{R}}(f)$ iff $\exists l \in L(\mathcal{R})$ and $p \in \mathcal{P}os_{\mathcal{F}}(l)$ such that $root(l|_p) = f$ and $p.i \in \mathcal{P}os_{\mathcal{F}}(l)$ [Luc98, Luc02a]. 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 $\mu_{\varphi}(f) = \{abs(i) \mid i \in \varphi(f) \wedge i \neq 0\}$. We say that φ is a *canonical* E-strategy map (and, slightly abusing notation, we write $\varphi \in CM_{\mathcal{R}}$) if $\mu_{\varphi} \in CM_{\mathcal{R}}$.

For an E-strategy map φ and a term $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, we define $\lfloor \_ \rfloor : \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi}) \to \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ as $\lfloor x_{nil|nil} \rfloor = x_{nil|nil}$ for $x \in \mathcal{X}$ and $\lfloor f^{\sharp}_{L_1|L_2}(t_1, \ldots, t_n) \rfloor = f^{\sharp}_{\lfloor L_1 \rfloor | \lfloor L_2 \rfloor}(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$.

Given an E-strategy map φ, we say φ is an *alternating* strategy map if whenever index i appears in $\varphi(f)$ for $f \in \mathcal{F}$, then index $-i$ does not appear in $\varphi(f)$, and vice versa.

Theorem 3 *Let $\mathcal{R}$ be a TRS and φ be an alternating E-strategy map such that $\lfloor \varphi \rfloor \in CM_{\mathcal{R}}$. Let $t, s \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, $p \in \mathcal{P}os_A(t)$, and $q \in \mathcal{P}os_A(s)$. Then, $\langle t, p \rangle \overset{\sharp}{\to}_{\varphi} \langle s, q \rangle$ if and only if $\langle \lfloor t \rfloor, p \rangle \overset{\sharp}{\to}^{=}_{\lfloor \varphi \rfloor} \langle \lfloor s \rfloor, q \rangle$.*

This result means that negative annotations are only meaningful if the positive indices do not include all indices in the canonical replacement map of the TRS. This motivates when and why we should use negative annotations (as it was suggested in Remark 1 above, and also in [AL02]):

1. the i-th argument of a symbol f is annotated with index i if all occurrences of f in the left-hand side of the rules contain a non-variable i-th argument;
2. 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;
3. in any other case, index $-i$ is given to f.

This is specially interesting when termination can be proved for those positive and negative annotations but cannot be proved for the canonical map including only positive annotations, since adding negative annotations to those positive annotations may yield termination while providing completeness; Example 3 illustrates such a situation. This has also implications on the execution time, as shown in Table 2 in Section 8.

The following result is a consequence of Theorems 2 and 3.

Corollary 1 *Let $\mathcal{R}$ be a TRS and φ be an alternating E-strategy map such that $\lfloor\varphi\rfloor \in CM_{\mathcal{R}}$. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, $p \in \mathcal{P}os_A(t)$, and $q \in \mathcal{P}os_A(s)$. Then, $\langle t,p\rangle \overset{\sharp}{\to}_\varphi \langle s,q\rangle$ if and only if $\langle \lfloor t\rfloor_{\mathbb{N}}, p\rangle \overset{\mathbb{N}}{\to}^{=}_{\lfloor\varphi\rfloor} \langle \lfloor s\rfloor_{\mathbb{N}}, q\rangle$.*

5.3 Ensuring head-normal forms

Example 23 above shows that restricting the evaluation by using on-demand strategy annotations can result in terms which are not even head-normal forms w.r.t. $\to_{\mathcal{R}}$, e.g., term `length'(from(0))` is a normal form w.r.t. $\to_\varphi$ but is not a normal form (and also not a head-normal form) w.r.t. the general rewriting relation $\to_{\mathcal{R}}$. The following result establishes conditions ensuring that the normal forms computed by the on-demand strategy (*ODE*) $\overset{\sharp}{\to}_\varphi$ are ordinary head-normal forms w.r.t. $\to_{\mathcal{R}}$.

Theorem 4 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. If $s \in eval^\sharp_\varphi(t)$, then s is a head-normal form of t.*

Left-linearity (LL) and constructor system (CS) conditions cannot be dropped, as [Luc98, Nag99] has shown for positive annotations and [Luc01a] has shown for on-demand rewriting (*ODR*). The following two counterexamples are an adaptation of the ones in [Luc01a]. Note that the models in [OF00, NO01] are neither able to compute the head-normal forms in the following examples, since the considered programs do not fulfill the LL and CS conditions, which are implicitly required by [OF00, NO01].

Example 24 *Consider the following TRS $\mathcal{R}$ from [Luc01a] which is not a CS:*

```
fmod NONCS is
  sort S .
  ops a b : -> S .
  op f : S -> S [strat (-1 0)] .
  op g : S S -> S [strat (1 2 0)] .
  var X : S .
  eq f(g(X,a)) = a .
  eq g(a,b) = g(b,a) .
endfm
```

The term $t =$ `f(g(a,b))` is not a head-normal form since we have

$$\texttt{f(}\underline{\texttt{g(a,b)}}\texttt{)} \to \underline{\texttt{f(g(b,a))}} \to \texttt{a}.$$

However, the head-normal form a *of t is not computed by $\overset{\sharp}{\rightarrow}_\varphi$:*

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

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

```
fmod NONLEFTLINEAR is
  sort S .
  ops a b : -> S .
  op f : S -> S [strat (-1 -2 0)] .
  var X : S .
  eq f(X,X) = X .
  eq a = b .
endfm
```

Term $t =$ `f(a,b)` *is not a head-normal form since we have*

$$\mathtt{f(\underline{a},b)} \rightarrow \underline{\mathtt{f(b,b)}} \rightarrow \mathtt{b}.$$

However, the head-normal form b *of t is not computed by $\overset{\sharp}{\rightarrow}_\varphi$:*

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

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

Theorem 4 suggests the following extension of the *normalization via φ-normalization procedure* of solution 1 in Section 1 to obtain normal forms of a term t: given an E-strategy map φ and $s = f(s_1,\ldots,s_k) \in eval^{\sharp}_{\varphi}(t)$, the evaluation of s proceeds by (recursively) normalizing $s_1,\ldots,s_k$ using the terms collected in $eval^{\sharp}_{\varphi}$ as intermediate values. It is not difficult to see that confluence and the termination of the TRS $\mathcal{R}$ (w.r.t. $\overset{\sharp}{\rightarrow}_\varphi$) guarantee that this procedure actually describes a normalizing strategy (see [Luc01a, Luc02a]).

In the following section, we show that the on-demand strategy (*ODE*) improves (i) Nakamura and Ogata's model, (ii) *lazy rewriting* (*LR*), a popular demand-driven technique to perform lazy functional computations which inspired the development of local strategies in OBJ, and (iii) *on-demand rewriting* (*ODR*), the natural extension of context-sensitive rewriting to deal with on-demand strategy annotations.

6 Comparison with other techniques dealing with on-demand annotations

In the following we show that our on-demand strategy (ODE) is strictly more restrictive than all previous proposals. The reasons are:

1. First, we are able to stop some useless computations thanks to the constructor test performed in $DP_l(t)$ and the non-positive and non-empty position test performed in $ADP_l(t)$.
2. Second, we always follow the order of evaluation specified by the user's strategy, and thus we can have termination where other proposals do not.

Note that this last statement can also be interpreted in the opposite way, thus meaning that we can get stuck because the user provided a malicious strategy. However, in such a situation, we are still coherent with the user's intention.

The *on-demand evaluation strategy* introduced in Nakamura's thesis [Nak02, Chapter 5] is another proposal for on-demand evaluation that splits a local strategy into two sequences: one specifying an order of arguments to be reduced, and the other specifying an order of arguments to be matched with the lhs of a rewrite rule. This approach of splitting into positive and "on-demand" annotations was already considered by the *on-demand rewriting* [Luc01a], introduced in Section 6.3 below, and it is contrary to our approach of allowing the user full control on the order of evaluation. That is, in Nakamura's approach [Nak02, Chapter 5], on-demand annotations cannot be attached to the position of the index 0, which determines when the whole term should be checked for evaluation and a strategy of our framework mixing positive indices, negative indices and several indices 0 cannot be specified in Nakamura's approach, as shown in the following example.

Example 26 *Consider the following two versions of this simple program that differ only in the position where negative annotation -2 is placed.*

```
fmod TEST1 is                     fmod TEST2 is
  sort S .                          sort S .
  var Y : S .                       var Y : S .
  ops a b c : -> S .                ops a b c : -> S .
  op f : S S -> S                   op f : S S -> S
     [strat (-2 0 1 0) .               [strat (0 1 -2 0) .
  eq f(a,Y) = a .                   eq f(a,Y) = a .
  eq f(b,c) = b .                   eq f(b,c) = b .
  op g : -> S .                     op g : -> S .
  eq g = a .                        eq g = a .
  op foo : -> S.                    op foo : -> S.
  eq foo = foo .                    eq foo = foo .
endm                              endm
```

Our on-demand evaluation strategy (ODE) will provide two different behaviors for term $t =$ `f(g,foo)`: for theory `TEST1` *the computation of term t never* *stops whereas for theory* TEST2 *the computation of term t stops and returns term* a*. However, if you disconnect negative annotations from indices* 0 *and positive annotations, as in Nakamura's approach, then you will always get the non-terminating behavior of theory* TEST1*. Nakamura's approach would be similar to having all the negative annotations pasted right before each index* 0*, i.e., strategy* $(-2\ 0\ 1\ -2\ 0)$ *for symbol* f*.* ■

Nevertheless, Nakamura's approach is quite similar to Nakamura and Ogata's strategy [NO01], compared below with our approach, and suffers the same inconsistencies of Nakamura and Ogata's strategy that we have shown in Section 3.2.3.

6.1 Nakamura and Ogata's model

In Section 3.2, we have introduced Nakamura and Ogata's model for arbitrary strategy annotations [NO01]. In the following, we relate their model to ours.

A defined function symbol $f \in \mathcal{D}$ is *completely defined* [TeR03] if it does not occur in any ground term in normal form, that is to say that functions are reducible on all ground terms (of appropriate sort). A TRS $\mathcal{R}$ is *completely defined* [TeR03] if each defined symbol of the signature is completely defined.

Let φ be a strategy map. We say φ is in *lexicographic order* if for each $f \in \mathcal{F}$ and for each $i, j \in \mathbb{Z}$ such that $\varphi(f) = (\cdots\ i\ j\ \cdots)$, we have that $|i| \leq |j|$. We say φ is *standard* if for each $f \in \mathcal{D}$, $\varphi(f)$ has only one occurrence of index 0 at the end. We say that a term l is in *φ-order* if for each pair of active positions $p_1, p_2 \in \mathcal{P}os_A(\varphi(l))$ such that $p_1 \leq_{\varphi(l)} p_2$, if p_2 is a non-variable position, i.e., $p_2 \in \mathcal{P}os_{\mathcal{F}}(l)$, then p_1 is also a non-variable position, i.e., $p_1 \in \mathcal{P}os_{\mathcal{F}}(l)$. Intuitively, the term l is in φ-order if it follows with non-variable positions the evaluation order fixed by the strategy φ. We say that a TRS $\mathcal{R}$ is in *φ-order* if every $l \in L(\mathcal{R})$ is. For an E-strategy map φ and a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, we define $\lfloor \_ \rfloor_{\mathbb{Z}} : \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \to \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_\varphi, \mathcal{X}^{\mathbb{Z}}_\varphi)$ as $\lfloor x_{nil|nil} \rfloor_{\mathbb{Z}} = x_{nil}$ for $x \in \mathcal{X}$ and $\lfloor f^\sharp_{L_1|L_2}(t_1, \ldots, t_n) \rfloor_{\mathbb{Z}} = f^1_{L_2}(\lfloor t_1 \rfloor_{\mathbb{Z}}, \ldots, \lfloor t_n \rfloor_{\mathbb{Z}})$.

Theorem 5 *Let $\mathcal{R}$ be a left-linear completely defined CS and φ be a standard E-strategy map in lexicographic order such that $\varphi \in CM_{\mathcal{R}}$. Let $\mathcal{R}$ be in φ-order. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. Then, $\langle t, \Lambda \rangle \overset{\sharp}{\to}^{!}_{\varphi} \langle s, \Lambda \rangle$ if and only if $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\to}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$.*

Note that we cannot relate one $\overset{\mathbb{Z}}{\to}_\varphi$-step to one $\overset{\sharp}{\to}_\varphi$-step due to the recursive calls to $\overset{\mathbb{Z}}{\to}_\varphi$ in Definition 2. Requiring $\mathcal{R}$ to be completely defined is necessary, as it was shown in Example 14 with term `lt(foo,0)`, and then we have to restrict ourselves also to the conditions of Theorem 4 to ensure that every $\overset{\mathbb{Z}}{\to}_\varphi$-normal form is a head-normal form. Moreover, we have to restrict ourselves to a strategy φ in lexicographic order because $\overset{\mathbb{Z}}{\to}_\varphi$ always selects the minimum demanded position w.r.t. such lexicographic order for each lhs. And we have to restrict ourselves to a TRS in φ-order because $\overset{\mathbb{Z}}{\to}_\varphi$ selects the maximum position among the demanded positions collected from all lhs's. A canonical strategy is also necessary because of inappropriate propagations of on-demand flags in $\overset{\mathbb{Z}}{\rightarrow}_{\varphi}$. Note that Example 3 satisfies all these properties and Theorem 5 can be applied, ensuring that evaluations are equivalent. However, Examples 4, 12, and 14 do not satisfy these properties, i.e., the strategy is not canonical in Example 4, the program is not in φ-order in Example 12, and the program is not completely defined in Example 14.

6.2 Lazy rewriting (*LR*)

In lazy rewriting (*LR*) [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 and with ℓ for the so-called *lazy* ones. Labeled terms are terms in $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ where $\mathcal{L} = \{e, \ell\}$. 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 27 *Consider the program of Example 3 (viewed as a TRS) and the replacement map μ given by $\mu(\_ . \_) = \{1\}$ and $\mu(f) = \{1, \ldots, ar(f)\}$ for any other $f \in \mathcal{F}$. Then, the labeling of*

$$s = \texttt{seriesPos(s(s(0)),0 . from(s(0)))}$$

is

$$t = label_\mu(s) = \texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \texttt{from}^\ell\texttt{(s}^e\texttt{(0}^e\texttt{)))}.$$

All positions are eager positions except position 2.2 *which is lazy. Figure 7 shows all the eager and lazy positions framed by a full or dashed line, respectively.* ■

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.

