

Correct and complete (positive) strategy annotations for OBJ^{*}

María Alpuente, Santiago Escobar, and Salvador Lucas

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

Abstract. Strategy annotations are used in several programming languages as replacement restrictions aimed at improving efficiency and/or reducing the risk of nontermination. Unfortunately, restrictions of rewriting can have a negative impact in the ability to compute normal forms. In this paper, we first ascertain/clarify the conditions ensuring correctness and completeness (regarding normalization) of computing with strategy annotations. Then, we define a program transformation methodology for (correct and) complete evaluations which applies to OBJ-like languages.

Keywords: Declarative programming, OBJ, strategy annotations.

1 Introduction

Strategy annotations are used in the OBJ family of languages¹ (OBJ2 [FGJM85], OBJ3 [GWMFJ00], CafeOBJ [FN97], and Maude [CELM96]) to *avoid nontermination* ([GWMFJ00], Section 2.4.4).

Example 1. The following OBJ3 program:

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

specifies an *explicit* strategy annotation for the list constructor `cons` which disables replacements on the second argument. In this way, we can ensure that computations with this program are terminating (see Example 3 below for a formal justification of this claim).

Termination of rewriting under strategy annotations has been studied in a number of papers [FGK01,Luc01a,Luc01b]. Unfortunately, using restrictions of rewriting may cause *incompleteness*, i.e., normal forms of input expressions could be unreachable by restricted computation. For instance, using the program of Example 1 we are *not* able to compute the list of integers between

* Work partially supported by CICYT TIC2001-2705-C03-01, Acciones Integradas HI 2000-0161, HA 2001-0059, HU 2001-0019, and Generalitat Valenciana GV01-424.

¹ As in [GWMFJ00], by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

one and five that corresponds to the evaluation of $\mathbf{first}(s(0), \mathbf{from}(0))$. As we show below, the evaluation of this expression stops yielding the term $\mathbf{cons}(0, \mathbf{first}(0, \mathbf{from}(s(0))))$. On the other hand, from the user's point of view, this must be thought of as a kind of *incorrect* evaluation also, when normal forms are expected as the result of a computation.

We show that these problems can be solved by using a program transformation while we are still able to preserve termination of computations.

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^* . An element $a \in A$ is an R -normal form, if there exists no b such that $a R b$; \mathbf{NF}_R is the set of R -normal forms. We say that b is an R -normal form of a (written $a R^! b$), if b is an R -normal form and $a R^* b$; in this case, we also say that a is R -normalizing. We say that R is *normalizing* if every $a \in A$ has an R -normal form, i.e., for all $a \in A$, there is $b \in A$ such that $a R^! b$; otherwise, a is called R -reducible. In a normalizing relation, each element $a \in A$ has (at least) one normal form. In a confluent and normalizing relation, the normal form exists and is *unique*. We say that R is *terminating* iff there is no infinite sequence $a_1 R a_2 R a_3 \dots$. Obviously, terminating relations are normalizing. Throughout the paper, \mathcal{X} denotes a countable set of variables and \mathcal{F} denotes a set of function symbols $\{f, g, \dots\}$, each having a fixed arity given by a function $ar : \mathcal{F} \rightarrow \mathbb{N}$. We denote the set of terms built from \mathcal{F} and \mathcal{X} by $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A context $C[]$ is a term from $\mathcal{T}(\mathcal{F} \cup \{\square\}, \mathcal{X})$, where \square is a new constant symbol. 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. Positions p, q, \dots are represented by chains of positive natural numbers used to address subterms of t . By Λ , we denote the empty chain. Given positions p, q , we denote its concatenation by $p.q$. If p is a position, and Q is a set of positions, $p.Q$ is the set $\{p.q \mid q \in Q\}$. By $\mathcal{Pos}(t)$ we denote the set of positions of a term t . Positions of non-variable symbols in t are denoted as $\mathcal{Pos}_{\mathcal{F}}(t)$ and $\mathcal{Pos}_{\mathcal{X}}(t)$ are the variable occurrences. 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 *substitution* is a mapping $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ which homomorphically extends to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$.

A rewrite rule is an ordered pair (l, r) , written $l \rightarrow r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $\mathcal{Var}(r) \subseteq \mathcal{Var}(l)$. The left-hand side (*lhs*) of the rule is l and the right-hand side (*rhs*) is r . 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} . \mathcal{R} is left-linear if $L(\mathcal{R})$ is a set of linear terms. Given $\mathcal{R} = (\mathcal{F}, R)$, we consider \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 $\mathcal{D} = \{f \mid f(\bar{l}) \rightarrow r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. Then, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ is the set of constructor terms. An instance $\sigma(l)$ of a *lhs* $l \in L(\mathcal{R})$ is a redex. The set of redex positions in t is $\mathcal{Pos}_{\mathcal{R}}(t)$. 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 $t \rightarrow s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some $l \rightarrow r \in R$, $p \in \mathcal{Pos}(t)$ and substitution σ . A term is a normal form if it is a \rightarrow -normal form. Let $\mathbf{NF}_{\mathcal{R}}$ be the set of normal forms of \mathcal{R} . A term t is a head-normal form if it cannot be rewritten to a redex. A TRS is terminating if \rightarrow is terminating.

