Electronic Notes in Theoretical Computer Science 86 No. 2 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume86.html 27 pages

On-demand Evaluation by Program Transformation [1]

María Alpuente [2] Santiago Escobar [3] Salvador Lucas [4]

DSIC, Universidad Politécnica de Valencia, Spain

Abstract

Strategy annotations are used in eager programming languages (e.g., OBJ2, OBJ3, CafeOBJ, and Maude) for improving efficiency and/or reducing the risk of nontermination. Syntactically, they are given either as lists of natural numbers or as lists of integers associated to function symbols whose (absolute) values refer to the arguments of the corresponding symbol. A positive index forces the evaluation of an argument whereas a negative index means "evaluation on-demand". Recently, we have introduced a formal description of the operational meaning of such on-demand strategy annotations which improves previous formalizations that were lacking satisfactory computational properties. In this paper, we introduce an automatic, semantics–preserving program transformation which produces a program (without negative annotations) which can be then correctly executed by typical OBJ interpreters. Moreover, to demonstrate the practicality of our ideas, the program transformation has been implemented (in Haskell) and we compare the evaluation of transformed programs with the original ones on a set of representative benchmarks.

1 Introduction

Eager rewriting-based programming languages such as Lisp, OBJ*, CafeOBJ, ELAN, or Maude evaluate expressions by innermost rewriting. Since nontermination is a known problem of innermost reduction, *syntactic annotations* (generally specified as sequences of integers associated to function arguments, called *local strategies*) have been used in OBJ2 [10], OBJ3 [12], CafeOBJ [11], and Maude [7] to improve efficiency and (hopefully) avoid nontermination. Local strategies are used in OBJ programs [5] for guiding the *evaluation strategy*

[1] Work partially supported by CICYT TIC2001-2705-C03-01 and MCYT grants HA2001-0059 and HU2001-0019.

[2] Email:alpuente@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/alpuente.html

[3] Email:sescobar@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/sescobar.html

[4] Email:slucas@dsic.upv.es, URL:http://www.dsic.upv.es/users/elp/slucas.html

[5] As in [12], by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

(abbr. E-strategy): when considering a function call $f(t_1, \ldots, t_k)$, only the arguments whose indices are present as *positive* integers in the local strategy for f are evaluated (following the specified ordering). If 0 is found, then the evaluation of f is attempted. Unfortunately, this restriction of rewriting can have a negative impact in the ability to compute normal forms. Whenever the user provides no local strategy for a given symbol, the (Maude, OBJ*, CafeOBJ) interpreter automatically assigns a *default* E-strategy. For instance, the default local strategy of Maude associates the list $(1\ 2 \cdots k\ 0)$ to each k-ary symbol f having no explicit strategy, i.e. all arguments are marked as evaluable.

Example 1.1 Consider the following OBJ program (borrowed from [19]) where reductions on the second argument of the symbol `cons` are disallowed in order to make the program terminating:

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

The evaluation of the term `2nd(from(0))` as performed by a typical OBJ interpreter[6] is:

```
Maude> red 2nd(from(0)) .
reduce in Ex1 : 2nd(from(0)) .
rewrites: 1 in -10ms cpu (0ms real) (~ rewrites/second)
result Nat: 2nd(cons(0, from(s(0))))
```

This corresponds to the following reduction sequence:

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

The evaluation stops at this point since reductions on the second argument of `cons` are disallowed due to the local strategy `(1)`.

The handicaps of using only positive annotations regarding correctness and completeness of computations are discussed in [1,2,15,16,20,19]: the problem is that the absence of some indices in the local strategies can have a negative impact in the ability of such strategies to compute normal forms. For instance, a further step is required (*demanded* by the rule of `2nd`) in order to obtain the desired outcome in Example 1.1:

[6] We use the SRI's Maude interpreter (version 1.0.5) available at: `http://maude.csl.sri.com/system/`.

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

At this stage, it is not necessary to perform any reduction on symbol `from`, since reducing at the root position yields the final value:

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

In [20,19], *negative* indices are proposed to indicate those arguments that should be evaluated only 'on-demand', where the 'demand' is an attempt to match an argument term with the left-hand side of a rewrite rule [8,12,20]. For instance, in [19] the authors suggest that `(1 -2)` is the "apt" local strategy for `cons` in Example 1.1; i.e. the first argument is always evaluated but the second argument is evaluated only "on-demand". Then, the evaluation with strategy `(1 -2)` for `cons` is able to reduce `2nd(from(0))` to `s(0)` without entering in a non-terminating evaluation, whereas evaluation only with positive annotations enters in an infinite derivation or does not provide the associated normal form. It is worthy to note that the calculus is simpler than typical functional lazy rewriting and the appropriate (on-demand) strategy annotations for achieving suitable normal forms can be inferred from the program (see [1,2,15,16,20,19]).

On-demand strategy annotations have not been properly implemented to date: even if negative annotations are (syntactically) accepted in current OBJ implementations, namely OBJ3 and Maude, unfortunately they do not have the expected (on-demand) effect over the computations.

Example 1.2 Consider the following OBJ program (similar to Example 1.1 except the on-demand strategy annotation for symbol `cons`):

```
obj Ex2 is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat [strat (1)] .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat [strat (1 -2)] .
  op from : Nat -> LNat [strat (1 0)] .
  op 2nd  : LNat -> Nat [strat (1 0)] .
  vars X Y : Nat . var Ys : LNat .
  eq 2nd(cons(X,cons(Y,Ys))) = Y .
  eq from(X) = cons(X,from(s(X))) .
endo
```

The OBJ3 interpreter does not implement negative (on-demand) annotations though does *accept* this program and the evaluation of `2nd(from(0))` surprisingly delivers the very same result as in Example 1.1. That is, the negative annotation is just disregarded by the OBJ3 interpreter (which, in this case, causes loss of completeness). On the other hand, the Maude interpreter neither implements negative (on-demand) annotations though does also accept this program and the evaluation of the same expression diverges. This is because the negative annotation is interpreted by Maude as a positive one thus resulting in non-termination.

On the other hand, CafeOBJ is able to deal with negative annotations using the on-demand evaluation model of [19]. For instance, the CafeOBJ interpreter is able to compute the intended value `s(0)` of Example 1.1. However, in [1] we discussed a number of problems of the on-demand evaluation model of [19,20], as shown in the following example.

Example 1.3 [1] Consider the following OBJ program:

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

The expression `length'(from(0))` is rewritten (in one step) to the expression `length(from(0))`. No evaluation is demanded on the argument of `length'` for enabling this step (i.e. the negative annotation `-1` is included for `length'` but the corresponding rule includes a variable at the first argument of `length'`) and no further evaluation on `length(from(0))` should be performed (due to the local strategy `(0)` of `length` which forbids evaluation on any argument of `length`). However, the annotation `-1` of function `length'` is treated in such a way by the operational model of [20,19] that the on-demand evaluation of the expression `length'(from(0))` yields an infinite evaluation sequence (see [1] for a more detailed explanation).

We proposed in [1] a solution to these problems which is based on a suitable extension of the E-evaluation strategy of OBJ-like languages (that only considers annotations given as natural numbers) to cope with on-demand strategy annotations. Our strategy incorporates a better treatment of demandness and also enjoys good computational properties; in particular, we show how to use it for computing (head-)normal forms and we prove it is conservative w.r.t. other on-demand strategies: *lazy rewriting* [9] and *on-demand rewriting* [15]. A program transformation for proving termination of the on-demand evaluation strategy was also formalized, which relies on standard techniques. Furthermore, a *direct* implementation of the on-demand evaluation strategy of [1] has been developed. The system is called OnDemandOBJ and is publicly available at `http://www.dsic.upv.es/users/elp/soft.html` (see [3] for a description).