Example 28 *(Example 27 cont'd) Positions* $\Lambda, 1, 1.1, 1.1.1, 2$ *and* 2.1 *are active in term t of Example 27; positions* 2.2.1 *and* 2.2.1.1 *are eager but* not *active, since position* 2.2 *above is lazy in t. See Figure 7.* ■

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:

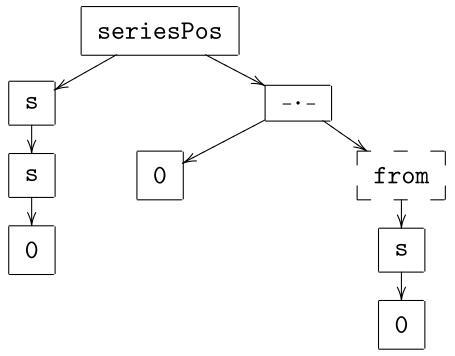


Figure 7: Eager and lazy positions (full and dashed frame, respectively) in the term `seriesPos(s(s(0)),0 . from(s(0)))` w.r.t. lazy rewriting (LR).

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 redices at active nodes'* [FKW00].

Definition 4 (Matching modulo laziness [FKW00]**)** *Let $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a linear term, $t \in \mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$, 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, $t \overset{\mathsf{R}}{\to}_{\mu} s$, 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 29 *(Example 27 cont'd) Some $\xrightarrow{\mathsf{A}}$ and $\overset{\mathsf{R}}{\to}_{\mu}$ steps for term* $t = \texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \texttt{from}^{\ell}\texttt{(s}^e\texttt{(0}^e\texttt{)))}$ *are (we underline the re-*

dex reduced and frame the superscript involved in each step):

$$\begin{array}{l}
\texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \texttt{from}^{\boxed{\ell}}\texttt{(s}^e\texttt{(0}^e\texttt{)))} \\
\xrightarrow{\mathsf{A}} \texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \underline{\texttt{from}^e\texttt{(s}^e\texttt{(0}^e\texttt{)))}} \\
\xrightarrow{\mathsf{R}}_\mu \texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \texttt{s}^e\texttt{(0}^e\texttt{)}\ \texttt{.}^{\boxed{\ell}}\ \texttt{from}^\ell\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{))))} \\
\xrightarrow{\mathsf{A}} \texttt{seriesPos}^e\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{)),0}^e\ \texttt{.}^e\ \texttt{s}^e\texttt{(0}^e\texttt{)}\ \texttt{.}^e\ \texttt{from}^\ell\texttt{(s}^e\texttt{(s}^e\texttt{(0}^e\texttt{))))}
\end{array}$$

Note that this last term is an $\xrightarrow{\mathsf{A}}$*-normal form, since the symbol* $\texttt{from}^\ell$ *is under a variable position in the corresponding lhs* `seriesPos(s(N),X . Y . XS)`. ■

Definition 5 (Lazy rewriting) [FKW00] *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-$eval_\mu(t)$ *of a term* $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ *using* LR *is given by* LR-$eval_\mu(t) = \{erase(s) \in \mathcal{T}(\mathcal{F},\mathcal{X}) \mid label_\mu(t) \xrightarrow{\mathsf{LR}}{}^!_\mu\, s\}$.

In the following, we show that each evaluation step of the on-demand strategy (*ODE*) is included into some evaluation steps of lazy rewriting. First, we give some definitions auxiliary for Theorem 6 below. 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 symbol at the position under consideration into eager symbols, as follows:

$lazy^p_\varphi(t) = \rho''_p(\rho'_t(label_{\mu_{\lfloor\varphi\rfloor}}(erase(t))))$ where

(1) $b \in \{e, \ell\}$,

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

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

Example 30 *Consider Example 4 with the E-strategy map* $\varphi(\_.\_) = (1)$, $\varphi(\texttt{length}) = (-1\ 0)$, $\varphi(\texttt{length'}) = (0)$, *and* $\varphi(f) = (1\ \cdots\ ar(f)\ 0)$ *for any other* $f \in \mathcal{F}$. *Let us consider the term* $t = \texttt{length'(from(0))}$. *The labeling using* φ *is*

$$\varphi(t) = \texttt{length'}_{nil|(0)}(\texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil}))$$

and its version transformed into LR *is*

$$lazy^\Lambda_\varphi(t) = \texttt{length}^e(\texttt{from}^\ell(\texttt{0}^e)).$$

However, we also have

$$lazy^1_\varphi(t) = \texttt{length}^e(\texttt{from}^e(\texttt{0}^e)),$$

which corresponds to the activated term obtained in Example 31 below. ■

The following theorem shows that each evaluation step of the on-demand strategy (*ODE*) 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 order $\leq_{lazy}$ for expressing this fact).

Theorem 6 *Let $\mathcal{R}$ be a left-linear TRS and φ be an E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, $p \in \mathcal{P}os_A(t)$ and $\mu = \mu_{\lfloor\varphi\rfloor}$. If $\langle t, p\rangle \overset{\sharp}{\to}_\varphi \langle s, q\rangle$ 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 $lazy^q_\varphi(s) \leq_{lazy} s'$.*

Note that lazy rewriting is defined only for left-linear TRSs (see Definition 4). Note also that one $\overset{\sharp}{\to}_\varphi$-step may correspond to zero $\xrightarrow{\mathsf{LR}}{}^*_\mu$-steps because of those $\overset{\sharp}{\to}_\varphi$-steps dealing only with annotations/marks on symbols; and can correspond to more than one $\xrightarrow{\mathsf{LR}}{}^*_\mu$-step because a demanded evaluation needs only one $\overset{\sharp}{\to}_\varphi$-step (i.e., when $OD_\mathcal{R}(t) = \{p'\}$) when several activation $\overset{\mathsf{A}}{\to}$-steps may be necessary to make position p active. In general, our strategy is more restrictive than *LR* as the following example shows.

Example 31 *Consider the program $\mathcal{R}$ (as a TRS) and the E-strategy map φ of Example 4. Consider the replacement map $\mu = \mu_{\lfloor\varphi\rfloor}$, i.e., $\mu(\texttt{length}) = \mu(\texttt{length'}) = \varnothing$, $\mu(\_.\_) = \{1\}$ and $\mu(f) = \{1, \ldots, ar(f)\}$ for any other $f \in \mathcal{F}$. Note that if no positive annotation is provided for an argument of a symbol, then* LR *freely demands this argument. In Example 40 below, we prove that $\mathcal{R}$ is $\overset{\sharp}{\to}_\varphi$-terminating. However,* LR *enters an infinite reduction sequence starting with the term $label_\mu(\texttt{length(from(0))})$ (we underline the redex reduced and frame the superscript involved at each step):*

$$\begin{array}{l}
\underline{\texttt{length}^e(\texttt{from}^\ell(\texttt{0}^e))} \\
\quad \overset{\mathsf{R}}{\to}_\mu \texttt{length'}^e(\texttt{from}^{\boxed{\ell}}(\texttt{0}^e)) \\
\quad \overset{\mathsf{A}}{\to} \texttt{length'}^e(\underline{\texttt{from}^e(\texttt{0}^e)}) \\
\quad \overset{\mathsf{R}}{\to}_\mu \underline{\texttt{length'}^e(\texttt{0}^e \;.^e\; \texttt{from}^\ell(\texttt{s}^e(\texttt{0}^e)))} \\
\quad \overset{\mathsf{R}}{\to}_\mu \texttt{s}^e(\texttt{length'}^e(\texttt{from}^{\boxed{\ell}}(\texttt{s}^e(\texttt{0}^e)))) \\
\quad \overset{\mathsf{A}}{\to} \texttt{s}^e(\texttt{length'}^e(\underline{\texttt{from}^e(\texttt{s}^e(\texttt{0}^e))})) \\
\quad \overset{\mathsf{LR}}{\to}_\mu \cdots
\end{array}$$

That is, in contrast to $\overset{\sharp}{\to}_\varphi$ (where $\varphi(\texttt{length'}) = (0)$), LR *can evaluate position 1 in the term* `length'(from(0))`*.* ■

By Theorems 4 and 6, we have the following result relating completeness of both strategies.

Theorem 7 *Let $\mathcal{R}$ be a left-linear completely defined CS and φ be an E-strategy map such that $\varphi \in CM_\mathcal{R}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. If $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\langle \varphi(t), \Lambda\rangle \overset{\sharp}{\to}{}^!_\varphi \langle s, \Lambda\rangle$, then $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^!_\mu lazy^\Lambda_\varphi(s)$.*

The complete definedness condition as well as those of Theorem 4 are necessary to ensure that every $\overset{\sharp}{\to}_{\varphi}$-normal form is a head-normal form with a constructor symbol at the top. We can use Example 14 with term `lt(foo,0)` as a counterexample. Note that the opposite direction of the Theorem cannot be proved because *LR* does not follow any particular order for activating symbols in contrast to $\overset{\sharp}{\to}_{\varphi}$. However, we can also provide the following result.

Theorem 8 *Let $\mathcal{R}$ be a left-linear completely-defined CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. Let $\mathcal{R}$ be $\overset{\sharp}{\to}_{\varphi}$-terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\langle \varphi(t), \Lambda\rangle \overset{\sharp}{\to}^{!}_{\varphi} \langle s, \Lambda\rangle$ if and only if $lazy^{\Lambda}_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{!}_{\mu} lazy^{\Lambda}_{\varphi}(s)$.*

6.3 On-demand rewriting (*ODR*)

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 $p \in \mathcal{P}os^{\mu}(t)$. As we mentioned above, context–sensitive rewriting inspired the frozen arguments of system modules in Maude whereas lazy rewriting inspired the local strategies of OBJ and Maude.

Example 32 *Consider $\mathcal{R}$ in Example 3 and the replacement map $\mu(\_ . \_) = \{1\}$ and $\mu(f) = \{1, \ldots, ar(f)\}$ for any other $f \in \mathcal{F}$ (which corresponds to the strategy map φ of Example 3). Then, we have:*

$\texttt{seriesPos(s(s(0)),}\underline{\texttt{from(0)}}\texttt{)} \hookrightarrow_{\mu} \texttt{seriesPos(s(s(0)),0 . from(s(0)))}$

and this last term cannot be further μ-rewritten. ■

The $\hookrightarrow_{\mu}$-normal forms are called μ-normal forms. 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 $f \in \mathcal{F}$, each index $j \in \mu_D(f)$ aims at enabling reductions on subterm t_j of a function call $f(t_1, \ldots, t_j, \ldots, t_k)$ if it can eventually lead to matching 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 at 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 6 (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 $t \xrightarrow{p} s$ 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 6). 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 have no variable position of l covering p (condition 2(2) of Definition 6); otherwise, the reduction at $t|_p$ would not improve the matching.

Example 33 *(Example 32 cont'd) Consider the following on-demand replacement map $\mu_D(\_.\_) = \{2\}$ and $\mu(f) = \varnothing$ for any other $f \in \mathcal{F}$ (where the union of μ and μ_D corresponds to the strategy map φ of Example 3). Now we have the following sequence starting from* `seriesPos(s(s(0)),from(0))`*, until symbol* `_._` *is obtained at the top:*

$\texttt{seriesPos(s(s(0)),}\underline{\texttt{from(0)}}\texttt{)}$
$\quad \hookrightarrow_{\langle \mu, \mu_D \rangle}$ $\texttt{seriesPos(s(s(0)),0 . }\underline{\texttt{from(s(0))}}\texttt{)}$
$\quad \hookrightarrow_{\langle \mu, \mu_D \rangle}$ $\underline{\texttt{seriesPos(s(s(0)),0 . s(0) . from(s(s(0))))}}$
$\quad \hookrightarrow_{\langle \mu, \mu_D \rangle}$ `s(0) . seriesNeg(s(0),from(s(s(0))))`

but

$\texttt{seriesPos(s(s(0)),0 . s(0) . }\underline{\texttt{from(s(s(0)))}}\texttt{)}$
$\quad \not\hookrightarrow_{\langle \mu, \mu_D \rangle}$ `seriesPos(s(s(0)),0 . s(0) . s(s(0)) . from(s(s(s(0)))))`

since $2.2.2 \notin \mathcal{P}os(l)$ *with* $l =$ `seriesPos(s(N),X . Y . XS)`*. Figure 8 shows the reducible and demanded parts of the term*

"`seriesPos(s(s(0)),0 . from(s(0)))`*"*

w.r.t. on-demand rewriting, framed with a full or dashed line, respectively. ■

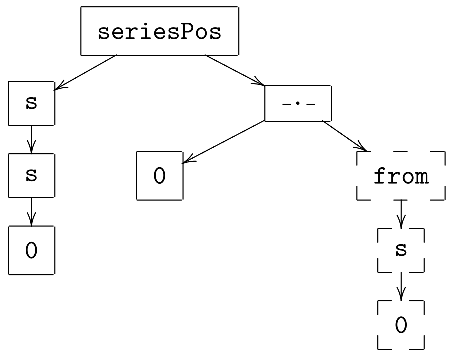


Figure 8: Reducible and on-demand positions (full and dashed frame, respectively) of term "`seriesPos(s(s(0)),0 . from(s(0)))`" w.r.t. on-demand rewriting (*ODR*).

In the following, we show that each evaluation step of the on-demand strategy (*ODE*) 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 that 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 $\langle \varphi(s), \Lambda\rangle \overset{\sharp}{\to}{}^*_\varphi \langle t, p\rangle$.

Theorem 9 *Let $\mathcal{R}$ be a left-linear CS and φ be an E-strategy map such that $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $\mu, \mu_D \in M_\mathcal{F}$ be such that $\mu = \mu_{\lfloor\varphi\rfloor}$ 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 $\langle t, p\rangle \overset{\sharp}{\to}_\varphi \langle s, q\rangle$ 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, the on-demand strategy (*ODE*) is more restrictive than *ODR* as the following example shows.