3 Rewriting with syntactic replacement restrictions

A mapping $\mu : \mathcal{F} \rightarrow \mathcal{P}(\mathbb{N})$ is a *replacement map* (or \mathcal{F} -map) if $\forall f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \dots, ar(f)\}$ [Luc98a]. The inclusion ordering \subseteq on $\mathcal{P}(\mathbb{N})$ extends to an ordering \sqsubseteq on $M_{\mathcal{F}}$, the set of all \mathcal{F} -maps: $\mu \sqsubseteq \mu'$ if for all $f \in \mathcal{F}$, $\mu(f) \subseteq \mu'(f)$. In this way, $\mu \sqsubseteq \mu'$ means that μ considers less positions than μ' for reduction. We also say that μ is more restrictive than (or equally restrictive to) μ' . Given a TRS $\mathcal{R} = (\mathcal{F}, R)$, we write $M_{\mathcal{R}}$ rather than $M_{\mathcal{F}}$. The set of μ -replacing positions $\mathcal{Pos}^{\mu}(t)$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is: $\mathcal{Pos}^{\mu}(t) = \{\Lambda\}$, if $t \in \mathcal{X}$ and $\mathcal{Pos}^{\mu}(t) = \{\Lambda\} \cup \bigcup_{i \in \mu(root(t))} i. \mathcal{Pos}^{\mu}(t|_i)$,

if $t \notin \mathcal{X}$. In *context-sensitive rewriting* (CSR [Luc98a]), we (only) rewrite *replacing* redexes: t μ -rewrites to s , written $t \hookrightarrow_{\mu} s$, if $t \xrightarrow{p} s$ and $p \in \mathcal{Pos}^{\mu}(t)$. The \hookrightarrow_{μ} -normal forms are called μ -normal forms. $\mathbf{NF}_{\mathcal{R}}^{\mu}$ is the set of μ -normal forms of \mathcal{R} . Note that, except for the trivial case $\mu = \mu_{\top}$, the μ -normal forms strictly include all normal forms of \mathcal{R} . A TRS \mathcal{R} is μ -terminating if \hookrightarrow_{μ} is terminating. The canonical replacement map $\mu_{\mathcal{R}}^{can}$ is *the most restrictive replacement map ensuring that the non-variable subterms of the left-hand sides of the rules of \mathcal{R} are replacing*. Note that $\mu_{\mathcal{R}}^{can}$ is easily obtained from \mathcal{R} : $\forall f \in \mathcal{F}, i \in \{1, \dots, ar(f)\}$,

$$i \in \mu_{\mathcal{R}}^{can}(f) \quad \text{iff} \quad \exists l \in L(\mathcal{R}), p \in \mathcal{Pos}_{\mathcal{F}}(l), (\text{root}(l|_p) = f \wedge p.i \in \mathcal{Pos}_{\mathcal{F}}(l))$$

Let $CM_{\mathcal{R}} = \{\mu \in M_{\mathcal{R}} \mid \mu_{\mathcal{R}}^{can} \sqsubseteq \mu\}$ be the set of replacement maps which are less or equally restrictive than $\mu_{\mathcal{R}}^{can}$.

4 E-strategies

A *positive local strategy* (or *E-strategy*) for a k -ary symbol $f \in \mathcal{F}$ is a sequence $\varphi(f)$ of integers taken from $\{0, 1, \dots, k\}$ which are given in parentheses (see Example 1). A mapping φ that associates a local strategy $\varphi(f)$ to every $f \in \mathcal{F}$ is called an *E-strategy map* [NO01]. Nagaya describes the operational semantics of term rewriting under *E-strategy maps* as follows [Nag99]: Let \mathcal{L} be the set of all lists consisting of natural numbers. By \mathcal{L}_n , we denote the set of all lists of natural numbers not exceeding $n \in \mathbb{N}$. We use the signature $\mathcal{F}_{\mathcal{L}} = \{f_L \mid f \in \mathcal{F} \wedge L \in \mathcal{L}_{ar(f)}\}$ and labelled variables $\mathcal{X}_{\mathcal{L}} = \{x_{nil} \mid x \in \mathcal{X}\}$. An *E-strategy map* φ for \mathcal{F} is extended to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ as follows:

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

The mapping *erase* : $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ removes labellings from symbols in the obvious way. The binary relation \rightarrow_{φ} on $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}}) \times \mathbb{N}_{+}^*$ (i.e., pairs $\langle t, p \rangle$ of labelled terms t and positions p) is [NO01, Nag99]: $\langle t, p \rangle \rightarrow_{\varphi} \langle s, q \rangle$ if and only if $p \in \mathcal{Pos}(t)$ and either

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

We write $e \in L$ to denote that item e appears somewhere within the list L . Given a *E-strategy map* φ for \mathcal{F} , we define $\mu^{\varphi} \in M_{\mathcal{F}}$ as follows: for all $f \in \mathcal{F}$, $\mu^{\varphi}(f) = \{i \neq 0 \mid i \in \varphi(f)\}$. We will drop superscript φ if no confusion arises.