In this paper, we show how OBJ programs that use local strategies containing negative annotations (and hence could be correctly executed by using the evaluation strategy proposed in [1]) can be (also) executed in the existing OBJ implementations which only admit positive annotations (e.g. Maude). This is done by means of an automatic program transformation which encodes the ‘on-demand’ strategy instrumented by the negative annotations within new function symbols (and corresponding program rules) that only use positive strategy annotations. Before entering in too technical details, we give an example which illustrates the power of our transformation.

Example 1.4 The program of Example 1.2 is transformed by using our method into the following OBJ program *without* negative annotations.

```
obj Ex3 is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat           [strat (1)] .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat     [strat (0)] .
  op cons$_{root}$ : Nat LNat -> LNat  [strat (1 0)] .
  op cons$_{+2}$ : Nat LNat -> LNat   [strat (2)] .
  op from : Nat -> LNat          [strat (1 0)] .
  op 2nd  : LNat -> Nat          [strat (1 0)] .
  op 2nd$_{+1}$  : LNat -> Nat        [strat (1 0)] .
  op quoteNat : Nat -> Nat       [strat (0)] .
  op quoteLNat : LNat -> LNat    [strat (0)] .
  vars X Y : Nat . vars Xs : LNat .
  eq 2nd(cons(X,Xs)) = 2nd$_{+1}$(cons$_{+2}$(X,Xs)) .
  eq 2nd$_{+1}$(cons$_{+2}$(X,cons(Y,Xs))) = quoteNat(Y) .
  eq from(X) = quoteLNat(cons(X,from(s(X)))) .
  eq quoteNat(2nd(Xs)) = 2nd(quoteLNat(Xs)) .
  eq quoteNat(2nd$_{+1}$(Xs)) = 2nd(Xs) .
  eq quoteNat(s(X)) = s(quoteNat(X)) .
  eq quoteNat(0) = 0 .
  eq quoteLNat(from(X)) = from(quoteNat(X)) .
  eq quoteLNat(cons(X,Xs)) = cons$_{root}$(quoteNat(X),Xs) .
  eq quoteLNat(cons$_{+2}$(X,Xs)) = cons(X,Xs) .
  eq quoteLNat(nil) = nil .
  eq cons$_{root}$(X,Xs) = cons(X,Xs) .
endo
```

where $\texttt{cons}_{root}$ and $\texttt{cons}_{+2}$ are new symbols introduced by the transformation which perform the pattern matching in a stepwise manner, and `quoteNat` and `quoteLNat` are auxiliary symbols which help to perform a correct evaluation to head normal forms. Roughly speaking, for each constructor symbol `c` with a negative annotation $-i$, we remove all strategy annotations for `c` and introduce an auxiliary constructor symbol $\texttt{c}_{+i}$ with positive annotation i. Also, we introduce a defined symbol $\texttt{c}_{root}$ without the negative annotations but with the positive ones plus 0 and we introduce a new rule which is used to translate the new symbol $\texttt{c}_{root}$ back to `c`. Then, we add new rules which re–define symbols using constructor `c` in terms of `c`, $\texttt{c}_{root}$, and $\texttt{c}_{+i}$. Finally, for each program rule $l \to r$, we introduce a symbol `quote` in r (specialized for each sort) and add a number of new rules for symbols quote which transform c into $\mathtt{c}_{root}$ and help to appropriately (head)-normalize terms.

Now, the evaluation of 2nd(from(0)) using Maude yields:

```
Maude> red quoteNat(2nd(from(0))) .
reduce in Ex3 : quoteNat(2nd(from(0))) .
rewrites: 16 in -10ms cpu (0ms real) (~ rewrites/second)
result Nat: s(0)
```

which is the desired result. Note that for evaluating expression e, we only need to call $\mathtt{quote}_\tau(e)$ for the appropriate sort τ of e.

After some preliminaries in Section 2, we recall the on-demand evaluation of [1] in Section 3. Then, Section 4 introduces the program transformation together with completeness and correctness results. In Section 5, we experimentally demonstrate that the program transformation pays off in practice. Section 6 concludes. Proofs of all technical results can be found at the Appendix.

2 Preliminaries

Given a set A, $\mathcal{P}(A)$ denotes the set of all subsets of A. Let $R \subseteq A \times A$ be a binary relation on a set A. We denote the reflexive closure of R by $R^=$, its transitive closure by R^+, and its reflexive and transitive closure by R^* [6]. An element $a \in A$ is an R-normal form, if there exists no b such that $a\ R\ b$. We say that b is an R-normal form of a (written $a\ R^!\ b$), if b is an R-normal form and $a\ R^* b$. We say that R is *terminating* iff there is no infinite sequence $a_1\ R\ a_2\ R\ a_3 \cdots$.

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

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

A mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a *replacement map* (or $\mathcal{F}$-map) if $\forall f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ [14]. Let $M_{\mathcal{F}}$ be the set of all $\mathcal{F}$-maps. The ordering $\sqsubseteq$ on $M_{\mathcal{F}}$, the set of all $\mathcal{F}$-maps, is: $\mu \sqsubseteq \mu'$ if for all $f \in \mathcal{F}$, $\mu(f) \subseteq \mu'(f)$. Let $\mu^{can}_{\mathcal{R}}$ be the canonical replacement map, i.e. *the most restrictive replacement map which ensures that the non-variable subterms of the left-hand sides of the rules of $\mathcal{R}$ are replacing*, which is easily obtained from $\mathcal{R}$: $\forall f \in \mathcal{F}$, $i \in \{1, \ldots, ar(f)\}$, $i \in \mu^{can}_{\mathcal{R}}(f)$ iff $\exists l \in L(\mathcal{R}), p \in \mathcal{P}os_{\mathcal{F}}(l), (root(l|_p) = f \wedge p.i \in \mathcal{P}os_{\mathcal{F}}(l))$. 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 or equally restrictive than $\mu^{can}_{\mathcal{R}}$.

3 On-demand evaluation strategy

A local strategy for a k-ary symbol $f \in \mathcal{F}$ is a sequence $\varphi(f)$ of integers taken from $\{-k, \ldots, -1, 0, 1, \ldots, k\}$ which are given in parentheses. Symbols without an explicit local strategy are given a *default* annotation which depends on the considered language. A mapping φ that associates a local strategy $\varphi(f)$ to every $f \in \mathcal{F}$ is called an E-strategy map [18,19]. The E-strategy maps are used to correctly guide the evaluation strategy of OBJ-like languages in order to evaluate expressions. In the following, we recall the on-demand E-evaluation strategy of [1].

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

Let $\mathbb{L}_n$ be the set of all lists of integers whose absolute value does not exceed $n \in \mathbb{N}$. Given an E-strategy map φ, we use the signature $\mathcal{F}^{\sharp}_{\varphi} =$

$\cup\{f_{L_1|L_2}, \overline{f}_{L_1|L_2} \mid f \in \mathcal{F} \wedge L_1, L_2 \in \mathbb{L}_{ar(f)}.(L_1{+}{+}L_2 \sqsubseteq \varphi(f))\}$ (where the function $++$ defines the concatenation of two sequences of integers) and labelled 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 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 this subterm, and the control goes back to the parent. The list L_2 of $L_1 \mid L_2$ denotes the (partially consumed) local strategy being considered for a given symbol, whereas L_1 is interpreted as a kind of store which records previously considered annotations. The main idea is that annotations move from L_2 to L_1 once they have been processed.

We use $f^{\sharp}$ to denote f or $\overline{f}$, for a symbol $f \in \mathcal{F}$. We define the list of *activable* indices of a labelled symbol $f^{\sharp}_{L_1|L_2}$ as $activable(f^{\sharp}_{L_1|L_2}) = \begin{cases} L_1 \text{ if } L_1 \neq nil \\ L_2 \text{ if } L_1 = nil \end{cases}$.

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

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

Also, the operator $erase : \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ removes all extra indices.