Example 34 *Consider the program:*

```
fmod TEST is
  sort Nat .
  op 0 : -> Nat .
  op foo : Nat -> Nat .
  op f : Nat -> Nat [strat (-1 0)] .
  op g : Nat -> Nat  [strat (-1 0)] .
  var X : Nat .
  eq foo = foo .
  eq f(X) = 0 .
  eq g(0) = 0 .
endfm
```

Consider the TRS underlying this program and the following replacement maps (whose union corresponds to the strategy map φ given in the module) $\mu(\mathtt{g}) = \mu(\mathtt{f}) = \varnothing$ and $\mu(h) = \{1, \ldots, ar(f)\}$ for each other symbol h; also, $\mu_D(\mathtt{g}) = \mu_D(\mathtt{f}) = \{1\}$ and $\mu_D(h) = \varnothing$ for each other symbol h. The term $t = \mathtt{g(f(foo))}$ has a single terminating evaluation sequence using $\overset{\sharp}{\rightarrow}_\varphi$:

$$
\begin{array}{l}
\langle\ \mathtt{g}_{nil|(\boxed{-1}\ 0)}(\mathtt{f}_{nil|(-1\ 0)}(\mathtt{foo}_{nil|(0)})),\ \Lambda\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \mathtt{g}_{(-1)|(\boxed{0})}(\mathtt{f}_{nil|(-1\ 0)}(\mathtt{foo}_{nil|(0)})),\ \Lambda\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \mathtt{g}_{(-1)|(0)}(\mathtt{f}_{nil|(\boxed{-1}\ 0)}(\mathtt{foo}_{nil|(0)})),\ 1\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \mathtt{g}_{(-1)|(0)}(\underline{\mathtt{f}_{(-1)|(\boxed{0})}(\mathtt{foo}_{nil|(0)})}),\ 1\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \mathtt{g}_{(-1)|(0)}(\mathtt{0}_{nil|\boxed{nil}}),\ 1\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \underline{\mathtt{g}_{(-1)|(\boxed{0})}(\mathtt{0}_{nil|nil})},\ \Lambda\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle\ \underline{\mathtt{g}_{(-1)|(\boxed{0})}(\mathtt{0}_{nil|nil})},\ \Lambda\ \rangle \\
\quad \overset{\sharp}{\rightarrow}_\varphi\ \langle \mathtt{0}_{nil|nil}, \Lambda \rangle
\end{array}
$$

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

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

the following non-terminating reduction sequence is also possible:

$$\mathtt{g(f(}\underline{\mathtt{foo}}\mathtt{))} \hookrightarrow_{\langle\mu,\mu_D\rangle} \mathtt{g(f(}\underline{\mathtt{foo}}\mathtt{))} \hookrightarrow_{\langle\mu,\mu_D\rangle} \cdots$$

Note that Λ is a positive position of $\mathtt{g(f(foo))}$ which is eventually rewritten into redex $\mathtt{g(0)}$, and positions 1 and 1.1 of $\mathtt{g(f(foo))}$ are always demanded by the lhs $\mathtt{g(0)}$, whereas only position 1 should be demanded, as it is done in ODE. ■

Moreover, the condition in Theorem 9 that φ be an E-strategy map such that $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ for $c \in \mathcal{C}$ cannot be dropped.

Example 35 *Consider the following module containing a non-terminating function* `inf`*:*

```
fmod TEST is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat [strat (1 0)] .
  op inf : -> Nat .
  op f : Nat -> Nat [strat (-1 0)] .
  var X : Nat . var XS : LNat .
  eq inf = s(inf) .
  eq f(s(X)) = 0 .
endfm
```

Consider the TRS underlying this program and the following replacement maps (whose union corresponds to the strategy map φ): $\mu(\mathtt{f}) = \varnothing$, and $\mu(h) = \{1, \ldots,$

$ar(f)\}$ for all other symbol h; and $\mu_D(\mathtt{f}) = \{1\}$ and $\mu_D(h) = \varnothing$ for all other symbol h. Now the term $t = \mathtt{f(inf)}$ has a unique normalizing evaluation sequence under ODR*:*

$$\mathtt{f(\underline{inf})} \hookrightarrow_{\langle\mu,\mu_D\rangle} \underline{\mathtt{f(s(inf))}} \hookrightarrow_{\langle\mu,\mu_D\rangle} \mathtt{0}$$

Subterm `inf` *of* `f(s(inf))` *is under the variable* `X` *of lhs* `f(s(X)` *and then it is not further evaluated. However, the evaluation sequence for $\overset{\sharp}{\to}_\varphi$ is non-terminating:*

$$\begin{array}{l}
\langle \mathtt{f}_{nil|(\boxed{\text{-1}}\ 0)}(\mathtt{inf}_{nil|(0)}), \Lambda\rangle \\
\overset{\sharp}{\to}_\varphi\ \langle \mathtt{f}_{(-1)|(\boxed{0})}(\mathtt{inf}_{nil|(0)}), \Lambda\rangle \\
\overset{\sharp}{\to}_\varphi\ \langle \mathtt{f}_{(-1)|(0)}(\underline{\mathtt{inf}_{nil|(\boxed{0})}}), 1\rangle \\
\overset{\sharp}{\to}_\varphi\ \langle \mathtt{f}_{(-1)|(0)}(\mathtt{s}_{nil|(\boxed{1}\ 0)}(\mathtt{inf}_{nil|(0)})), 1\rangle \\
\overset{\sharp}{\to}_\varphi\ \langle \mathtt{f}_{(-1)|(0)}(\mathtt{s}_{(1)|(0)}(\underline{\mathtt{inf}_{nil|(\boxed{0})}})), 1.1\rangle \\
\overset{\sharp}{\to}_\varphi\ \cdots
\end{array}$$

The reason is that we follow the user's annotations and thus we always evaluate the argument of `s` *according to its strategy* (1 0) *and generate an infinite computation.* ■

In Theorem 9, the condition of $\mathcal{R}$ being a *CS* cannot be dropped either. An argument similar to the one in Example 35 can be used if we consider a defined symbol at a non-root position of a lhs which is given a positive annotation in the strategy map.

We have the following result relating completeness of both strategies.

Theorem 10 *Let $\mathcal{R}$ be a left-linear completely-defined CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$, $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$, and $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $\mathcal{R}$ be $\overset{\sharp}{\to}_\varphi$-terminating. Let $\mu, \mu_D \in M_{\mathcal{F}}$ be such that $\mu = \mu_{\lfloor\varphi\rfloor}$ and $\mu \sqcup \mu_D = \mu_\varphi$. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. Then, $\langle\varphi(t),\Lambda\rangle \overset{\sharp}{\to}{}^!_\varphi \langle s,\Lambda\rangle$ if and only if $t \hookrightarrow^!_{\langle\mu,\mu_D\rangle} erase(s)$.*

To conclude this section, Figure 9 summarizes the precise relationships between the different strategies, i.e., under which conditions two strategies have the same behavior. In the following section, we consider an important aspect of the definition of a suitable (on-demand) execution strategy, namely termination, and formulate a method for proving termination of the on-demand evaluation (*ODE*).

7 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

Strategies	Conditions for equivalence w.r.t. on-demand evaluation (*ODE*)	
Nagaya's model		P-φ (Theo 2)
Nakamura and Ogata's model	(LL, CD, CS)-$\mathcal{R}$,	(S, LO, C)-φ (Theo 5)
Lazy rewriting (*LR*)	(LL, CD, CS, T)-$\mathcal{R}$,	(C, E0)-φ (Theo 8)
On-demand rewriting (*ODR*)	(LL, CD, CS, T)-$\mathcal{R}$,	(C, E0, C0)-φ (Theo 10)

LL	Left-linear TRS	CS	Constructor TRS
CD	Completely defined TRS	T	$\overset{\sharp}{\to}_{\varphi}$-terminating TRS
LO	Lexicographic order strategy	S	Standard strategy
P	Positive strategy	E0	Strategy ended with 0 for each defined symbol
C	Canonical strategy		
C0	Constructor symbols do not have positive annotations		

Figure 9: Summary of the relation between different on-demand strategies and the on-demand evaluation (*ODE*).

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 a new 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 [Luc02c, GM04] for surveys on this topic) and tools such as MU-TERM [Luc06] and AProVE [GTSKF04]. This provides a formal framework for proving termination of lazy rewriting (*LR*).

A simple modification of such transformation provides a sound technique for proving the φ-termination of TRSs for arbitrary strategy annotations φ. Here, as in [Luc01a, Luc02b], by φ-termination of a TRS $\mathcal{R}$ we mean the absence of infinite $\overset{\sharp}{\to}_{\varphi}$-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 $l \to r \in R$, we define

$$NegPos(l) = min_{\leq_{\varphi(l)}}(\{p.i \in \mathcal{P}os_{\mathcal{F}}(l) \cap \mathcal{P}os_{A}(l) \mid -i \in \varphi(root(l|_p))\})$$

Note that we take the minimum to ensure that the transformed program will follow the proper evaluation order. Assume that $NegPos(l) = \{p.i\}$ for some p

and i (i.e., $NegPos(l) \neq \varnothing$) and let $f = root(l|_p)$. Then, $\mathcal{R}^\diamond = (\mathcal{F}^\diamond, R^\diamond)$ and $\varphi^\diamond$ are as follows: $\mathcal{F}^\diamond = \mathcal{F} \cup \{f_i\}$, where f_i is a new symbol of arity $ar(f_i) = ar(f)$, and

$$R^\diamond = R - \{l \to r\} \cup \{l' \to r, l[\overline{x}]_P \to l'[\overline{x}]_P\}$$

where $l' = l[f_i(l|_{p.1}, \ldots, l|_{p.ar(f)})]_p$, $P = minimal_\leq(\{p' \in \mathcal{P}os_A(l) - \mathcal{P}os_P(l) \mid p.i \leq_{\varphi(l)} p'\})$, and $\overline{x}$ is a sequence of new different variables. We let $\varphi^\diamond(f_i) = (i_1 \cdots i \cdots i_n)$ such that $\varphi(f) = (i_1 \cdots -i \cdots i_n)$, and $\varphi^\diamond(h) = \varphi(h)$ for any other $h \in \mathcal{F}$. 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 (though we remove all negative annotations only at the end).

Example 36 *Consider the following program with the strategy* `(1 -2 0)` *for symbol* $\_.\_$ *of module* `LIST-NAT`*:*

```
fmod LIST-NAT is
  ...
  op _._ : Nat LNat -> LNat [strat (1 -2 0)] .
  ...
endfm

fmod LIST-NAT-3RD is
  protecting LIST-NAT .
  vars X Y Z : Nat . var XS : LNat .
  op 3rd : LNat -> Nat .
  eq 3rd(X . Y . Z . XS) = Z .
endfm
```

The one-step transformation $\mathcal{R}^\diamond$ generates the following two rules from l (we write `_[.]_` *instead of symbol $\_.\_{}_2$ or $\_._2\_$):*

```
  eq 3rd(X . XS) = 3rd(X [.] XS) .
  eq 3rd(X [.] Y . Z . XS) = Z .
```

Note that the left-most symbol $\_.\_$ in l is the one selected for the transformation because it is the minimum element (w.r.t. orders $\leq_{\varphi(l)}$ and $\leq$) of the positions in l with a negative index, i.e.,

$$\{\ 1.2,\ 1.2.2\ \} = \{p.i \in \mathcal{P}os_\mathcal{F}(l) \cap \mathcal{P}os_A(l) \mid -i \in \varphi(root(l|_p))\}$$

and $1.2 \leq_{\varphi(l)} 1.2.2$. ■

The transformation proceeds in this way (repeatedly constructing $\mathcal{R}^\diamond$ and $\varphi^\diamond$) until obtaining $\mathcal{R}^\natural = (\mathcal{F}^\natural, R^\natural)$ and $\varphi^\natural$ such that $NegPos(l) = \varnothing$ for each $l \in R^\natural$, and then we remove all negative annotations from $\varphi^\natural$. Note that this removal of negative annotations does not modify the evaluation behavior of $\mathcal{R}^\natural$, as it is proved in Theorem 3.

Example 37 *(Example 36 cont'd) The final transformed program $\mathcal{R}^\natural$ is:*

```
fmod LIST-NAT-NONEG is
  ...
  op _._ : Nat LNat -> LNat [strat (1 0)] .
  op _[.]_ : Nat LNat -> LNat [strat (1 2 0)] .
  ...
endfm

fmod LIST-NAT-3RD-NONEG is
  protecting LIST-NAT-NONEG .
  vars X Y Z : Nat . var XS : LNat .
  op 3rd : LNat -> Nat .
  eq 3rd(X . XS) = 3rd(X [.] XS) .
  eq 3rd(X [.] Y . XS) = 3rd(X [.] Y [.] XS) .
  eq 3rd(X [.] Y [.] Z . XS) = Z .
endfm
```

■

Finally, we can state a sufficient condition for $\overset{\sharp}{\rightarrow}_{\varphi}$-termination as $\overset{\sharp}{\rightarrow}_{\varphi^\natural,\mathcal{R}^\natural}$-termination in the transformed program $\mathcal{R}^\natural$.

Theorem 11 (Termination) *Let $\mathcal{R}$ be a CS and φ be a standard E-strategy map. If the relation $\overset{\sharp}{\rightarrow}_{\varphi^\natural,\mathcal{R}^\natural}$ is terminating, then the relation $\overset{\sharp}{\rightarrow}_{\varphi,\mathcal{R}}$ is also terminating.*

Note that the opposite does not hold, as shown in the following example.

Example 38 *Consider the module* `LIST-NAT-3RD` *of Example 36 with the following extra lines:*

```
  op g : -> Nat .
  eq g = 3rd(0 . g . 0 . nil) .
```

For the term `g`*, Figure 10 shows its evaluation sequence according to $\overset{\sharp}{\rightarrow}_{\varphi}$, which corresponds to the (rather simple) general rewriting sequence:*

$$\underline{\texttt{g}} \rightarrow \underline{\texttt{3rd(0 . g . 0 . nil)}} \rightarrow \texttt{0}$$

No other sequence can be computed[13] *for term* `g` *using $\overset{\sharp}{\rightarrow}_{\varphi}$. If we add the two lines defining symbol* `g` *to the module* `LIST-NAT-3RD-NONEG` *of Example 37, we have an evaluation sequence from term* `g` *using the relation $\overset{\sharp}{\rightarrow}_{\varphi}$ that corresponds*

[13] The TRS without symbol `g`, i.e., the TRS in Example 36, is terminating and we can prove it using the technique of this section, the proof is similar to the one in Figure 11. However, the TRS including symbol `g` is terminating but we cannot prove it with the transformation technique of this section.

$$\langle\ \underline{\mathtt{g}_{nil|(\boxed{0})}},\ \Lambda\ \rangle$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{nil|(\boxed{1}\ 0)}(\mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ \Lambda\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{(1)|(0)}(\mathtt{0}_{nil|nil}\ \cdot_{nil|(\boxed{1}\ -2\ 0)} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ 1\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{(1)|(0)}(\mathtt{0}_{nil|\boxed{nil}}\ \cdot_{(1)|(-2\ 0)} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ 1.1\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{(1)|(0)}(\mathtt{0}_{nil|nil}\ \cdot_{(1)|(\boxed{-2}\ 0)} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ 1\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{(1)|(0)}(\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|(\boxed{0})} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ 1\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \mathtt{3rd}_{(1)|(0)}(\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|\boxed{nil}} \\
&\qquad \mathtt{g}_{nil|(0)}\ \cdot_{nil|(1\ -2\ 0)} \\
&\qquad\qquad \mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil}),\ 1\ \rangle
\end{aligned}$$

$$\begin{aligned}
\xrightarrow{\sharp}_{\varphi} \langle\ & \underline{\mathtt{3rd}_{(1)|(\boxed{0})}(\mathtt{0}_{nil|nil}\ \cdot_{(1\ -2)|nil}} \\
&\qquad \underline{\mathtt{g}_{nil|(0)}}\ \underline{\cdot_{nil|(1\ -2\ 0)}} \\
&\qquad\qquad \underline{\mathtt{0}_{nil|nil}\ \cdot_{nil|(1\ -2\ 0)}\ \mathtt{nil}_{nil|nil})},\ \Lambda\ \rangle
\end{aligned}$$

$$\xrightarrow{\sharp}_{\varphi} \langle\ \mathtt{0}_{nil|nil},\ \Lambda\ \rangle$$

Figure 10: Evaluation of term $\mathtt{g}$ of Example 38

to the following (infinite) ordinary rewriting sequence (recall from Example 37 that $\varphi(\_.\_) = (1\ 0)$ *and* $\varphi(\_[.]\_) = (1\ 2\ 0)$*):*

$$\begin{aligned}
\underline{\texttt{g}} &\to \underline{\texttt{3rd(0 . g . 0 . nil)}}\\
&\to \texttt{3rd(0 [.] }\underline{\texttt{g}}\texttt{ . 0 . nil)}\\
&\to \texttt{3rd(0 [.] }\underline{\texttt{3rd(0 . g . 0 . nil)}}\texttt{ . 0 . nil)}\\
&\to \texttt{3rd(0 [.] 3rd(0 [.] }\underline{\texttt{g}}\texttt{ . 0 . nil) . 0 . nil)} \to \cdots
\end{aligned}$$

■

Then, since the transformed program does not include any negative annotation, we can use μ-termination of *CSR* to approximate $\overset{\mathbb{N}}{\to}_{\varphi}$-termination, see [Luc01b] for further details. It is well-known that *CSR* does not completely capture the φ-termination property of an OBJ program with only positive strategy annotations, since the order between positive indices included in a strategy does not appear in a replacement map. 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 used along the paper can be proved terminating by this technique.

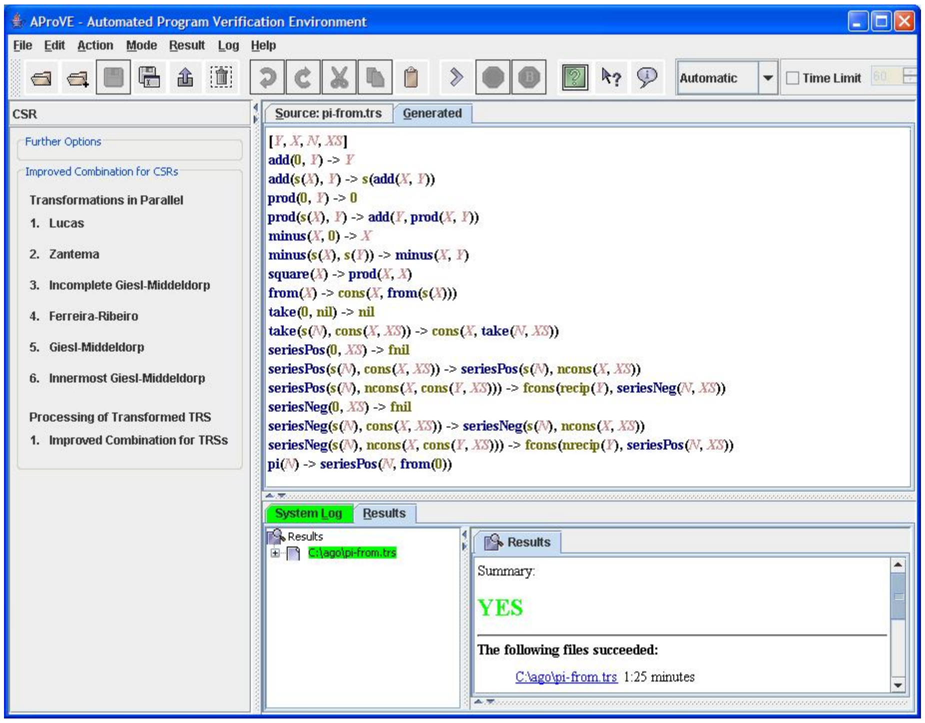
Example 39 *Consider the module* PI *in Example 3 with the strategy* $(1\ -2\ 0)$ *for symbol* $\_.\_$*, i.e., the only change is the following line in module* LIST-NAT*:*

```
fmod LIST-NAT is
  ...
  op _._ : Nat LNat -> LNat [strat (1 -2 0)] .
  ...
endfm
```

Then, the transformed program $\mathcal{R}^{\natural}$ *is (we show only the changes in module* LIST-NAT *and the module* PI*, which is the only one transformed):*

```
fmod LIST-NAT is
  ...
  op _._ : Nat LNat -> LNat [strat (1 0)] .
  op _[.]_ : Nat LNat -> LNat [strat (1 2 0)] .
  ...
endfm

fmod PI-NONEG is
  protecting LIST-NAT .
  protecting LIST-FRAC .
  op pi : Nat -> LIntFrac .
  ops seriesPos seriesNeg : Nat LNat -> LIntFrac .
  vars N X Y : Nat . var XS : LNat .
  eq seriesPos(0,XS) = nil .
  eq seriesPos(s(N),X . XS) =  seriesPos(s(N),X [.] XS) .
  eq seriesPos(s(N),X [.] Y . XS) =  1/ Y . seriesNeg(N,XS) .
  eq seriesNeg(0,XS) = nil .
```


APro VE - Automated Program Verification Environment
File Edit Action Mode Result Log Help
Automatic
Time Limit
CSR
Further Options
Improved Combination for CSRs
Transformations in Parallel
1. Lucas
2. Zantema
3. Incomplete Giesl-Middeldorp
4. Ferreira-Ribeiro
5. Giesl-Middeldorp
6. Innermost Giesl-Middeldorp
Processing of Transformed TRS
1. Improved Combination for TRSs
Source: pi-from.trs
Generated
System Log
Results
C:\ago\pi-from.trs

```
[Y, X, N, XS]
add(0, Y) -> Y
add(s(X), Y) -> s(add(X, Y))
prod(0, Y) -> 0
prod(s(X), Y) -> add(Y, prod(X, Y))
minus(X, 0) -> X
minus(s(X), s(Y)) -> minus(X, Y)
square(X) -> prod(X, X)
from(X) -> cons(X, from(s(X)))
take(0, nil) -> nil
take(s(N), cons(X, XS)) -> cons(X, take(N, XS))
seriesPos(0, XS) -> fnil
seriesPos(s(N), cons(X, XS)) -> seriesPos(s(N), ncons(X, XS))
seriesPos(s(N), ncons(X, cons(Y, XS))) -> fcons(recip(Y), seriesNeg(N, XS))
seriesNeg(0, XS) -> fnil
seriesNeg(s(N), cons(X, XS)) -> seriesNeg(s(N), ncons(X, XS))
seriesNeg(s(N), ncons(X, cons(Y, XS))) -> fcons(nrecip(Y), seriesPos(N, XS))
pi(N) -> seriesPos(N, from(0))
```

Results

Summary:

YES

The following files succeeded:

C:\ago\pi-from.trs 1:25 minutes

Figure 11: Output of APro VE for the proof of termination of the transformed program of Example 39

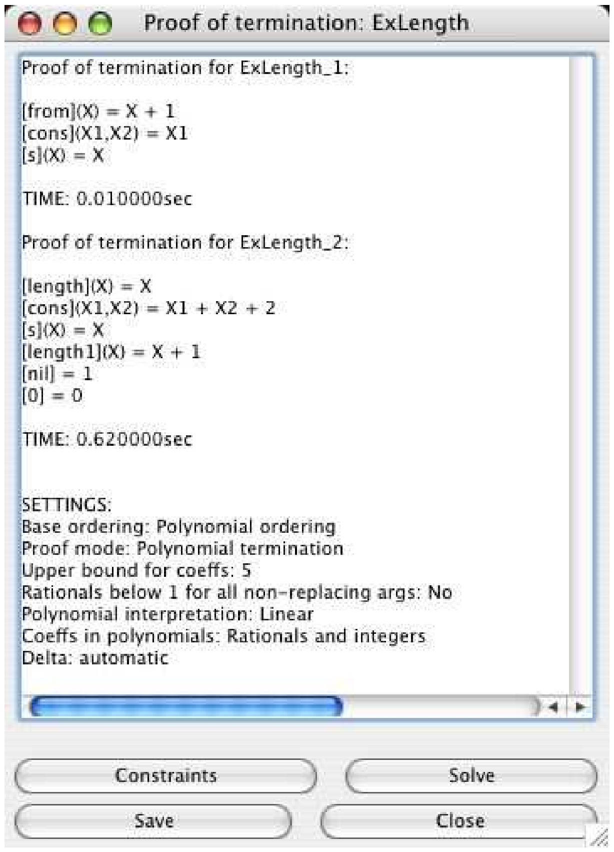
```
    eq seriesNeg(s(N),X . XS) = seriesNeg(s(N),X [.] XS) .
    eq seriesNeg(s(N),X [.] Y . XS) = -1/ Y . seriesPos(N,XS) .
    eq pi(N) = seriesPos(N,from(0)) .
  endfm
```

The $\mu_{\varphi^\natural}$*-termination of* $\mathcal{R}^\natural$ *is automatically proved by the tools* MU-TERM[14] *and* APro VE[15]*. Figure 11 shows the outcome of the tool* APro VE*.* ■

Example 40 *Consider the module* LIST-NAT-LENGTH *of Example 4. Our transformation returns the original module with only the following change in module* LIST-NAT*:*

```
  fmod LIST-NAT is
    ...
    op _._ : Nat LNat -> LNat [strat (1 0)] .
    ...
  endfm
```

[14] Available at http://www.dsic.upv.es/~slucas/csr/termination/muterm.
[15] Available at http://www-i2.informatik.rwth-aachen.de/AProVE/.



```
Proof of termination for ExLength_1:

[from](X) = X + 1
[cons](X1,X2) = X1
[s](X) = X

TIME: 0.010000sec

Proof of termination for ExLength_2:

[length](X) = X
[cons](X1,X2) = X1 + X2 + 2
[s](X) = X
[length1](X) = X + 1
[nil] = 1
[0] = 0

TIME: 0.620000sec


SETTINGS:
Base ordering: Polynomial ordering
Proof mode: Polynomial termination
Upper bound for coeffs: 5
Rationals below 1 for all non-replacing args: No
Polynomial interpretation: Linear
Coeffs in polynomials: Rationals and integers
Delta: automatic
```

Figure 12: Output of MU-TERM for the proof of termination of the transformed program of Example 40

The $\mu_{\varphi^\natural}$*-termination of module* LIST-NAT-LENGTH *can be automatically proved, e.g., by the tool* MU-TERM *as shown in Figure 12. More precisely, termination of each module is proved separately and then a modularity result is applied [GL02].* ■

8 Experiments

In order to demonstrate the practicality of the on-demand evaluation strategy (*ODE*) proposed in this paper, two interpreters have been implemented, one in Haskell (version 6.4 available at `http://www.haskell.org/ghc`) and the other one in Full Maude (version 2.1.1.a available at `http://maude.cs.uiuc.edu`). The first system is called OnDemandOBJ and is described in [AEL03] and the

	`PI`	`LIST-NAT-LENGTH`
OnDemandOBJ	659 ms.	< 1 ms.
ODMaude	675 sec.	10 ms.
CafeOBJ	370 ms.	*overflow*
Maude	*unavailable*	*unavailable*

Table 1: Execution of call `pi((3^2)^2)` and `length(from(0))`.

second system is called ODMaude and is described in [DEL04]. Both systems are publicly available at `http://www.dsic.upv.es/users/elp/soft.html`.

Tables 1 and 2 show the runtimes[16] of the benchmarks for the different OBJ-family systems. CafeOBJ (version 1.4.6 available at `http://www.ldl.jaist.ac.jp/Research/CafeOBJ/system.html`) is developed in Lisp at the Japan Advanced Institute of Science and Technology (JAIST); Maude (version 2.1.1 available at `http://maude.cs.uiuc.edu/`) is developed in C++ and maintained by the University of Illinois at Urbana-Champaign. Maude provides only computations with positive annotations whereas CafeOBJ provides computations with negative annotations as well, using the on-demand evaluation of [OF00]. OnDemandOBJ and ODMaude compute with negative annotations using the on-demand evaluation strategy (*ODE*) provided in this paper. That is, they compute the same and the only difference is how are they implemented. In Table 1, mark *overflow* indicates that the execution raised a memory overflow without computing the associated normal form (see Example 4), whereas the mark *unavailable* in Tables 1 and 2 indicates that the program cannot be executed in such an OBJ implementation. Note that since Maude is implemented in C++, typical execution times are close to 0 milliseconds. In ODMaude, programs are executed using a very naïve implementation of the on-demand strategy (*ODE*) of Section 4 and thus runtimes are much greater. The objective of the implementation in ODMaude is to give an extremely compact and readable implementation of the strategy whereas its implementation is improved in OnDemandOBJ.