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

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

Note that the restriction of disagreeing positions to positions with defined symbols (in other words, non-constructor symbols) disables the evaluation of subterms which could never produce a redex (see [1,4,13,17]).

Example 3.1 Let us consider l_1 = `2nd(cons(0,cons(Y,Ys)))` and l_2 = `2nd(cons(s(X),cons(Y,Ys)))`, where $\mathcal{C} = \{\texttt{cons}, \texttt{nil}, \texttt{s}, \texttt{0}\}$ and $\mathcal{D} = \{\texttt{2nd}, \texttt{from}\}$. Let t_1 = `2nd(cons(0,nil))`, we have $DP_{l_1}(t_1) = DP_{l_2}(t_1) = \varnothing$, i.e. no position is demanded by l_1 or l_2 because of a constructor conflict with subterm `nil` at position 1.2. Let t_2 = `2nd(cons(2nd(from(0)),from(0)))`, we have $DP_{l_1}(t_2) = DP_{l_2}(t_2) = \{1.1, 1.2\}$, i.e. positions 1.1 and 1.2 are demanded by l_1 and l_2 because both positions are function-rooted. And, let t_3 = `2nd(cons(0,from(0)))`, we have $DP_{l_1}(t_3) = \{1.2\}$ but $DP_{l_2}(t_3) = \varnothing$, i.e. position 1.2 is demanded by l_1 but not by l_2 because of a constructor conflict with l_2.

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

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

and the set of activable 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 activable demanded positions to non-positive and non-empty positions is consistent w.r.t the intended meaning of strategy annotations since positive or empty positions should not be evaluated on-demand (see [1]).

Example 3.2 Let us consider the term $l =$ `2nd(cons(0,cons(Y,Yz)))`. Let

$$t_1 = \texttt{2nd}_{(1)|(0)}(\texttt{cons}_{(1\ 2)|nil}(\texttt{0}_{nil|nil}, \texttt{from}_{nil|(1\ 0)}(\texttt{s}_{nil|(1)}(\texttt{0}_{nil|nil}))))$$

we have $DP_l(erase(t_1)) = \{1.2\}$ but $ADP_l(t_1) = \varnothing$, i.e. position 1.2 is demanded by l but it is a positive position. Let

$$t_2 = \texttt{2nd}_{(1)|(0)}(\texttt{cons}_{(1\ -2)|nil}(\texttt{0}_{nil|nil}, \texttt{from}_{nil|nil}(\texttt{s}_{nil|(1)}(\texttt{0}_{nil|nil}))))$$

we have $ADP_l(t_2) = \varnothing$, i.e. position 1.2 is still demanded by l but it is rooted by a symbol with an empty annotation list. Let

$$t_3 = \texttt{2nd}_{(1)|(0)}(\texttt{cons}_{(1)|nil}(\texttt{0}_{nil|nil}, \texttt{from}_{nil|(1\ 0)}(\texttt{s}_{nil|(1)}(\texttt{0}_{nil|nil}))))$$

we have $ADP_l(t_3) = \varnothing$, i.e. position 1.2 is again demanded by l but it is not an activable position. Finally, let

$$t_4 = \texttt{2nd}_{(1)|(0)}(\texttt{cons}_{(1\ -2)|nil}(\texttt{0}_{nil|nil}, \texttt{from}_{nil|(1\ 0)}(\texttt{s}_{nil|(1)}(\texttt{0}_{nil|nil}))))$$

we have $ADP_l(t_4) = \{1.2\}$.

Given a term $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, the total ordering $\leq_s$ between activable positions of s is defined as (1) $\Lambda \leq_s p$ for all $p \in \mathcal{P}os_A(s)$; (2) if $i.p, i.q \in \mathcal{P}os_A(s)$ and $p \leq_{s|_i} q$, then $i.p \leq_s i.q$; and (3) if $i.p, j.q \in \mathcal{P}os_A(s)$, $i \neq j$, and i (or $-i$) appears before j (or $-j$) in $activable(root(s))$, then $i.p \leq_s j.q$. The ordering $\leq_s$ allows us to choose a position from the set of all activable demanded positions in s, which is consistent with user's annotations (see $min_{\leq_s}$ below). We define the set $OD_\mathcal{R}(s)$ of on-demand positions of a term $s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ w.r.t. TRS $\mathcal{R}$ as follows:

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

Example 3.3 Continuing Example 3.2. Let us consider the term

$$t_5 = \texttt{2nd}_{(1)|(0)}(\texttt{cons}_{(-1\ -2)|nil}(\texttt{2nd}_{nil|(10)}(\texttt{nil}_{nil|nil}), \texttt{from}_{nil|(1\ 0)}(\texttt{0}_{nil|nil})))$$

we have $ADP_l(t_5) = \{1.1, 1.2\}$ but $OD_{\{l\}}(t_5) = \{1.1\}$ since annotation `-1` appears before `-2` in the memoizing list for symbol `cons` and hence $1.1 \leq_{t_5} 1.2$.

Given a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and position $p \in \mathcal{P}os(t)$, $mark(t,p)$ is the term s with all symbols above p (except the root) marked as non-evaluable, in symbols $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t)$, if $\Lambda < q < p$ and $root(t|_q) = f_{L_1|L_2}$, then $root(s|_q) = \overline{f}_{L_1|L_2}$, otherwise $root(s|_q) = root(t|_q)$. This is useful for avoiding recursive definitions of the evaluation strategy (see [1]).

Example 3.4 Continuing Example 3.2. We have that $mark(t_4, 1.2) =$

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

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

Definition 3.5 Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and an arbitrary E-strategy map φ for $\mathcal{F}$, we define: $eval^\varphi_\mathcal{R}(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle \varphi(t), \Lambda \rangle \overset{\sharp}{\to}{}^!_\varphi \langle s, \Lambda \rangle\}$. The binary relation $\overset{\sharp}{\to}_\varphi$ on $\mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi) \times \mathbb{N}^*_+$ is defined as follows: $\langle t, p \rangle \overset{\sharp}{\to}_\varphi \langle s, q \rangle$ if and only if $p \in \mathcal{P}os(t)$ and either

(i) $t|_p = f_{L|nil}(t_1, \ldots, t_k)$, $s = t$ and $p = q.i$ for some i; or

(ii) $t|_p = f_{L_1|i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $q = p.i$, and $s = t[f_{L_1@i|L_2}(t_1, \ldots, t_k)]_p$; or

(iii) $t|_p = f_{L_1|-i:L_2}(t_1, \ldots, t_k)$, $i > 0$, $q = p$, and $s = t[f_{L_1@-i|L_2}(t_1, \ldots, t_k)]_p$; or

(iv) $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k) = \sigma(l')$, $erase(l') = l$, $s = t[\sigma(\varphi(r))]_p$ for some $l \to r \in R$ and substitution σ, $q = p$; or

(v) $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $erase(t|_p)$ 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

(vi) $t|_p = f_{L_1|0:L_2}(t_1, \ldots, t_k)$, $erase(t|_p)$ is not a redex, $OD_\mathcal{R}(t|_p) = \{p'\}$, $s = t[mark(t|_p, p')]_p$, $q = p.p'$; or

(vii) $t|_p = \overline{f}_{L_1|L_2}(t_1, \ldots, t_k)$, $s = t[f_{L_1|L_2}(t_1, \ldots, t_k)]_p$ and $p = q.i$ for some i.