Table 1 compares the existing OBJ implementations through the evaluation of a concrete term of the module `PI` of Example 3 and also the term `length(from(0))` of the module `LIST-NAT-LENGTH` of Example 4. It witnesses that negative annotations are actually useful in practice because they allow reductions in situations where only positive annotations are not sufficient and has better computational properties than previous proposals.

On the other hand, Table 2 illustrates the interest[17] of using negative annotations to improve the behavior of programs:

- the benchmark `NAT_eager` represents the module `NAT` of Example 1 encoding functions `_+_`, `_-_`, `_*_`, and `_^2` over natural numbers using only positive annotations. Every k-ary symbol f is given a strategy $(1\ 2\ \cdots k\ 0)$

[16]The average of 10 executions measured in a AMD Athlon XP machine running Fedora Core 3.

[17]We exclude ODMaude from this comparison because runtimes are extremely high, more than 30 minutes for `NAT_neg` and `NAT_lazy`.

	`NAT_eager`	`NAT_can`	`NAT_neg`	`NAT_lazy`
OnDemandOBJ	406 ms.	1005 ms.	<1 ms.	<1 ms.
	500 ms.	2059 ms.	2050 ms.	1750 ms.
CafeOBJ	170 ms.	460 ms.	<1 ms.	<1 ms.
	540 ms.	850 ms.	830 ms.	850 ms.
Maude	1 ms.	10 ms.	*unavailable*	*unavailable*
	2 ms.	12 ms.	*unavailable*	*unavailable*

Table 2: Execution of terms `0 - (((3 ^2)^2)^2)` and `(((3 ^2)^2)^2) - (((3 ^2)^2)^2)`.

(this corresponds to *default* strategies in Maude) yielding $\varphi($_+_$) = (1\ 2\ 0)$, $\varphi($_-_$) = (1\ 2\ 0)$, $\varphi($_*_$) = (1\ 2\ 0)$, and $\varphi($_^2$) = (1\ 0)$. Note that this module `NAT` is terminating as a TRS (i.e., without any annotation).

- The benchmark `NAT_can` is similar to `NAT_eager`, but *canonical* positive strategies are provided: the i-th argument of a symbol f is annotated only 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]), i.e., we have $\varphi($_+_$) = (1\ 0)$, $\varphi($_-_$) = (1\ 2\ 0)$, $\varphi($_*_$) = (1\ 0)$, and $\varphi($_^2$) = (0)$.

- The benchmark `NAT_neg` is similar to `NAT_can`, though *canonical arbitrary* strategies are provided: now (from left-to-right), the i-th argument of a defined symbol f is annotated if all occurrences of f in the left-hand side of the rules contain a non-variable i-th argument; if all occurrences of f in the left-hand side of the rules have a variable i-th argument, then the argument is not annotated; in any other case, annotation $-i$ is given to f (see Remark 1 and [AL02]), i.e., we have $\varphi($_+_$) = (1\ 0)$, $\varphi($_-_$) = (1\ {-2}\ 0)$, $\varphi($_*_$) = (1\ 0)$, and $\varphi($_^2$) = (0)$.

- The benchmark `NAT_lazy` is similar to `NAT_neg`, but each positive annotation is replaced by its negative counterpart, i.e., we have $\varphi($_+_$) = (-1\ 0)$, $\varphi($_-_$) = (-1\ {-2}\ 0)$, $\varphi($_*_$) = (-1\ 0)$, and $\varphi($_^2$) = (0)$.

Then, for instance, the first term runs in less time when using program `NAT_neg` than using programs `NAT_eager` and `NAT_can`, which do not include negative annotations. For the second (more general) term, `NAT_eager` runs faster than other programs (because it does not replicate computations due to the equation of symbol _^2) but we can observe that `NAT_neg` does not significantly increase the runtimes w.r.t. `NAT_can`. Moreover, we can see that `NAT_lazy`, which uses only negative annotations, has similar runtimes to `NAT_can` and `NAT_neg` in CafeOBJ and it is even better in OnDemandOBJ.

9 Conclusions

We have provided a suitable extension of the positive evaluation strategy of OBJ-like languages to general (positive as well as negative) annotations. This 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 and 2). The main contributions of the paper are:

(a) the definition of a suitable and well-defined approach to demandedness via E-strategies (see Examples 4, 11, 13, 12, 14, 23, 31, and 34 for motivation regarding the inadequacy of the model in the previous proposals),

(b) the demonstration of the computational properties associated to the on-demand evaluation strategy (*ODE*) (Theorem 2, Corollary 1, and Theorem 4),

(c) the definition of techniques for analyzing termination under strategy annotations (Theorem 11),

(d) the experimental results of Section 8 which demonstrate that our approach is better suited for implementation.

We have shown that our on-demand evaluation strategy (*ODE*) improves the three most important evaluation strategies dealing with on-demand annotations, and we have also provided conditions for the equivalence of *ODE* w.r.t. the other three:

- on-demand evaluation with negative annotations of Nakamura and Ogata [NO01] as well as Ogata and Futatsugi [OF00] (Theorem 5),
- *lazy rewriting* (*LR*) [FKW00], a popular, demand-driven technique to perform lazy functional computations which inspired the development of on-demand strategies in OBJ (Theorems 6, 7, and 8), 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* (Theorems 9 and 10).

The reader might think that the computational model introduced by negative annotations is too complex, e.g. arguing that a program with such a negative (lazy) behavior is difficult to understand by somebody else than its author and thus, vulnerable to programming errors. However, we have (hopefully) proved that the computational model is very intuitive (up to the complexity associated to user-defined strategies) and automatically driven by the order among (positive and negative) annotations given by the user. Note that this was not achieved yet by previous proposals dealing with negative annotations.

Let us conclude by summarizing the *modus operandi*: Given a TRS $\mathcal{R}$ and a strategy map φ, if the positive indices in the strategy φ do not cover all the non-variable positions in the left-hand sides (this idea is captured by the notion of *canonical replacement map* [Luc98], see page 29 above), then we have to add positive or negative indices to recover completeness, i.e., we add indices until a canonical replacement map is obtained. This is exactly the problem shown in Example 3. Once completeness is ensured, if termination can be proved for the original program but it cannot be proved for the program annotated with only positive indices, then we must shift some positive annotations into negative ones, since they can preserve termination while achieving completeness for broader classes of programs. Where and why shall we use negative annotations?

- If for each occurrence of a symbol f in the left-hand side of the rules, the i-th argument contains a non-variable term, then f must be annotated with positive index i.
- If for each occurrence of a symbol f in the left-hand side of the rules, the i-th argument is a variable, then f must not be annotated with positive index i nor negative index $-i$.
- If the two previous cases do not apply, negative index $-i$ is given to f.

Alternatively, in many situations we can work without any positive index at all, using only negative indices (see Table 2 in Section 8 for some practical examples).

As future work, we plan to extend the program transformation developed in [AEL04], 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 M. Baaz and A. 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.

[AEL03] M. Alpuente, S. Escobar, and S. Lucas. OnDemandOBJ: a laboratory for strategy annotations. In J.L. Giavitto and P.E. Moreau, editors, *Proc. of the 4th International Workshop on Rule-Based Programming, RULE 2003*, volume 86.2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2003.

[AEL04] M. Alpuente, S. Escobar, and S. Lucas. Correct and complete (positive) strategy annotations for OBJ. In F. Gadducci and U. Montanari, editors, *Proc. of the 4th International Workshop on Rewriting Logic and its Applications, WRLA 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2004.

[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.

[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.

[CDE^{+}07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[DEL04] F. Durán, S. Escobar, and S. Lucas. On-demand evaluation for maude. In S. Abdennadher and C. Ringeissen, editors, *5th International Workshop on Rule-Based Programming, RULE'04*, volume 124 of *Electronic Notes in Theoretical Computer Science*, pages 25–39. Elsevier Sciences Publisher, 2004.

[Eke00] Steven 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.

[EMT05] Santiago Escobar, José Meseguer, and Prasanna Thati. Natural narrowing for general term rewriting systems. In J. Giesl, editor, *Proc. of 16th International Conference on Rewriting Techniques and Applications, RTA'05*, volume 3467 of *Lecture Notes in Computer Science*, pages 279–293. Springer-Verlag, Berlin, 2005.

[Esc03] S. Escobar. Refining weakly outermost-needed rewriting and narrowing. In D. Miller, editor, *Proc. of 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'03*, pages 113–123. ACM Press, New York, 2003.

[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.

[FR99] M.C.F. Ferreira and A.L. Ribeiro. Context-sensitive ac-rewriting. In P. Narendran and M. Rusinowitch, editors, *Proc. of 10th International Conference on Rewriting Techniques and Applications, RTA'99*, volume 1631 of *Lecture Notes in Computer Science*, pages 173–181. Springer-Verlag, Berlin, 1999.

[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.

[GM04] J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14:329–427, 2004.

[GTSKF04] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In V. van Oostromn, editor, *Proc. of 15th International Conference on Rewriting Techniques and Applications, RTA'04*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220. Springer-Verlag, Berlin, 2004.

[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*, pages 3–167. 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.

[Luc06] Salvador Lucas. Proving termination of context-sensitive rewriting by transformation. *Inf. Comput.*, 204(12):1782–1846, 2006.

[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.

[Nak02] M. Nakamura. Evaluation strategies for term rewriting systems. PhD thesis, School of Information Science, Japan Advanced Insitute of Science and Technology, 2002.

[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.

[Ohl02] E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, Berlin, 2002.

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

A Proofs

A.1 Proofs of Section 5

Theorem 1 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. Then, $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}{}^*_\varphi \langle s, \Lambda\rangle$ if and only if $\langle \|t\|, \Lambda\rangle \overset{\sharp}{\rightarrow}{}^*_{\|\varphi\|} \langle \|s\|, \Lambda\rangle$.*

Proof. We consider the more general situation of $p \in \mathcal{P}os_A(t)$ instead of Λ. Let us consider that $\langle t, p\rangle \overset{\sharp}{\rightarrow}{}^n_\varphi \langle s, p\rangle$ and $\langle \|t\|, p\rangle \overset{\sharp}{\rightarrow}{}^m_{\|\varphi\|} \langle s', p\rangle$. We prove $n \geq m$ and $s' = \|s\|$ by induction on n. If $n = 0$, then $t = s$, $m = 0$, and $s' = s$, since no step on $\|t\|$ can be done with $\|\varphi\|$ whenever no step on t can be done with φ. For $n > 0$, $\langle t, p\rangle \overset{\sharp}{\rightarrow}_\varphi \langle t', q\rangle \overset{\sharp}{\rightarrow}{}^{n-1}_\varphi \langle s, p\rangle$ and we consider different cases:

- If $t|_p = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i \neq 0$, and $root(\|t|_p\|)$ is not of the form $f_{L'_1|i:L'_2}$, then $i \in L_1$, since the occurrence in L_1 cancels out the current number i. Let $t' = t[f_{L_1 \oplus i|L_2}(t_1, \ldots, t_k)]_p$. If $i < 0$, then, by definition, $\langle t, p\rangle \overset{\sharp}{\rightarrow}_\varphi \langle t', p\rangle$. If $i > 0$, then $root(t|_{p.i}) = g_{L|nil}$, since subterm $t|_{p.i}$ has already been evaluated, and, by definition, $\langle t, p\rangle \overset{\sharp}{\rightarrow}_\varphi \langle t', p.i\rangle \overset{\sharp}{\rightarrow}_\varphi \langle t', p\rangle$. In both cases, $\|t\| = \|t'\|$ and no step on $\overset{\sharp}{\rightarrow}_{\|\varphi\|}$ is necessary. Finally, the conclusion follows by induction hypothesis.

- If $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, then $root(\|t|_p\|)$ is also of the form $f_{L'_1|0:L'_2}$. It is easy to see that $OD_{\mathcal{R}}(t|_p) = OD_{\mathcal{R}}(\|t|_p\|)$, since $\mathcal{P}os_P(t) = \mathcal{P}os_P(\|t\|)$, $\mathcal{P}os_N(t) = \mathcal{P}os_N(\|t\|)$, and $\mathcal{P}os_{nil}(t) = \mathcal{P}os_{nil}(\|t\|)$. Therefore, $\langle \|t\|, p\rangle \overset{\sharp}{\rightarrow}_{\|\varphi\|} \langle \|t'\|, q\rangle$ and the conclusion follows by induction hypothesis.

- For all the remaining cases, $\langle \|t\|, p\rangle \overset{\sharp}{\rightarrow}_{\|\varphi\|} \langle \|t'\|, q\rangle$ and by induction hypothesis $\langle \|t'\|, q\rangle \overset{\sharp}{\rightarrow}{}^{m-1}_{\|\varphi\|} \langle \|s\|, p\rangle$. □

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

Proof. Given $l \in L(\mathcal{R})$ such that $root(l) = root(erase(t))$, if $p \in DP_l(erase(t))$, then, since φ is alternating and $\lfloor\varphi\rfloor \in CM_{\mathcal{R}}$, either $p \in \mathcal{P}os_P(t)$ or $p \notin \mathcal{P}os_A(t)$. Thus, $ADP_l(t) = \varnothing$ for any $l \in L(\mathcal{R})$ such that $root(l) = root(erase(t))$ and $OD_{\mathcal{R}}(t) = \varnothing$. □

Theorem 3 *Let $\mathcal{R}$ be a TRS and φ be an alternating E-strategy map such that $\lfloor\varphi\rfloor \in CM_{\mathcal{R}}$. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, $p \in \mathcal{P}os_A(t)$, and $q \in \mathcal{P}os_A(s)$. Then, $\langle t, p\rangle \overset{\sharp}{\rightarrow}_\varphi \langle s, q\rangle$ if and only if $\langle \lfloor t\rfloor, p\rangle \overset{\sharp}{\rightarrow}{}^=_{\lfloor\varphi\rfloor} \langle \lfloor s\rfloor, q\rangle$.*

Proof. By Lemma 1, $OD_{\mathcal{R}}(t) = \varnothing$ and thus steps 6 and 7 of Definition 3 are never going to be used. We can remove all the negative annotations from φ, i.e., use $\lfloor\varphi\rfloor$ instead of φ. The steps performed by $\overset{\sharp}{\rightarrow}_{\varphi}$ consuming negative annotations are simply discarded by $\overset{\sharp}{\rightarrow}_{\lfloor\varphi\rfloor}$. □

Theorem 4 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $s \in eval^{\sharp}_{\varphi}(t)$, 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 $\langle f_{nil|(0)}, \Lambda\rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle f_{nil|nil}, \Lambda\rangle$. 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 $l \in L(\mathcal{R})$ such that $root(l) = f$. 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 $\langle t', \Lambda\rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle s', \Lambda\rangle$, $s = erase(s')$, $root(t') = f_{L|(0)}$ 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 (recall that $s = erase(s') = erase(t')$) is also not a redex and $\nexists l \in L(\mathcal{R})$ and $\sigma \in Subst(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ such that $s = \sigma(l)$. By left-linearity, there are disagreeing positions in s' ($Pos_{\neq}(s, l) \neq \varnothing$ for all $l \in L(\mathcal{R})$) that could be demanded but they are not ($OD_{\mathcal{R}}(s') = OD_{\mathcal{R}}(t') = \varnothing$). Note that if we drop left-linearity then it is possible that s is not a redex but there is $l \in L(\mathcal{R})$ such that $Pos_{\neq}(s, l) = \varnothing$. In the following, we prove that for each $l \in L(\mathcal{R})$ either $Pos_{\neq}(s, l) = \{\Lambda\}$, i.e., $root(l) \neq root(s)$, or $p \in Pos_{\neq}(s, l)$ and $s|_p$ is a head-normal form. And, by the CS property of the rules, for any position $p > \Lambda$ in s that is a head-normal form, no evaluation above it can rewrite the symbol at position p into the symbol expected at rule l, and thus we conclude s is a head-normal form.

We have that $OD_{\mathcal{R}}(s') = \varnothing$ implies $ADP_{\mathcal{R}}(s') = \varnothing$. If $ADP_{\mathcal{R}}(s') = \varnothing$, then either (i) there is no $l \in L(\mathcal{R})$ such that $root(s) = root(l)$, i.e., $Pos_{\neq}(s, l) = \{\Lambda\}$ for each $l \in L(\mathcal{R})$, or (ii) $ADP_l(s') = \varnothing$ for each $l \in L(\mathcal{R})$ such that $root(s) = root(l)$. In case (ii), we have that those rules which could be applied to s have a conflict of positions with s. If $ADP_l(s') = \varnothing$ for $l \in L(\mathcal{R})$ such that $root(s) = root(l)$, then either (iii) $DP_l(s) = \varnothing$, or (iv) $DP_l(s) \neq \varnothing$ and $DP_l(s) \cap \mathcal{P}os_A(s') = \varnothing$, (v) $DP_l(s) \neq \varnothing$ and $DP_l(s) \cap \mathcal{P}os_P(s') \neq \varnothing$, or (vi) $DP_l(s) \neq \varnothing$ and $DP_l(s) \cap \mathcal{P}os_{nil}(s') \neq \varnothing$. In case (iii), if $DP_l(s) = \varnothing$ but $Pos_{\neq}(s, l) \neq \varnothing \neq \{\Lambda\}$, then there is a position $p \in Pos_{\neq}(s, l)$ such that $p \in \mathcal{P}os_{\mathcal{C}}(s)$ and the conclusion follows. The case (iv) is not possible because $\varphi \in CM_{\mathcal{R}}$. In cases (v) and (vi) we have that there is a position $p \in DP_l(s)$ such that $p > \Lambda$ and the position is disagreeing with l and should be demanded and evaluated. However, by hypothesis, $s|_p$ is a head-normal form and the conclusion follows. □

A.2 Proofs of Section 6.1

Proposition 1 *Let $\mathcal{R}$ be a left-linear completely defined CS and φ be a standard E-strategy map in lexicographic order such that $\varphi \in CM_{\mathcal{R}}$. Let $\mathcal{R}$ be in φ-order. Let $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ such that $erase(t)$ is not a redex and $root(t) = f_{L|(0)}$. Let $p \in \mathcal{P}os_A(t) \cap \mathcal{P}os_{\mathcal{D}}(erase(t))$ such that $p > \Lambda$. Then, $\{p\} = OD_{\mathcal{R}}(t)$ if and only if $\{p\} = DF_{\mathcal{R}}(erase(t))$.*

Proof.
($\Rightarrow$) If $\{p\} = OD_{\mathcal{R}}(t)$, then $p \in ADP_{\mathcal{R}}(t)$ and p is the minimum position w.r.t. the total order $\leq_t$ ($p = min_{\leq_t}(ADP_{\mathcal{R}}(t))$). Since φ is in lexicographic order, p is also the minimum position w.r.t. the total order $\leq_{lex}$. Since $p \in ADP_{\mathcal{R}}(t)$, there is a lhs $l \in L(\mathcal{R})$ such that $root(erase(t)) = root(l)$ and $p \in ADP_l(t)$. And then, $p \in DP_l(erase(t))$ and $p \in minimal_{\leq}(\mathcal{P}os_{\neq}(erase(t), l))$.

Now, since $p \in minimal_{\leq}(\mathcal{P}os_{\neq}(erase(t), l))$ and $p = min_{\leq_t}(ADP_{\mathcal{R}}(t))$, we have that $p = df_l(erase(t)) = min_{\leq_{lex}}(\mathcal{P}os_{\neq}(erase(t), l))$. Then, for proving that $DF_{\mathcal{R}}(erase(t)) = \{p\} = max_{\leq_{lex}}(\{df_l(erase(t)) \mid l \to r \in \mathcal{R}\})$, we simply have to prove that for each $l' \in L(\mathcal{R})$ such that $l' \neq l$, it is impossible that $df_{l'}(erase(t)) = \{p'\}$ such that $p <_{lex} p'$. If such $p' = min_{\leq_{lex}}(\mathcal{P}os_{\neq}(erase(t), l'))$ exists, then $p \notin \mathcal{P}os_{\neq}(erase(t), l')$. But then, by the property that $\mathcal{R}$ is in φ-order, we have that $p \in \mathcal{P}os_{\mathcal{F}}(l)$ and, by the CS property, $p \in \mathcal{P}os_{\mathcal{C}}(erase(t))$, contradicting the assumption that $p \in \mathcal{P}os_{\mathcal{D}}(erase(t))$. Thus, $DF_{\mathcal{R}}(erase(t)) = \{p\}$.

($\Leftarrow$) If $p \in DF_{\mathcal{R}}(erase(t))$, then $p = max_{\leq_{lex}}(\{df_l(erase(t)) \mid l \to r \in \mathcal{R}\})$. Since $\mathcal{R}$ is completely defined, there exists $l \in L(\mathcal{R})$ such that $\mathcal{P}os_{\neq}(erase(t), l) \subseteq \mathcal{P}os_{\mathcal{D}}(erase(t))$. Thus, $p \in DF_l(erase(t))$. Since φ is standard and $\varphi \in CM_{\mathcal{R}}$, $\mathcal{P}os_P(t) \subseteq \mathcal{P}os_{nil}(t)$, i.e., all positive positions in t have been already evaluated. By Theorem 4 and since φ is standard, for each $p \in \mathcal{P}os_{nil}(t)$, $t|_p$ is a head-normal form and, by the completely defined property, $p \in \mathcal{P}os_{\mathcal{C}}(erase(t))$. Thus, by the CS property, $minimal_{\leq}(\mathcal{P}os_{\neq}(erase(t), l)) \cap (\mathcal{P}os_P(t) \cup \mathcal{P}os_{nil}(t)) = \varnothing$. Thus, $p \in ADP_l(t)$ and $p \in ADP_{\mathcal{R}}(t)$. Finally, since $\mathcal{R}$ is in φ-order, for each position $p' <_{lex} p$, we have that $p' \in \mathcal{P}os_{\mathcal{F}}(l)$ and, by the CS property, $p' \in \mathcal{P}os_{\mathcal{C}}(l)$. Moreover, for any other lhs $l' \in L(\mathcal{R})$ such that $p' \in \mathcal{P}os_{\neq}(erase(t), l') - \{\Lambda\}$, $p' \notin DP_{l'}(erase(t))$. Thus, $ODR_{\mathcal{R}}(t) = \{min_{\leq_t}(ADP_{\mathcal{R}}(t))\} = \{p\}$. □

Lemma 2 *Let $\mathcal{R}$ be a TRS and φ be a standard E-strategy map such that $\varphi \in CM_{\mathcal{R}}$. Let $t, s, t', s' \in \mathcal{T}(\mathcal{F}^{\mathbb{Z}}_{\varphi}, \mathcal{X}^{\mathbb{Z}}_{\varphi})$ such that t' and s' have more symbols with the on-demand flag activated than t and s, respectively. Then, $\langle t, p\rangle \overset{\mathbb{Z}}{\to}_{\varphi} \langle s, q\rangle$ if and only $\langle t', p\rangle \overset{\mathbb{Z}}{\to}_{\varphi} \langle s', q\rangle$.*

Proof. The only case in Definition 2 affected is case 4(c), where $t|_p = f^b_{0:L}(t_1, \ldots, t_k)$, and $DF_{\mathcal{R}}(erase(t|_p)) = p' \neq \top$ and $flag(t, p.p') = 0$. However, this case is impossible, since $\varphi \in CM_{\mathcal{R}}$ and φ is standard. That is, φ being standard implies that the only index 0 is at the end of any strategy list and thus every possible on-demand flag has been activated before index 0 at position p in t is reached. And the property $\varphi \in CM_{\mathcal{R}}$ implies that every non-variable position in any lhs is covered by a positive or negative position. □

Theorem 5 *Let $\mathcal{R}$ be a left-linear completely defined CS and φ be a standard E-strategy map in lexicographic order such that $\varphi \in CM_{\mathcal{R}}$. Let $\mathcal{R}$ be in φ-order. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. Then, $\langle t, \Lambda\rangle \overset{\sharp}{\to}{}^!_\varphi \langle s, \Lambda\rangle$ if and only if $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda\rangle$.*

Proof. First note that, by Lemma 2, it is not a problem to add more on-demand flags than the necessary and thus, we can use $\lfloor t \rfloor_{\mathbb{Z}}$ without loosing any $\overset{\mathbb{Z}}{\to}_\varphi$-step.

($\Rightarrow$) By induction on the number n of steps of $\langle t, \Lambda\rangle \overset{\sharp}{\to}{}^n_\varphi \langle s, \Lambda\rangle$.

1. If $n = 0$, then $s = t$ and $root(t) = f_{L|nil}$. Thus, $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{nil}$ and the conclusion follows.

2. If $n > 0$, then we consider the first step $\langle t, \Lambda\rangle \overset{\sharp}{\to}_\varphi \langle t', p\rangle \overset{\sharp}{\to}{}^!_\varphi \langle s, \Lambda\rangle$ and all its possible cases:

   (a) If $t = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $t' = f_{L_1 \oplus i|L_2}(t_1, \ldots, t_k)$ and $p = i$, then $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{i:L_2}(\lfloor t_1 \rfloor_{\mathbb{Z}}, \ldots, \lfloor t_k \rfloor_{\mathbb{Z}})$ and thus $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}_\varphi \langle \lfloor t' \rfloor_{\mathbb{Z}}, p\rangle$. We are in the following situation

   $$\langle t, \Lambda\rangle \overset{\sharp}{\to}_\varphi \langle t', p\rangle \overset{\sharp}{\to}{}^{n'}_\varphi \langle t'', p\rangle \overset{\sharp}{\to}_\varphi \langle s', \Lambda\rangle \overset{\sharp}{\to}{}^!_\varphi \langle s, \Lambda\rangle$$

   where $\langle t'|_p, \Lambda\rangle \overset{\sharp}{\to}{}^!_\varphi \langle t''|_p, \Lambda\rangle$ in n' steps and $n' < n$. Then, by induction hypothesis, $\langle \lfloor t'|_p \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor t''|_p \rfloor_{\mathbb{Z}}, \Lambda\rangle$ and $\langle \lfloor s' \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda\rangle$, thus we conclude

   $$\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}_\varphi \langle \lfloor t' \rfloor_{\mathbb{Z}}, p\rangle \overset{\mathbb{Z}}{\to}{}^*_\varphi \langle \lfloor t'' \rfloor_{\mathbb{Z}}, p\rangle \overset{\mathbb{Z}}{\to}_\varphi \langle \lfloor s' \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda\rangle.$$

   (b) If $t = f_{L_1|-i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $t' = f_{L_1 @ -i|L_2}(t_1, \ldots, t_k)$ and $p = \Lambda$, then, since $\lfloor t' \rfloor_{\mathbb{Z}}$ raises all on-demand flags at the same time, $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}_\varphi \langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda\rangle$ and, by induction hypothesis, $\langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda\rangle$.

   (c) If $t = f_{L_1|0:L_2}(t_1, \ldots, t_k) = \sigma(l')$, $erase(l') = l$, $t' = \sigma(\varphi(r))$ for some $l \to r \in R$ and substitution σ, and $p = \Lambda$, then $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}_\varphi \langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda\rangle$ and, by induction hypothesis, $\langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda\rangle \overset{\mathbb{Z}}{\to}{}^!_\varphi \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda\rangle$.

(d) If $t = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $\mathit{erase}(t)$ is not a redex, $OD_{\mathcal{R}}(t) = \varnothing$, $t' = f_{L_1|L_2}(t_1, \ldots, t_k)$, and $p = \Lambda$, then, since $\varphi \in CM_{\mathcal{R}}$ and $\mathcal{R}$ is completely defined, we have that $\mathit{root}(t) \in \mathcal{C}$ and $DF_{\mathcal{R}}(\mathit{erase}(t)) = \Lambda$. Thus, $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}_{\varphi} \langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda \rangle$ and, by induction hypothesis, $\langle \lfloor t' \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$.