Case (i) means that no more annotations are provided and the evaluation is completed. In case (ii), a positive argument index is provided and the evaluation jumps to the subterm at such argument (note that the index is stored). Case (iii) only stores the negative index for further use. Cases (iv), (v), and (vi) consider the attempt to match the term against the left-hand sides of the rules of the program. Case (iv) applies if the considered (unlabelled) subterm is a redex (which is, then, contracted). If the subterm is not a redex, cases (v) and (vi) are considered (possibly involving some on-demand evaluation). We use the lists of indices labelling the symbols for recording the concrete positions on which we are able to allow on-demand evaluations; in particular, the first (memoizing) list is crucial for achieving this (see definition of set $OD_\mathcal{R}$, which uses function *activable* and the ordering $\leq_s$ for indicating the positions demanded by some rule in order to become a redex). Case (v) applies if no demanded evaluation is allowed (or required). Case (vi)

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

Fig. 1. On-demand evaluation of term `2nd(from(0))`

applies if the on-demanded evaluation of the subterm $t|_{p.p'}$ is required. In this case, the symbols lying on the path from $t|_p$ to $t|_{p.p'}$ (excluding the border ones) are overlined. Then, the evaluation process continues onto $t|_{p.p'}$. Once the evaluation of $t|_{p.p'}$ has finished, the only possibility is the repeated (but possibly empty) application of steps corresponding to the last case (vii), which implements the return of the evaluation process back to position p (which originated the on-demand evaluation).

Example 3.6 Following the Example 1.2, the appropriate evaluation sequence of the term `2nd(from(0))` via $eval^{\varphi}_{\mathcal{R}}$ is depicted in Figure 1, where the index considered in each step is surrounded by a box. Roughly speaking, the following rewriting sequence

$$
\begin{aligned}
&\mathtt{2nd(}\underline{\mathtt{from(0)}}\mathtt{)}\\
&\quad\to\ \mathtt{2nd(cons(0,}\underline{\mathtt{from(s(0))}}\mathtt{))}\\
&\quad\to\ \underline{\mathtt{2nd(cons(0,cons(s(0),from(s(s(0))))))}}\\
&\quad\to\ \mathtt{s(0)}
\end{aligned}
$$

is performed while managing strategy annotations.

4 The Program Transformation

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

Example 4.1 Consider the following OBJ program which is mainly borrowed from a CafeOBJ program in [20] where we consider negative indices for `cons`:

```
obj Ex3rd is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat       [strat (1)] .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat [strat (1 -2)] .
  op from : Nat -> LNat      [strat (1 0)] .
  op 3rd  : LNat -> Nat      [strat (1 0)] .
  vars X Y Z : Nat . var Zs : LNat .
  eq 3rd(cons(X,cons(Y,cons(Z,Zs)))) = Z .
  eq from(X) = cons(X,from(s(X))) .
endo
```

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

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

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

Let us consider a naïve approach for transforming negative indices into positive indices which duplicates the symbols containing negative indices into new symbols containing the positive counterparts (as in the program transformation showed in [1] for approximating termination). The raw application of such program transformation to the previous example delivers the following rules:

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

together with the following definitions for symbols `cons` and $\texttt{cons}_{+2}$:

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

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

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

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

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

Nevertheless, we can define a new symbol $\texttt{cons}_{root}$ which inherits the behavior of the original symbol cons (i.e. the positive indices of cons), and consistently rename the symbols in the TRS through a special symbol quote before evaluating each term (note that quote is specialized to sorts). Moreover, we introduce a new rule for translate $\texttt{cons}_{root}$ back to cons. We finally obtain the program:

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

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

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

4.1 Transformation for fixing rules

Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, and φ be an E-strategy map. We define the set of positions which are activable (but not reducible) of a term $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ as $\mathcal{P}os_{A-P}(t) = \mathcal{P}os_A(t) - \mathcal{P}os_P(t)$. Given a strategy map φ, φ_+ denotes the result of removing all negative indices from φ.

Let us define the set of symbols of the original TRS at positions activable but not reducible which have positive indices which can be lost as $\mathcal{F}^\varphi_\mathcal{R} = \{f \in \mathcal{F} \mid ar(f) > 0 \wedge \varphi_+(f) \neq nil \wedge \exists l \in L(\mathcal{R}) : \mathcal{P}os_f(l) \cap \mathcal{P}os_{A-P}(\varphi(l)) \neq \varnothing\}$. Note that if $\mathcal{R}$ is a CS, then $\mathcal{F}^\varphi_\mathcal{R} \subseteq \mathcal{C}$. In the following, we define the set of auxiliary rules $Quote_{\mathcal{R}^\mathbb{I},\mathcal{F}^\varphi_\mathcal{R}}$ to appropriately (head)-normalize terms. Intuitively, `quote` translates a symbol $f \in \mathcal{F}^\varphi_\mathcal{R}$ at a positive position into the new symbol f_{root}.

$$Quote_{\mathcal{R}^\mathbb{I},\mathcal{F}^\varphi_\mathcal{R}} = \bigcup_{f\in\mathcal{F}} \begin{cases} \mathtt{quote}(f(\overline{x})) \to f_{root}(\rho_f(\overline{x})) & \text{if } f \in \mathcal{F}^\varphi_\mathcal{R} \\ \mathtt{quote}(f(\overline{x})) \to f(\rho_f(\overline{x})) & \text{if } f \notin \mathcal{F}^\varphi_\mathcal{R} \end{cases}$$

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

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

$$R^\mathbb{I} = \{f(\bar{t}) \to \mathtt{quote}(r) \mid f(\bar{t}) \to r \in R\} \cup \{f_{root}(\overline{x}) \to f(\overline{x}) \mid f \in \mathcal{F}^\varphi_\mathcal{R}\}$$
$$\cup\, Quote_{\mathcal{R}^\mathbb{I},\mathcal{F}^\varphi_\mathcal{R}}$$

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

Example 4.2 Consider the TRS $\mathcal{R}$ and the E-strategy map φ of Example 4.1. The TRS $\mathcal{R}^\mathbb{I}$ together with $\varphi^\mathbb{I}$ is:

```
obj Ex3rdI is
  sorts Nat LNat .
  op 0    : -> Nat .
  op s    : Nat -> Nat            [strat (1)] .
  op nil  : -> LNat .
  op cons : Nat LNat -> LNat      [strat (1 -2)] .
  op cons_root : Nat LNat -> LNat  [strat (1 -2 0)] .
  op from : Nat -> LNat           [strat (1 0)] .
  op 3rd  : LNat -> Nat           [strat (1 0)] .
  op quoteNat : Nat -> Nat        [strat (0)] .
```

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

4.2 Transformation for eliminating negative indices

We formulate the *transformer* for switching negative indices into positive ones. Intuitively, we tranform a rule $l \to r$ with a position p which is activable but not reducible into the rules $l[x]_p \to l'[x]_p$ and $l' \to r$, where each symbol f in l from the root to p has been replaced in l' by the new symbol f_{+i} and its negative indices are replaced by their positive counterparts.

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

$$R^{neg} = R - \{l \to r\} \cup \{l[y]_{p.i} \to l'[y]_{p.i}, l' \to r\} \cup \{\mathtt{quote}(f^q_{+j_q}(\overline{x})) \to f^q(\overline{x})\}$$

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

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

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

$$\varphi^{neg}(f_{+j}) = \begin{cases} \mathtt{(j\ 0)} & \text{if } \mathtt{(-j\ 0)} \sqsubseteq \varphi(f) \vee \mathtt{(j\ 0)} \sqsubseteq \varphi(f) \\ \mathtt{(j)} & \text{if } \mathtt{(-j\ 0)} \not\sqsubseteq \varphi(f) \wedge \mathtt{(j\ 0)} \not\sqsubseteq \varphi(f) \end{cases}$$