(e) If $t = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $\mathit{erase}(t)$ is not a redex, $OD_{\mathcal{R}}(t) = \{p\}$, and $t' = \mathit{mark}(t, p)$, then, by Proposition 1, $DF_{\mathcal{R}}(\mathit{erase}(t)) = p$ and, by definition, $\mathit{flag}(\lfloor t \rfloor_{\mathbb{Z}}, p) = 1$. Thus, a recursive subsequence for $t|_p$ is going to be started. We are in the following situation

$$\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', p \rangle \overset{\sharp}{\rightarrow}^{n'}_{\varphi} \langle t'', p \rangle \overset{\sharp}{\rightarrow}^{*}_{\varphi} \langle s', \Lambda \rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi} \langle s, \Lambda \rangle$$

where $\langle t'|_p, \Lambda \rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi} \langle t''|_p, \Lambda \rangle$ in n' steps and $n' < n$. By induction hypothesis, $\langle \lfloor t'|_p \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor t''|_p \rfloor_{\mathbb{Z}}, \Lambda \rangle$. By Theorem 4, $\mathit{erase}(t''|_p)$ is a head-normal form. By the property $\mathcal{R}$ is completely defined, $p \in \mathcal{P}os_{\mathcal{C}}(\mathit{erase}(t''))$. And, since $\mathcal{R}$ is completely defined, $p \neq DF_{\mathcal{R}}(\mathit{erase}(t''))$. Thus, we are in case 4(d) of Definition 2 and, since $\mathit{erase}(s') = \mathit{erase}(t'')$ and the steps $\langle t'', p \rangle \overset{\sharp}{\rightarrow}^{*}_{\varphi} \langle s', \Lambda \rangle$ only remove bars above symbols, $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}_{\varphi} \langle \lfloor s' \rfloor_{\mathbb{Z}}, \Lambda \rangle$. Then, by induction hypothesis, $\langle \lfloor s' \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$.

(f) We cannot have the case where $t = \overline{f}_{L_1|L_2}(t_1, \ldots, t_k)$.

($\Leftarrow$) By induction on the number n of steps of this sequence $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$ plus the steps of all the subsequences (and their subsequences) demanded inside this sequence. Note that, since $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$ is finite, we know there is a finite number of recursive subsequences inside other subsequences and thus n is finite.

1. If $n = 0$, then $\lfloor s \rfloor_{\mathbb{Z}} = \lfloor t \rfloor_{\mathbb{Z}}$ and $\mathit{root}(\lfloor t \rfloor_{\mathbb{Z}}) = f^1_{nil}$. Thus, $t = f_{L|nil}$ and the conclusion follows.

2. If $n > 0$, then we consider the first step $\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}_{\varphi} \langle \lfloor t' \rfloor_{\mathbb{Z}}, p \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$ and all its possible cases:

   (a) If $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{i:L}(t_1, \ldots, t_k)$, $i > 0$, $\lfloor t' \rfloor_{\mathbb{Z}} = f^1_L(t_1, \ldots, t_k)$ and $p = i$, then $\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', p \rangle$. We are in the following situation

$$\langle \lfloor t \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}_{\varphi} \langle \lfloor t' \rfloor_{\mathbb{Z}}, p \rangle \overset{\mathbb{Z}}{\rightarrow}^{n'}_{\varphi} \langle \lfloor t'' \rfloor_{\mathbb{Z}}, p \rangle \overset{\mathbb{Z}}{\rightarrow}_{\varphi} \langle \lfloor s' \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor s \rfloor_{\mathbb{Z}}, \Lambda \rangle$$

   where $\langle \lfloor t'|_p \rfloor_{\mathbb{Z}}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^{!}_{\varphi} \langle \lfloor t''|_p \rfloor_{\mathbb{Z}}, \Lambda \rangle$ in n' steps and $n' < n$. By induction hypothesis, $\langle t'|_p, \Lambda \rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi} \langle t''|_p, \Lambda \rangle$ and $\langle s', \Lambda \rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi} \langle s, \Lambda \rangle$. Thus,

$$\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', p \rangle \overset{\sharp}{\rightarrow}^{*}_{\varphi} \langle t'', p \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle s', \Lambda \rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi} \langle s, \Lambda \rangle$$

(b) If $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{-i:L}(t_1, \ldots, t_k)$, $i > 0$, $\lfloor t' \rfloor_{\mathbb{Z}} = f^1_L(t_1, \ldots, up(t_i), \ldots, t_k) = f^1_L(t_1, \ldots, t_i, \ldots, t_k)$ and $p = \Lambda$, then $\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', \Lambda \rangle$ and, by induction hypothesis, $\langle t', \Lambda \rangle \overset{\sharp}{\rightarrow}^!_{\varphi} \langle s, \Lambda \rangle$.

(c) If $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{0:L}(t_1, \ldots, t_k)$, $p = \Lambda$, $\lfloor t' \rfloor_{\mathbb{Z}} = \theta(\varphi(r))$, $DF_{\mathcal{R}}(erase(t)) = \top$, $\lfloor t \rfloor_{\mathbb{Z}} = \theta(l')$, $erase(l') = l$ and $l \to r \in \mathcal{R}$, then $\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', \Lambda \rangle$ and, by induction hypothesis, $\langle t', \Lambda \rangle \overset{\sharp}{\rightarrow}^!_{\varphi} \langle s, \Lambda \rangle$.

(d) If $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{0:L}(t_1, \ldots, t_k)$, $p = \Lambda$, $\lfloor t' \rfloor_{\mathbb{Z}} = f^1_L(t_1, \ldots, t_k)$, and $DF_{\mathcal{R}}(erase(t)) = \Lambda$, then $root(erase(t)) \in \mathcal{C}$. Note that the case where $DF_{\mathcal{R}}(erase(t)) = \top$ and $erase(t)$ is not a redex is not possible because $\mathcal{R}$ is left-linear and the case where $DF_{\mathcal{R}}(erase(t)) = p' \neq \Lambda$ and $flag(t, p.p') = 0$ is not possible by definition of $\lfloor \_ \rfloor_{\mathbb{Z}}$. Thus $OD_{\mathcal{R}}(t) = \varnothing$ and $\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', \Lambda \rangle$. By induction hypothesis, $\langle t', \Lambda \rangle \overset{\sharp}{\rightarrow}^!_{\varphi} \langle s, \Lambda \rangle$.

(e) The case where $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{0:L}(t_1, \ldots, t_k)$, $p = \Lambda$, $DF_{\mathcal{R}}(erase(t)) = p' = i.p''$ for $i \in \mathbb{N}$, $flag(\lfloor t \rfloor_{\mathbb{Z}}, p') = 1$, $\langle dn(\lfloor t \rfloor_{\mathbb{Z}}|_{p'}), \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^!_{\varphi} \langle w, \Lambda \rangle$, $DF_{\mathcal{R}}(erase(\lfloor t \rfloor_{\mathbb{Z}}[w]_{p'})) = p'$, and $t'' = f^1_L(t_1, \ldots, t_i[up(w)]_{p''}, \ldots, t_k)$ is not possible by Proposition 1 and the property that $\mathcal{R}$ is completely defined.

(f) If $\lfloor t \rfloor_{\mathbb{Z}} = f^1_{0:L}(t_1, \ldots, t_k)$, $p = \Lambda$, $DF_{\mathcal{R}}(erase(t)) = p' \neq \Lambda$, $flag(\lfloor t \rfloor_{\mathbb{Z}}, p') = 1$, $\langle dn(\lfloor t \rfloor_{\mathbb{Z}}|_{p'}), \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^!_{\varphi} \langle w, \Lambda \rangle$, $\lfloor t' \rfloor_{\mathbb{Z}} = t[up(w)]_{p'}$, and either $p' <_{lex} DF_{\mathcal{R}}(erase(t[w]_{p'}))$ or $DF_{\mathcal{R}}(erase(t[s]_{p'})) = \top$, then $p' \in \mathcal{P}os_{\mathcal{D}}(erase(t))$ and, by Proposition 1, $OD_{\mathcal{R}}(t) = \{p'\}$. By induction hypothesis, $\langle t|_{p'}, \Lambda \rangle \overset{\mathbb{Z}}{\rightarrow}^!_{\varphi} \langle t'', \Lambda \rangle$ such that $w = \lfloor t'' \rfloor_{\mathbb{Z}}$. The steps $\langle t'', p \rangle \overset{\sharp}{\rightarrow}^*_{\varphi} \langle s', \Lambda \rangle$ remove just bars above symbols, i.e., $erase(t'') = erase(s')$. Also by induction hypothesis, $\langle s', \Lambda \rangle \overset{\sharp}{\rightarrow}^!_{\varphi} \langle s, \Lambda \rangle$. And thus,

$$\langle t, \Lambda \rangle \overset{\sharp}{\rightarrow}_{\varphi} \langle t', p \rangle \overset{\sharp}{\rightarrow}^*_{\varphi} \langle t'', p \rangle \overset{\sharp}{\rightarrow}^*_{\varphi} \langle s', \Lambda \rangle \overset{\sharp}{\rightarrow}^!_{\varphi} \langle s, \Lambda \rangle$$

□

A.3 Proofs of Section 6.2

Lemma 3 *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 (i) $\exists l \in L(\mathcal{R})$ such that l matches modulo laziness $label_{\mu_{\lfloor \varphi \rfloor}}(erase(t))$, (ii) $root(erase(t)|_p) \neq root(l|_p) \notin \mathcal{X}$, (iii) for all p' such that $\Lambda \leq p' < p$, $root(erase(t)|_{p'}) = root(l|_{p'})$, and (iv) at least one position $p' : \Lambda < p' \leq p$ is declared essential.*

Proof. If $OD_{\mathcal{R}}(t) = \{p\}$, then 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' s.t. $\Lambda \leq p' < p$, $root(erase(t)|_{p'}) = root(l|_{p'})$. Also, $ADP_l(erase(t)) \cap$

$\mathcal{P}os_P(t) = \varnothing$ and $ADP_l(erase(t)) \cap \mathcal{P}os_{nil}(t) = \varnothing$. That is, for all $q \in DP_l(erase(t))$, $q \notin \mathcal{P}os_P(t)$ and then, there is $q' \leq q$ such that $q' \in \mathcal{P}os_A(t)$, $root(label_{\mu_{\lfloor\varphi\rfloor}}(erase(t))|_{q'}) = f^{\ell}$ and for all q'' such that $\Lambda \leq q'' < q'$, symbol $root(label_{\mu_{\lfloor\varphi\rfloor}}(erase(t))|_{q''})$ is marked as eager. Hence, the conclusion follows and l matches modulo laziness t. $\square$

Lemma 4 *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 $t^{\varphi} = label_{\mu_{\lfloor\varphi\rfloor}}(erase(t))$, there exists θ for* LR *such that $label_{\mu_{\lfloor\varphi\rfloor}}(erase(r')) = \theta(label_{\mu_{\lfloor\varphi\rfloor}}(r))$.*

Proof. Note that all variables of l' have the same labeling, 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, $\theta(label_{\mu_{\lfloor\varphi\rfloor}}(r)) = label_{\mu_{\lfloor\varphi\rfloor}}(erase(\sigma(r)))$, i.e. $label_{\mu_{\lfloor\varphi\rfloor}}(erase(r')) = \theta(label_{\mu_{\lfloor\varphi\rfloor}}(r))$. $\square$

Theorem 6 *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 $\mu = \mu_{\lfloor\varphi\rfloor}$. If $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi} \langle s, q\rangle$ 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 $lazy^{p}_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^{p}_{\varphi}(t)$ and $lazy^{q}_{\varphi}(s) \leq_{lazy} lazy^{p}_{\varphi}(t)$. 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 $i > 0$, $s = t[f_{L_1 \oplus i|L_2}(t_1, \ldots, t_k)]_p$ and $q = p.i$, then $lazy^{p}_{\varphi}(t) = lazy^{q}_{\varphi}(s)$, since $p, q \in \mathcal{P}os_P(t)$, and, $q \in \mathcal{A}ct(lazy^{q}_{\varphi}(s))$, since $i \in \mu(f)$. Thus, we have $lazy^{p}_{\varphi}(t) \xrightarrow{\mathsf{LR}}{}^{=}_{\mu} lazy^{q}_{\varphi}(s)$.

3. If $t|_p = f_{L_1|-i:L_2}(t_1, \ldots, t_k)$ with $i > 0$, $s = t[f_{L_1 \oplus -i|L_2}(t_1, \ldots, t_k)]_p$ 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 4, 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 4, 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'' s.t. $p \leq p'' \leq p.p'$, $root(lazy^q_\varphi(s)|_{p''}) = f^e$ and $q \in \mathcal{A}ct(lazy^q_\varphi(s))$. Thus, the only difference between $lazy^p_\varphi(t)$ and $lazy^q_\varphi(s)$ is the activated set of positions between p and $p.p'$, i.e., for all p'' s.t. $p < p'' \leq p.p'$, $root(lazy^q_\varphi(s)|_{p''}) = f^e$ and $root(lazy^p_\varphi(t)|_{p''}) = f^\ell$. Moreover, we know that there is a lhs $l \in L(\mathcal{R})$ such that for all p'' s.t. $\Lambda < p'' \leq p'$, $root(l|_{p''}) \in \mathcal{F}$. Thus, by successive applications of Lemma 3, there are several activation steps of lazy rewriting and 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 $q.i \in \mathcal{A}ct(lazy^p_\varphi(t))$. □

Theorem 7 *Let $\mathcal{R}$ be a left-linear completely defined CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. If $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\langle \varphi(t), \Lambda \rangle \overset{\sharp}{\to}{}^{!}_{\varphi} \langle s, \Lambda \rangle$, then $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^{!}_{\mu} lazy^\Lambda_\varphi(s)$.*

Proof. By Theorem 6, there is a sequence $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^{*}_{\mu} s'$ such that $lazy^\Lambda_\varphi(s) \leq_{lazy} s'$. By Theorem 4, $erase(s') = erase(s)$ is a head-normal form. And by the property $\mathcal{R}$ is completely defined, $root(erase(s)) \in \mathcal{C} \cup \mathcal{X}$. If $erase(s) = erase(s') = x \in \mathcal{X}$, then it is clear that $s' = lazy^\Lambda_\varphi(s)$. Otherwise, since all positive positions have been evaluated in s' and a constructor symbol cannot activate any position, we can speak of normalization instead of an arbitrary number of steps, i.e., $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^{!}_{\mu} s'$ and $s' = lazy^\Lambda_\varphi(s)$. □

Theorem 8 *Let $\mathcal{R}$ be a left-linear completely-defined CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$ and $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$. Let $\mathcal{R}$ be $\overset{\sharp}{\to}_{\varphi}$-terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\langle \varphi(t), \Lambda \rangle \overset{\sharp}{\to}{}^{!}_{\varphi} \langle s, \Lambda \rangle$ if and only if $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^{!}_{\mu} lazy^\Lambda_\varphi(s)$.*

Proof. We only have to prove the $\Leftarrow$ implication, since the $\Rightarrow$ implication is proved by Theorem 7. We have $lazy^\Lambda_\varphi(t) \xrightarrow{\mathsf{LR}}{}^{!}_{\mu} w$ for some term w. Since $\mathcal{R}$ is a left-linear CS, $erase(w)$ is a head-normal form. Since $\mathcal{R}$ is completely defined, $root(erase(w)) \in \mathcal{C} \cup \mathcal{X}$. Then, since $\mathcal{R}$ is $\overset{\sharp}{\to}_{\varphi}$-terminating, we can take any sequence $\langle \varphi(t), \Lambda \rangle \overset{\sharp}{\to}{}^{!}_{\varphi} \langle s, \Lambda \rangle$ for some term s. By Theorem 4, s is a head-normal form and, since $\mathcal{R}$ is completely defined, $root(erase(s)) \in \mathcal{C} \cup \mathcal{X}$. Finally, by Theorem 7, $w = lazy^\Lambda_\varphi(s)$. □

A.4 Proofs of Section 6.3

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 $\langle t, p\rangle \overset{\sharp}{\to}{}^*_\varphi \langle t', q\rangle$ such that $p \leq q$, $erase(t) = erase(t')$, and $erase(t'|_q)$ is a redex.