Example 4.3 Consider the TRS $\mathcal{R} = \mathcal{R}^\mathbb{I}$ and the E-strategy map $\varphi = \varphi^\mathbb{I}$ in Example 4.2. The application of the transformer, the TRS $\mathcal{R}^{neg}$ together with φ^{neg}, is:

```
  obj Ex3rdINeg is
    sorts Nat LNat .
    op 0    : -> Nat .
    op s    : Nat -> Nat           [strat (1)] .
    op nil  : -> LNat .
    op cons : Nat LNat -> LNat     [strat (1 -2)] .
    op cons$_{root}$ : Nat LNat -> LNat  [strat (1 -2 0)] .
    op cons$_{+2}$ : Nat LNat -> LNat    [strat (2)] .
    op from : Nat -> LNat          [strat (1 0)] .
    op 3rd  : LNat -> Nat          [strat (1 0)] .
    op 3rd$_{+1}$  : LNat -> Nat         [strat (1 0)] .
    op quoteNat : Nat -> Nat       [strat (0)] .
    op quoteLNat : LNat -> LNat    [strat (0)] .
    vars X Y Z : Nat . var Zs : LNat .
    eq 3rd(cons(X,cons(Y,Zs))) = 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,Zs))) .
    eq 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,cons(Z,Zs)))) = quoteNat(Z) .
    eq from(X) = quoteLNat(cons(X,from(s(X)))) .
    eq quoteNat(3rd(Zs)) = 3rd(quoteLNat(Zs)) .
    eq quoteNat(3rd$_{+1}$(Zs)) = 3rd(Zs) .
    eq quoteNat(s(X)) = s(quoteNat(X)) .
    eq quoteNat(0) = 0 .
    eq quoteLNat(from(X)) = from(quoteNat(X)) .
    eq quoteLNat(cons(X,Zs)) = cons$_{root}$(quoteNat(X),Zs) .
    eq quoteLNat(cons$_{+2}$(X,Zs)) = cons(X,Zs) .
    eq quoteLNat(nil) = nil .
    eq cons$_{root}$(X,Zs) = cons(X,Zs) .
  endo
```

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

```
    eq 3rd(cons(X,cons(Y,Zs))) = 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,Zs))) .
    eq 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,cons(Z,Zs)))) = quoteNat(Z) .
```

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

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

Example 4.4 Continuing with Example 4.3. The final TRS $\mathcal{R}^{\mathbb{III}}$ together with $\varphi^{\mathbb{III}}$ is:

```
  obj Ex3rdII is
    sorts Nat LNat .
    op 0    : -> Nat .
    op s    : Nat -> Nat           [strat (1)] .
    op nil  : -> LNat .
    op cons : Nat LNat -> LNat     [strat (0)] .
    op cons$_{root}$ : Nat LNat -> LNat  [strat (1 0)] .
    op cons$_{+2}$ : Nat LNat -> LNat    [strat (2)] .
    op from : Nat -> LNat          [strat (1 0)] .
    op 3rd  : LNat -> Nat          [strat (1 0)] .
    op 3rd$_{+1}$  : LNat -> Nat         [strat (1 0)] .
    op quoteNat : Nat -> Nat       [strat (0)] .
    op quoteLNat : LNat -> LNat    [strat (0)] .
    vars X Y Z : Nat . var Zs : LNat .
    eq 3rd(cons(X,Zs)) = 3rd$_{+1}$(cons$_{+2}$(X,Zs))) .
    eq 3rd$_{+1}$(cons$_{+2}$(X,cons(Y,Zs))) = 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,Zs))) .
    eq 3rd$_{+1}$(cons$_{+2}$(X,cons$_{+2}$(Y,cons(Z,Zs)))) = quoteNat(Z) .
    eq from(X) = quoteLNat(cons(X,from(s(X)))) .
    eq quoteNat(3rd(Zs)) = 3rd(quoteLNat(Zs)) .
    eq quoteNat(3rd$_{+1}$(Zs)) = 3rd(Zs) .
    eq quoteNat(s(X)) = s(quoteNat(X)) .
    eq quoteNat(0) = 0 .
    eq quoteLNat(from(X)) = from(quoteNat(X)) .
    eq quoteLNat(cons(X,Zs)) = cons$_{root}$(quoteNat(X),Zs) .
    eq quoteLNat(cons$_{+2}$(X,Zs)) = cons(X,Zs) .
    eq quoteLNat(nil) = nil .
    eq cons$_{root}$(X,Zs) = cons(X,Zs) .
  endo
```

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

Example 4.5 Consider the following program rule:

```
min(s(X),0,s(0),s(s(Y))) = 0
```

with the strategy map φ(`min`) = `(-3 -2 1 -4 0)`, φ(`s`) = `(-1)`, and φ(`0`) = *nil*. The transformation of this rule produces (without considering symbol `quote`):

```
min(s(X),Z,W,Y) = min$_{+3}$(s(X),Z,W,Y)
min$_{+3}$(s(X),Z,s(W),Y) = min$_{+3}$(s(X),Z,s$_{+1}$(W),Y)
min$_{+3}$(s(X),Z,s$_{+1}$(0),Y) = min$_{+2}$(s(X),Z,s(0),Y)
min$_{+2}$(s(X),0,s(0),Y) = min$_{+4}$(s(X),0,s(0),Y)
min$_{+4}$(s(X),0,s(0),s(Y)) = min$_{+4}$(s(X),0,s(0),s$_{+1}$(Y))
min$_{+4}$(s(X),0,s(0),s$_{+1}$(s(Y))) = 0
```

and the strategy map $\varphi^{\mathbb{III}}$(`min`) = `(1 0)`, $\varphi^{\mathbb{III}}$(`min`$_{+2}$) = `(2 0)`, $\varphi^{\mathbb{III}}$(`min`$_{+3}$) = `(3 0)`, $\varphi^{\mathbb{III}}$(`min`$_{+4}$) = `(4 0)`, $\varphi^{\mathbb{III}}$(`s`) = *nil*, $\varphi^{\mathbb{III}}$(`s`$_{+1}$) = `(1)`, and $\varphi^{\mathbb{III}}$(`0`) = *nil*. This transformed program reproduces the ordering between the different on-demand paths of the left-hand side.

4.3 Properties

In the following, we establish the main results of the program transformation. We define a *standard* E-strategy map φ as an E-strategy map where no index different to `0` appears at the right of any index `0` in the annotation sequence.

Given a strategy map φ for $\mathcal{F}$, we say that a TRS $\mathcal{R} = (\mathcal{F}, R)$ is φ-terminating if, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, there is no infinite $\overset{\sharp}{\rightarrow}_{\varphi}$-rewrite sequence starting from $\langle \varphi(t), \Lambda \rangle$. Note that Examples 1.2, 1.3, and 4.1 can be proved φ-terminating using the technique developed in [1]. A defined function $f \in \mathcal{D}$ is *completely defined* if it does not occur in any ground term in normal form, i.e. functions are reducible on all ground terms (of appropriate sort). A TRS $\mathcal{R}$ is *completely defined* (or CD) if each defined function of the signature is completely defined. In the following theorem we prove completeness of the transformation, i.e. that the transformation preserves normal forms.

Theorem 4.6 (Completeness) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that $\mathcal{R}$ is φ-terminating. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in eval^{\varphi}_{\mathcal{R}}(t)$, then $s \in eval^{\varphi^{\mathbb{II}}}_{\mathcal{R}^{\mathbb{II}}}(t)$.*

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

Theorem 4.7 (Correctness) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in eval^{\varphi^{\mathbb{II}}}_{\mathcal{R}^{\mathbb{II}}}(t)$, then $s \in eval^{\varphi}_{\mathcal{R}}(t)$.*

Finally, termination is preserved by the transformation.

Theorem 4.8 (Termination) *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. $\mathcal{R}$ is φ-terminating iff $\mathcal{R}^{\mathbb{II}}$ is $\varphi^{\mathbb{II}}$-terminating.*

5 Experiments

A prototype implementation of the transformation proposed in this paper has been developed in Haskell (using ghc 5.04.2). The system is publicly available `http://www.dsic.upv.es/users/elp/soft.html`.

Tables 1, and 2, show the runtimes[7] in milliseconds and the number of evaluation steps of the benchmarks for the different OBJ-family systems. The OnDemandOBJ interpreter is the on-demand prototype interpreter of the on-demand evaluation of [1]. CafeOBJ[8] is developed in Lisp at the Japan Advanced Inst. of Science and Technology (JAIST); OBJ3[9], also written in LISP, is maintained by the University of California at San Diego; Maude[10] is developed in C++ and maintained by the Computer Science Lab at SRI International. OBJ3 and Maude provide only computations with positive annotations whereas CafeOBJ provides computations with negative annotations as well, using the on-demand evaluation of [20,19]. OnDemandOBJ computes

[7] The average of 10 executions measured in a Pentium III machine running redhat 7.2.
[8] Available at `http://www.ldl.jaist.ac.jp/Research/CafeOBJ/system.html`.
[9] Available at `http://www.kindsoftware.com/products/opensource/obj3/OBJ3/`.
[10] Available at `http://maude.cs.uiuc.edu/current/system/`.

with negative annotations using the on-demand evaluation of [1]. Note that CafeOBJ and OBJ3 implement sharing of variables whereas Maude and OnDemandOBJ do not; thus, the number of evaluation steps in Table 2 is pairwise equivalent: CafeOBJ and OBJ3 in one hand and Maude and OnDemandOBJ in the other hand. Also, since Maude is implemented in C++, typical execution times are nearly 0 milliseconds. Finally, the mark *overflow* in Table 2 indicates that execution raised a memory overflow and normal form was not achieved; whereas the mark *unavailable* in Tables 1 and 2 indicates that the program can not be executed in such OBJ implementation.

The benchmark `pi` codifies the well-known infinite serie expansion to approximate number π: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$ and uses negative annotations to obtain a terminating and complete example, which can not be obtained using only positive annotations. Termination of the program can be formally proved using the technique of [1]. Also, by using the results in [2], we can guarantee that every expression such as `pi(`n`)` for some n of sort `Nat` produces (as expected) a completely evaluated expression of sort `LRecip`. The benchmark `pi_noneg` consists of the application of the program transformation described in this paper to `pi`. Table 1 compares the evaluation of expression `pi(square(square(3)))` using `pi` and `pi_noneg`. Note that the right input expression for `pi_noneg` is `quoteLRecip(pi(square(square(3))))`. It witnesses that negative annotations are extremely useful in practice and that the program transformation enables the execution of negatively annotated programs in all OBJ implementations. On the other hand, Table 1 also evidences that the implementation of the on-demand evaluation strategy in other systems is quite promising.

On the other hand, Table 2 illustrates the interest of using negative annotations to improve the behavior of programs: the benchmark `msquare_eager` codifies the functions `square`, `minus`, `times`, and `plus` over natural numbers using only positive annotations. Every k-ary symbol f is given a strategy $(1\ 2\ \cdots k\ 0)$ (this corresponds to *default* strategies in Maude). Note that the program is terminating as a TRS (i.e., without any strategy annotation). The benchmark `msquare_apt` is similar to `msquare_eager`, but *canonical* positive strategies are provided: the i-th argument of a symbol f is annotated if there is an occurrence of f in the left-hand side of a rule having a non-variable i-th argument; otherwise, the argument is not annotated (see [5]). The benchmark `msquare_neg` is similar to `msquare_eager`, though *canonical* arbitrary strategies are provided: now (from left-to-right), the i-th argument of a defined symbol f is annotated if all occurrences of f in the left-hand side of the rules contain a non-variable i-th argument; if all occurrences of f in the left-hand side of the rules have a variable i-th argument, then the argument is not annotated; in any other case, annotation $-i$ is given to f (see [5]). The benchmark `msquare_noneg` represents the application of the

ms./rewrites	`pi`	`pi_noneg`
OnDemandOBJ	25/364	215/35532
CafeOBJ	30/364	190/35532
OBJ3	*unavailable*	100/35532
Maude	*unavailable*	30/35532

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

ms./rewrites	`msquare_eager`	`msquare_apt`	`msquare_neg`	`msquare_noneg`
OnDemandOBJ	33/ 715 40/ 914	62/ 1640 78/ 1992	0/ 1 80/ 1992	0/ 4 750/ 126089
CafeOBJ	40/ 715 50/ 914	50/ 715 60/ 914	0/ 1 60/ 914	0/ 4 630/ 126089
OBJ3	20/ 715 30/ 914	*overflow* *overflow*	*unavailable* *unavailable*	0/ 4 *overflow*
Maude	0/ 715 0/ 914	0/ 1640 3/ 1992	*unavailable* *unavailable*	0/ 4 90/ 126089

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

program transformation to `msquare_neg`. Note that the right input expressions for `msquare_noneg` are `quoteNat(minus(0,square(square(5))))` and `quoteNat(minus(square(square(5)),square(square(3))))`. Then, for instance, program `msquare_neg` runs in less time and requires a smaller number of rewrite steps than `msquare_eager` or `msquare_apt`, which do not include negative annotations. Note the difference in the number of rewrite steps of benchmarks `msquare_eager` and `msquare_apt` for the Maude and OnDemandOBJ systems, which is due to the absence of variable sharing. Moreover, note that the program transformation is also very useful since execution of the expression `minus(0,square(square(5)))` is improved. These experimental results, together with the OBJ source programs, are available at `http://www.dsic.upv.es/users/elp/ondemandOBJ/experiments`. The OBJ source programs can also be found at [3].

6 Conclusions

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

- The proposed transformation preserves the termination of the original program which uses (positive and) negative annotations. That is, if the original program terminates under the evaluation strategy of [1], then the transformed program terminates also.
- Correct and completeness of the transformation holds w.r.t. the semantics of strategy annotations given in [1]. That is, the semantics of input expressions in the original program (under the on-demand E-strategy of [1]) and in the transformed program (under the E-strategy) do coincide.

Moreover, our transformation is useful both for

(i) making possible the use of arbitrary strategy annotations in languages that (syntactically) allow them but that still do not provide the necessary operational support (e.g., OBJ3).

(ii) providing a notion of negative strategy annotation (somewhat laziness) for languages that does not allow them (e.g., Maude).

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

References

[1] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. Improving on-demand strategy annotations. In 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.

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

[3] M. Alpuente, S. Escobar, and S. Lucas. Ondemandobj: a laboratory for strategy annotations. In J. Giavitto and P. 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.

[4] 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.

[5] 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.

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

[7] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proc. of the 1st International Workshop on Rewriting Logic and its Applications, RWLW 96*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89. Elsevier Sciences Publisher, 1996.

[8] S. Eker. Term rewriting with operator evaluation strategies. In C. Kirchner and H. Kirchner, editors, *Proc. of the 2nd International Workshop on Rewriting Logic and its Applications, WRLA 98*, volume 15 of *Electronic Notes in Theoretical Computer Science.* Elsevier Sciences Publisher, 2000.

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

[10] 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.

[11] 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.

[12] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[13] R. Loogen, F. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In *Proc. of PLILP'93*, volume 714 of *Lecture Notes in Computer Science*, pages 184–200. Springer-Verlag, Berlin, 1993.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

A Proofs

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

Definition A.1 Let φ be a strategy map over signature $\mathcal{F}$. Let $\varphi^{\mathbb{II}}$ be a strategy map over signature $\mathcal{F}^{\mathbb{II}}$. Let $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$. We define the translation of t into terms $\mathcal{T}({\mathcal{F}^{\mathbb{II}}}^\sharp_{\varphi^{\mathbb{II}}}, \mathcal{X}^\sharp_{\varphi^{\mathbb{II}}})$ as $t_{\downarrow\varphi^{\mathbb{II}}} = s$ where $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t).root(t|_q) = f_{L_1|L_2}$, $root(s|_q) = f_{L'_1|L'_2}$ where $L'_1 = L_{1\downarrow\varphi^{\mathbb{II}}(f)}$ and $L'_2 = L_{2\downarrow\varphi^{\mathbb{II}}(f)}$.