Lemma 5 *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map such that $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ 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 $t|_p$ is a $\overset{\sharp}{\to}_\varphi$-normal form. □

Lemma 6 *Let $\mathcal{R}$ be a left-linear CS and φ be an E-strategy map such that $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ 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 for all w s.t. $q < w \leq p$, w is a stop position.*

Proof. By induction on the length n of the evaluation sequence $\langle\varphi(s), \Lambda\rangle \overset{\sharp}{\to}{}^n_\varphi \langle t, p\rangle$ 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 $\langle\varphi(s), \Lambda\rangle \overset{\sharp}{\to}{}^{n-1}_\varphi \langle t', p'\rangle \overset{\sharp}{\to}_\varphi \langle t, p\rangle$. 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 $t'|_{p'} = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $t = t'[f_{L_1 \oplus i|L_2}(t_1, \ldots, t_k)]_{p'}$ 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 5, 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 9 *Let $\mathcal{R}$ be a left-linear CS and φ be an E-strategy map such that $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $\mu, \mu_D \in M_{\mathcal{F}}$ be such that $\mu = \mu_{\lfloor\varphi\rfloor}$ 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 $\langle t, p\rangle \overset{\sharp}{\to}_\varphi \langle s, q\rangle$ and $\langle t, p\rangle$ is consistent, then $erase(t) \overset{p}{\hookrightarrow}{}^{=}_{\langle\mu,\mu_D\rangle} erase(s)$.*

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 6, 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 $t' = erase(t)[r_1]_{p_1}\cdots[r_n]_{p_n}$, $t'|_q = \sigma(l)$ and, for all $w \in \mathcal{P}os(l)$ s.t. $sprefix_{erase(t)|_q}(w) = sprefix_l(w)$ 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 $erase(t|_q)$ 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 $\mathcal{P}os_{\neq}(erase(t|_q), l) \neq \varnothing$ 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 $p_1, \ldots, p_n \in \mathcal{L}azy_{\langle\mu,\mu_D\rangle}(erase(t))$, $r_1, \ldots, r_n, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \to r \in R$, and substitution σ such that $t' = erase(t)[r_1]_{p_1} \cdots [r_n]_{p_n}$, $t'|_q = \sigma(l)$ and for all $w \in \mathcal{P}os(l)$ s.t. $sprefix_{erase(t)|_q}(w) = sprefix_l(w)$, whenever $q.w \leq p$, we have that $l|_w \notin \mathcal{X}$. $\square$

Theorem 10 *Let $\mathcal{R}$ be a left-linear completely-defined CS and φ be an E-strategy map such that $\varphi \in CM_{\mathcal{R}}$, $\varphi(f)$ ends with 0 for all $f \in \mathcal{D}$, and $\mu_{\lfloor\varphi\rfloor}(c) = \varnothing$ for $c \in \mathcal{C}$. Let $\mathcal{R}$ be $\overset{\sharp}{\to}_{\varphi}$-terminating. Let $\mu, \mu_D \in M_{\mathcal{F}}$ be such that $\mu = \mu_{\lfloor\varphi\rfloor}$ and $\mu \sqcup \mu_D = \mu_{\varphi}$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\langle\varphi(t), \Lambda\rangle \overset{\sharp}{\to}^!_{\varphi} \langle s, \Lambda\rangle$ if and only if $t \hookrightarrow^!_{\langle\mu,\mu_D\rangle} erase(s)$.*

Proof. We only have to prove the $\Leftarrow$ implication, since the $\Rightarrow$ implication is proved by Theorem 9. The proof for the $\Rightarrow$ implication is similar to Theorem 8. $\square$

A.5 Proofs of Section 7

In the following, we write $\overset{\sharp}{\to}_{\varphi,\mathcal{R}}$ instead of $\overset{\sharp}{\to}_{\varphi}$ to denote that the TRS $\mathcal{R}$ is used.

Proposition 2 *Let $\mathcal{R}$ be a CS and φ be a standard E-strategy map. Let $l \to r \in R$ such that $NegPos(l) = \{p'.j\}$, $f = root(l|_{p'})$, and $l' = l[f_j(l|_{p'.1}, \ldots, l|_{p'.ar(f)})]_{p'}$ for the new symbol f_j. Let $\mathcal{R}^\diamond$ be $\overset{\sharp}{\to}_{\varphi^\diamond}$-terminating. Let $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os(t)$. If $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q\rangle$, then $\langle t, p\rangle \overset{\sharp}{\to}^+_{\varphi^\diamond,\mathcal{R}^\diamond} \langle s', q\rangle$ such that either (i) s' and s differ only in some positions $Q \in \mathcal{P}os(s') \cap \mathcal{P}os(s)$ such that for each $w \in Q$, $\langle s|_w, \Lambda\rangle \overset{\sharp}{\to}^!_{\varphi,\mathcal{R}} \langle s'|_w, \Lambda\rangle$ or (ii) there exists a position $q' \in \mathcal{P}os(s) \cap \mathcal{P}os(s')$ such that $q > \Lambda$ and $erase(s)$ and $erase(s')$ differ only in the symbol at position q'.*

Proof. We can prove that $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q\rangle$ implies $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi^\diamond,\mathcal{R}^\diamond} \langle s, q\rangle$ except in the following three cases, where we have that $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q\rangle$ implies $\langle t, p\rangle \overset{\sharp}{\to}_{\varphi^\diamond,\mathcal{R}^\diamond} \langle s', q\rangle$:

1. When $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k) = \sigma(l')$ for some term l' such that $erase(l') = l$. In this case, we have that

$$\langle t, p\rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q\rangle$$

with $s = t[\sigma(\varphi(r))]_p$ for some substitution σ and $q = p$. For $\mathcal{R}^\diamond$ however, we apply rule $l[\overline{x}]_P \to l'[\overline{x}]_P$ instead for $P = \mathit{minimal}_\leq(\{p'' \in \mathcal{P}os_A(l) - \mathcal{P}os_P(l) \mid p'.j \leq_{\varphi(l)} p''\})$, and obtain

$$\langle t, p\rangle \overset{\sharp}{\to}_{\varphi^\diamond,\mathcal{R}^\diamond} \langle t', p\rangle$$

for some t' such that *erase*(t') and *erase*(t) differ only in the symbol at position $p.p'$ and annotations have been reset in t' w.r.t. t. By definition, every position $p'' <_t p.p'.j$ is positive. By the property φ is standard, every position p'' is evaluated before position $p.p'.j$ w.r.t. φ and $\mathcal{R}$. By the property $\mathcal{R}^\diamond$ is $\overset{\sharp}{\to}_{\varphi^\diamond}$-terminating, we have that

$$\langle t', p\rangle \overset{\sharp}{\to}^*_{\varphi^\diamond,\mathcal{R}^\diamond} \langle t'', p.p'.j\rangle$$

such that t'' and t only differ in the symbol at position $p.p'$, i.e., all annotations in the lists of symbols in t'' coincide with t, except the symbol at position $p.p'$, which is f_j. Then, since $\mathcal{R}$ is a CS and $t|_p = \sigma(l')$, $root(t|_{p.p'.j}$ is a constructor symbol and thus, since $\mathcal{R}$ is $\overset{\sharp}{\to}_{\varphi^\diamond}$-terminating,

$$\langle t'', p.p'.j\rangle \overset{\sharp}{\to}^*_{\varphi^\diamond,\mathcal{R}^\diamond} \langle t''', p\rangle$$

such that the rule $l' \to r$ can be applied

$$\langle t'', p\rangle \overset{\sharp}{\to}_{\varphi^\diamond,\mathcal{R}^\diamond} \langle s', p\rangle.$$

Note that s' and s differ only in some positions $Q \in \mathcal{P}os(s') \cap \mathcal{P}os(s)$ such that for each $w \in Q$, $\langle s|_w, \Lambda\rangle \overset{\sharp}{\to}^!_{\varphi,\mathcal{R}} \langle s'|_w, \Lambda\rangle$.

2. When $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, *erase*$(t|_p)$ is not a redex, and $OD_{\mathcal{R}}(t|_p) = \{p'.j\}$. In this case, we have that

$$\langle t, p\rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q\rangle$$

with $s = t[mark(t|_p, p'.j)]_p$ and $q = p.p'.j$. For $\mathcal{R}^\diamond$ however, we apply rule $l[\overline{x}]_P \to l'[\overline{x}]_P$ and obtain

$$\langle t, p\rangle \overset{\sharp}{\to}_{\varphi^\diamond,\mathcal{R}^\diamond} \langle t', p\rangle$$

for some t' such that *erase*(t') and *erase*(t) differ only in the symbol at position $p.p'$ and annotations have been reset in t' w.r.t. t. Then, like in case 1, we have that

$$\langle t', p\rangle \overset{\sharp}{\to}^*_{\varphi^\diamond,\mathcal{R}^\diamond} \langle t'', p.p'.j\rangle$$

such that t'' and t only differ in the symbol at position $p.p'$, i.e., all annotations in the lists of symbols in t'' coincide with t, except the symbol at position $p.p'$, which is f_j.

3. When $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $erase(t|_p)$ is not a redex, and $OD_{\mathcal{R}}(t|_p) = \varnothing$ such that $p.p'.j \in minimal_{\leq}(\mathcal{P}os_{\neq}(erase(t|_p), l))$ and either $p.p'.j \in \mathcal{P}os_{\mathcal{C}}(erase(t|_p))$ or $p.p'.j \in \mathcal{P}os_{nil}(erase(t|_p))$. In such case, we have that
$$\langle t, p' \rangle \overset{\sharp}{\to}_{\varphi,\mathcal{R}} \langle s, q \rangle$$
with $s = t[f_{L_1|L_2}(t_1, \ldots, t_k)]_p$, and $q = p$. For $\mathcal{R}^{\diamond}$ however, we apply rule $l[\overline{x}]_P \to l'[\overline{x}]_P$ and obtain
$$\langle t, p \rangle \overset{\sharp}{\to}_{\varphi^{\diamond},\mathcal{R}^{\diamond}} \langle t', p \rangle$$
for some t' such that $erase(t')$ and $erase(t)$ differ only in the symbol at position $p.p'$ and annotations have been reset in t' w.r.t. t. However, note that annotations have not been modified for symbol at position $p.p'.j$. In any case $p.p'.j \in \mathcal{P}os_{\mathcal{C}}(erase(t|_p))$ or $p.p'.j \in \mathcal{P}os_{nil}(erase(t|_p))$, then, like in case 1, we have that
$$\langle t', p \rangle \overset{\sharp}{\to}{}^{*}_{\varphi^{\diamond},\mathcal{R}^{\diamond}} \langle t'', p.p'.j \rangle \overset{\sharp}{\to}{}^{*}_{\varphi^{\diamond},\mathcal{R}^{\diamond}} \langle t''', p \rangle$$
such that t''' and t only differ in the symbol at position $p.p'$, i.e., all annotations in the lists of symbols in t'' coincide with t, except the symbol at position $p.p'$, which is f_j. In the case $p.p'.j \in \mathcal{P}os_{\mathcal{C}}(erase(t|_p))$, rule $l' \to r$ cannot be applied by the conflict between symbols $root(t'''|_{p.p'.j})$ and $root(l|_{p'.j})$ and then the conclusion follows. In the case $p.p'.j \in \mathcal{P}os_{nil}(erase(t|_p))$, rule $l' \to r$ cannot be applied because the symbol $root(t'''|_{p.p'.j})$ has not changed, and then the conclusion follows. □

Theorem 12 *Let $\mathcal{R}$ be a CS and φ be a standard E-strategy map. Let $l \to r \in R$ such that $NegPos(l) = \{p'.j\}$, $f = root(l|_{p'})$, and $l' = l[f_j(l|_{p'.1}, \ldots, l|_{p'.ar(f)})]_{p'}$ for the new symbol f_j. If the relation $\overset{\sharp}{\to}_{\varphi^{\diamond},\mathcal{R}^{\diamond}}$ is terminating, then the relation $\overset{\sharp}{\to}_{\varphi,\mathcal{R}}$ is also terminating.*

Proof. By Proposition 2, we can associate a sequence $\langle t, p \rangle \overset{\sharp}{\to}{}^{n'}_{\varphi^{\diamond},\mathcal{R}^{\diamond}} \langle s', q \rangle$ to each sequence $\langle t, p \rangle \overset{\sharp}{\to}{}^{n}_{\varphi,\mathcal{R}} \langle s, q \rangle$ such that $n' \geq n$. Then, the conclusion follows. □

Theorem 11 *Let $\mathcal{R}$ be a CS and φ be a standard E-strategy map. If the relation $\overset{\sharp}{\to}_{\varphi^{\natural},\mathcal{R}^{\natural}}$ is terminating, then the relation $\overset{\sharp}{\to}_{\varphi,\mathcal{R}}$ is also terminating.*

Proof. By Theorem 12 and induction on the number n of transformation steps $\langle \mathcal{R}_0, \varphi_0 \rangle, \ldots, \langle \mathcal{R}_n, \varphi_n \rangle$, where $\mathcal{R}_0 = \mathcal{R}$, $\varphi_0 = \varphi$, and $\mathcal{R}_n = \mathcal{R}^{\natural}$, $\varphi_n = \varphi^{\natural}$. □