Definition A.2 Let φ be a strategy map over signature $\mathcal{F}$ and $\varphi^{\mathbb{II}}$ be a strategy map over signature $\mathcal{F}^{\mathbb{II}}$. Let $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os_A(t)$. We define the translation of t into terms $\mathcal{T}({\mathcal{F}^{\mathbb{II}}}^\sharp_{\varphi^{\mathbb{II}}}, \mathcal{X}^\sharp_{\varphi^{\mathbb{II}}})$ as $pos_p(t) = s$ where $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t).root(t|_q) = f_{L_1|L_2}$, we have:

$$root(s|_q) = \begin{cases} f_{root\,L'_1|L'_2} \text{ if } p \in \mathcal{P}os_P(t), q \leq p, \text{ and } f \in \mathcal{F}^\varphi_\mathcal{R}; \\ \qquad\qquad \text{where } L'_1 = L_{1\downarrow\varphi^{\mathbb{II}}(f_{root})}, \text{ and } L'_2 = L_{2\downarrow\varphi^{\mathbb{II}}(f_{root})} \\ f_{L'_1|L'_2} \quad \text{otherwise; where } L'_1 = L_{1\downarrow\varphi^{\mathbb{II}}(f)}, \text{ and } L'_2 = L_{2\downarrow\varphi^{\mathbb{II}}(f)} \end{cases}$$

Definition A.3 Let φ be a strategy map over signature $\mathcal{F}$ and $\varphi^{\mathbb{II}}$ be a strategy map over signature $\mathcal{F}^{\mathbb{II}}$. Let $t \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$ and $p \in \mathcal{P}os_A(t)$. We define the translation of t into terms $\mathcal{T}({\mathcal{F}^{\mathbb{II}}}^\sharp_{\varphi^{\mathbb{II}}}, \mathcal{X}^\sharp_{\varphi^{\mathbb{II}}})$ as $neg_p(t) = s$ where $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t).root(t|_q) = f_{L_1|L_2}$, we have:

$$root(s|_q) = \begin{cases} f_{+i\,L'_1|L'_2} \text{ if } p \in \mathcal{P}os_{A-P}(t) \text{ and } q.i \leq p; \text{ where } L'_1 = \texttt{(i)} \text{ and} \\ \qquad\qquad L'_2 = \begin{cases} \texttt{(0)} \text{ if } \texttt{(-i 0)} \sqsubseteq \varphi(f) \vee \texttt{(i 0)} \sqsubseteq \varphi(f) \\ nil \text{ if } \texttt{(-i 0)} \not\sqsubseteq \varphi(f) \wedge \texttt{(i 0)} \not\sqsubseteq \varphi(f) \end{cases} \\ f_{L'_1|L'_2} \quad \text{otherwise; where } L'_1 = L_{1\downarrow\varphi^{\mathbb{II}}(f)}, \text{ and } L'_2 = L_{2\downarrow\varphi^{\mathbb{II}}(f)} \end{cases}$$

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

Proposition A.4 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that $\mathcal{R}$ is φ-terminating. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. For all $t, s \in \mathcal{T}(\mathcal{F}^\sharp_\varphi, \mathcal{X}^\sharp_\varphi)$, and $q \in \mathcal{P}os_A(s)$, if $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}_{\varphi,\mathcal{R}} \langle s, q\rangle$, then $\langle pos_\Lambda(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^*_{\varphi^{\mathbb{II}},\mathcal{R}^{\mathbb{II}}} \langle s', q\rangle$ where either (1) $s' = \texttt{quote}_\tau(s_{\downarrow\varphi^{\mathbb{II}}})$, (2) $s' = pos_\Lambda(s)$, (3) $s' = neg_q(s)$, or (4) s and s' are $\overset{\sharp}{\rightarrow}$-normal forms with a defined symbol at root position.*

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

(i) The cases $t = f_{L|nil}(t_1,\ldots,t_k)$ or $t = \overline{f}_{L_1|L_2}(t_1,\ldots,t_k)$ are impossible.

(ii) Let $t = f_{L_1|i:L_2}(t_1,\ldots,t_k)$, $i > 0$, $q = i$, and $s = f_{L_1@i|L_2}(t_1,\ldots,t_k)$. If $f \notin \mathcal{F}^{\varphi}_{\mathcal{R}}$, then $root(pos_\Lambda(t)) = f_{L_1|i:L_2}$ and the conclusion follows. If $f \in \mathcal{F}^{\varphi}_{\mathcal{R}}$, then $root(pos_\Lambda(t)) = f_{L'_1|i:L'_2}$ such that L'_1 and L'_2 do not contain negative annotations. Thus, the conclusion follows and condition (2) is fulfilled.

(iii) Let $t = f_{L_1|-i:L_2}(t_1,\ldots,t_k)$, $i > 0$, $q = \Lambda$, and $s = f_{L_1@-j|L_2}(t_1,\ldots,t_k)$. In this case, it is clear that index $-i$ would not appear in $pos_\Lambda(t)$. Thus, since $neg_\Lambda(t') = t'$ for any term t', $\langle pos_\Lambda(t),\Lambda\rangle \xrightarrow{\sharp}{}^{=}_{\varphi^{\mathbb{II}},\mathcal{R}^{\mathbb{II}}} \langle neg_\Lambda(pos_\Lambda(t)),\Lambda\rangle$ and condition (2) is fulfilled.

(iv) Let $t = f_{L_1|0:L_2}(t_1,\ldots,t_k) = \sigma(l')$, $erase(l') = l$ for $l \to r \in \mathcal{R}$ and substitution σ, $s = \sigma(\varphi(r))$, and $q = \Lambda$. Note that since $\mathcal{R}$ is a CS and $f \in \mathcal{D}$, $f \notin \mathcal{F}^{\varphi}_{\mathcal{R}}$ and $root(pos_\Lambda(t)) = f_{L'_1|0:L'_2}$.

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

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

First, note that it is impossible that $erase(t)$ is not a redex w.r.t. $\mathcal{R}^{\mathbb{II}}$ but $OD_{\mathcal{R}}(t) \neq \varnothing$, since there are no negative annotations in $\mathcal{R}^{\mathbb{II}}$.

Consider $erase(t) = \sigma(l)$ for $l \to r \in \mathcal{R}^{\mathbb{II}}$. By definition, $\exists l' \to r' \in \mathcal{R}$ s.t. l differs from l' in a set of variables positions, $\exists Q \subseteq \mathcal{P}os(l')$ s.t. $l = l'[\overline{x}]_Q$. Then, there is a set of rules $l_1 \to r_1,\ldots,l_n \to r_n \in \mathcal{R}^{\mathbb{II}}$ such that $l_1 = l$ and $r_n = \mathtt{quote}_\tau(r')$. Moreover, when these rules are applied sequentially to term $pos_\Lambda(t)$, they produce a term t' which differs from t in a set of symbols $\{f^\Lambda,\ldots,f^{q'}\}$ for a position $q' \in \mathcal{P}os(t)$ such that $t' = t[f^\Lambda,\ldots,f^{q'}]_{\{\Lambda,\ldots,q'\}}$. Note that since $erase(t)$ is not a redex w.r.t. $\mathcal{R}$, the sequence of rules $l_1 \to r_1,\ldots,l_n \to r_n$ will not be able to evaluate t to r_n at it will stop at some intermediate step, which is just the expression t' with symbols f^{+i}. Then, since $\mathcal{R}$ is φ-terminating, the term t' can be produced without entering in an infinite evaluation sequence. Hence, condition (4) is fulfilled.

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

Again, there is a set of rules $l_1 \to r_1,\ldots,l_n \to r_n \in \mathcal{R}^{\mathbb{II}}$ such that $l_1 = l[\overline{x}]_Q$ for a set of positions Q and l_n differs from l in a set of symbols $\{f^\Lambda,\ldots,f^{p''}\}$ for a position $p'' \in \mathcal{P}os(l)$ s.t. $p''.i = p'$ for $i \in \mathbb{N}$ and

$l_n = l[\overline{y}]_{Q'}[f^{\Lambda}, \ldots, f^{p''}]_{\{\Lambda,\ldots,p''\}}$ for another set of positions Q'. That is, these rules are auxiliary rules introduced to stepwise the pattern matching process until position p' (which is the demanded position) is reached and its evaluation started. Hence, condition (3) is fulfilled.

2

We denote by $pos(t)$ the extension of $pos_p(t)$ to include symbols f_{root} w.r.t. all positive positions in t, i.e. $root(pos(t)|_p) = f_{root\,L_1'|L_2'}$ if $p \in \mathcal{P}os_P(t) \cap \mathcal{P}os_{\mathcal{F}^{\varphi}_{\mathcal{R}}}(t)$, where $t|_p = f_{L_1|L_2}$, $L_1' = L_{1\downarrow\varphi^{\mathbb{II}}(f_{root})}$, and $L_2' = L_{2\downarrow\varphi^{\mathbb{II}}(f_{root})}$; and $root(pos(t)|_p) = f_{L_1'|L_2'}$ otherwise, where $t|_p = f_{L_1|L_2}$, $L_1' = L_{1\downarrow\varphi^{\mathbb{II}}(f)}$, and $L_2' = L_{2\downarrow\varphi^{\mathbb{II}}(f)}$.

The following lemma ensures that normalization from a term $\mathtt{quote}_{\tau}(t)$ or from a term $pos(t)$ is equivalent.

Lemma A.5 *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS and φ be a standard strategy map. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. Let $t, s \in \mathcal{T}({\mathcal{F}^{\mathbb{II}}}^{\sharp}_{\varphi^{\mathbb{II}}}, \mathcal{X}^{\sharp}_{\varphi^{\mathbb{II}}})$ s.t. $\tau = \tau(root(t))$. Then, $\langle \mathtt{quote}_{\tau}(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi^{\mathbb{II}},\mathcal{R}^{\mathbb{II}}} \langle s, \Lambda\rangle$ if and only if $\langle pos(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi^{\mathbb{II}},\mathcal{R}^{\mathbb{II}}} \langle s, \Lambda\rangle$.*

Proof. Straightforward because *pos* and $\mathtt{quote}_{\tau}$ modify the same symbols in $\mathcal{P}os_P(t) \cap \mathcal{P}os_{\mathcal{F}^{\varphi}_{\mathcal{R}}}(t)$ and, since φ is a standard strategy map, positive indices are always reduced before the root symbol. 2

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

Definition A.6 [Maximal constructor context] Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and φ be an E-strategy map. The *maximal constructor context* $C_t[\,]$ of a term $t \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ is defined as: $C_t[\,] = 2$ if $root(erase(t)) \notin \mathcal{C}$; $C_t[\,] = c(C_{t_1}[\,], \ldots, C_{t_k}[\,])$ if $root(erase(t)) = c \in \mathcal{C}$.

Theorem A.7 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that $\mathcal{R}$ is φ-terminating. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. For all $t, s \in \mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$, if $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi,\mathcal{R}} \langle s, \Lambda\rangle$, then $\langle pos(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi^{\mathbb{II}},\mathcal{R}^{\mathbb{II}}} \langle s', \Lambda\rangle$ where $C_s = C_{s'}$ and $\forall p \in minimal_{\leq}(\mathcal{P}os_{\mathcal{F}^{\mathbb{II}}-\mathcal{F}}(s'))$, $s'|_p$ is a $\overset{\sharp}{\rightarrow}$-normal form.*

Proof. (Sketch) By induction on the length of the sequence $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi,\mathcal{R}} \langle s, \Lambda\rangle$ and considering Proposition A.4 and Lemma A.5. 2

Theorem 4.6 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map such that $\mathcal{R}$ is φ-terminating. Let $\mathcal{R}^{\mathbb{II}} = (\mathcal{F}^{\mathbb{II}}, R^{\mathbb{II}})$ and $\varphi^{\mathbb{II}}$. For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in eval^{\varphi}_{\mathcal{R}}(t)$, then $s \in eval^{\varphi^{\mathbb{II}}}_{\mathcal{R}^{\mathbb{II}}}(t)$.*

Proof. By Theorem A.7 and Lemma A.5. 2

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

Definition A.8 Let φ be a strategy map over signature $\mathcal{F}$. Let $\varphi^{\mathbb{III}}$ be a strategy map over signature $\mathcal{F}^{\mathbb{III}}$. Let $t \in \mathcal{T}({\mathcal{F}^{\mathbb{III}}}^{\sharp}_{\varphi^{\mathbb{III}}}, \mathcal{X}^{\sharp}_{\varphi^{\mathbb{III}}})$. We define the translation of t into terms $\mathcal{T}(\mathcal{F}^{\sharp}_{\varphi}, \mathcal{X}^{\sharp}_{\varphi})$ as $rem(t) = s$ where $\mathcal{P}os(s) = \mathcal{P}os(t)$ and $\forall q \in \mathcal{P}os(t)$,

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

Proposition A.9 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\mathbb{III}} = (\mathcal{F}^{\mathbb{III}}, R^{\mathbb{III}})$ and $\varphi^{\mathbb{III}}$. For all $t, s \in \mathcal{T}({\mathcal{F}^{\mathbb{III}}}^{\sharp}_{\varphi^{\mathbb{III}}}, \mathcal{X}^{\sharp}_{\varphi^{\mathbb{III}}})$ without symbols* $\texttt{quote}_\tau$*, and $q \in \mathcal{P}os_A(s)$, if $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}_{\varphi^{\mathbb{III}},\mathcal{R}^{\mathbb{III}}} \langle s, q\rangle$, then $\langle rem(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^{=}_{\varphi,\mathcal{R}} \langle rem(s), q\rangle$.*

Proof. Straightforward since cases of $\overset{\sharp}{\rightarrow}$ for positive strategy annotations are only considered. 2

Theorem A.10 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\mathbb{III}} = (\mathcal{F}^{\mathbb{III}}, R^{\mathbb{III}})$ and $\varphi^{\mathbb{III}}$. For all $t, s \in \mathcal{T}({\mathcal{F}^{\mathbb{III}}}^{\sharp}_{\varphi^{\mathbb{III}}}, \mathcal{X}^{\sharp}_{\varphi^{\mathbb{III}}})$ without symbols* $\texttt{quote}_\tau$*, and $q \in \mathcal{P}os_A(s)$, if $\langle t, \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi^{\mathbb{III}},\mathcal{R}^{\mathbb{III}}} \langle s, \Lambda\rangle$, then $\langle rem(t), \Lambda\rangle \overset{\sharp}{\rightarrow}^{!}_{\varphi,\mathcal{R}} \langle s', q\rangle$ where $C_s = C_{s'}$ and $\forall p \in minimal_{\leq}(\mathcal{P}os_{\mathcal{F}^{\mathbb{III}}-\mathcal{F}}(s))$, $s'|_p$ is a $\overset{\sharp}{\rightarrow}$-normal form.*

Proof. Similar to Theorem A.7 but using Proposition A.9 and without any condition on termination. 2

Theorem 4.7 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. Let $\mathcal{R}^{\mathbb{III}} = (\mathcal{F}^{\mathbb{III}}, R^{\mathbb{III}})$ and $\varphi^{\mathbb{III}}$. For all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, if $s \in eval^{\varphi^{\mathbb{III}}}_{\mathcal{R}^{\mathbb{III}}}(t)$, then $s \in eval^{\varphi}_{\mathcal{R}}(t)$.*

Proof. By Theorem A.10 and Lemma A.5. 2

Finally, termination is preserved by the transformation.

Theorem 4.8 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a CS and φ be a standard E-strategy map. $\mathcal{R}$ is φ-terminating iff $\mathcal{R}^{\mathbb{III}}$ is $\varphi^{\mathbb{III}}$-terminating.*

Proof. By Propositions A.4 and A.9. 2