Example 2. The TRS \mathcal{R} :

$$\begin{array}{lll} \text{sel}(0, \mathbf{x} : \mathbf{z}) & \rightarrow \mathbf{x} & \text{first}(0, \mathbf{z}) & \rightarrow \square \\ \text{sel}(\mathbf{s}(\mathbf{x}), \mathbf{y} : \mathbf{z}) & \rightarrow \text{sel}(\mathbf{x}, \mathbf{z}) & \text{first}(\mathbf{s}(\mathbf{x}), \mathbf{y} : \mathbf{z}) & \rightarrow \mathbf{y} : \text{first}(\mathbf{x}, \mathbf{z}) \\ \text{from}(\mathbf{x}) & \rightarrow \mathbf{x} : \text{from}(\mathbf{s}(\mathbf{x})) & & \end{array}$$

together with the replacement map

$$\mu(\mathbf{s}) = \mu(:) = \mu(\text{from}) = \{1\} \quad \text{and} \quad \mu(\text{sel}) = \mu(\text{first}) = \{1, 2\}.$$

correspond to the OBJ program of Example 1 (we use $:$ and \square instead of `cons` and `nil`, respectively).

Rewriting with strategy annotations is closely related to CSR.

Theorem 1. [Luc01a] *Let \mathcal{R} be a TRS and φ be a positive E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$, and $p \in \mathcal{Pos}^{\mu}(\text{erase}(t))$ be s.t. $\text{root}(t|_p) = f_L$ for some suffix L of $\varphi(f)$. If $\langle t, p \rangle \rightarrow_{\varphi} \langle s, q \rangle$, then $q \in \mathcal{Pos}^{\mu}(\text{erase}(s))$ and $\text{erase}(t) \hookrightarrow_{\mu}^{\varphi} \text{erase}(s)$.*

Algebraic languages **OBJ2**, **OBJ3**, **CafeOBJ**, and **Maude** admit the specification of E -strategies. Symbols without an explicit local strategy are given a *default* one whose concrete shape depends on the language considered². Semantics of **OBJ** programs under a given E -strategy map φ is given by means of a mapping $eval_\varphi$ (from terms to their sets of ‘computed values’). Nakamura and Ogata describe $eval_\varphi$ for positive E -strategy maps by using the reduction relation \rightarrow_φ [NO01]: given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and a positive E -strategy map φ for \mathcal{F} , $eval_\varphi : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ is defined as $eval_\varphi(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle \varphi(t), A \rangle \rightarrow_\varphi^! \langle s, A \rangle\}$. Given an **OBJ** program P , we consider the corresponding TRS \mathcal{R} which consists of the set of rewriting rules in P and the replacement map μ^φ associated to the E -strategy map φ . A TRS \mathcal{R} is φ -terminating if, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, there is no infinite \rightarrow_φ -rewrite sequence starting from $\langle \varphi(t), A \rangle$. According to this, we can say that an **OBJ** program P is

terminating if, the corresponding TRS \mathcal{R} is φ -terminating.

The following result connects φ -termination of an **OBJ** program and termination of *CSR*.

Theorem 2. [Luc01a] *An **OBJ** program P with positive E -strategy map φ is terminating if the corresponding TRS \mathcal{R} is μ^φ -terminating.*

Termination of *CSR* has been studied in a number of papers, see [Luc02b] for an overview and comparison of the different methods for proving termination of *CSR*.

Example 3. Consider \mathcal{R} and μ as in Example 2. The μ -termination of (a superset of) \mathcal{R} is demonstrated in Example 7 of [BLR02]. Hence, by Theorem 2, the **OBJ** program of Example 1 is terminating. In fact, by using an **OBJ3** interpreter for evaluating a (potentially) non-terminating expression such as `from(0)`, we obtain the following outcome (we use the version 2.0 of the SRI’s **OBJ3** interpreter³):

```
=====
obj EXAMPLE
=====
reduce in EXAMPLE : from(0)
rewrites: 1
result LNat: cons(0,from(s(0)))
```

5 Correctness and completeness

A *rewriting semantics* for a TRS $\mathcal{R} = (\mathcal{F}, R)$ is a mapping $S : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ such that, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $s \in S(t)$, $t \rightarrow_{\mathcal{R}}^* s$ [Luc01c]. Users of rewriting-based engines (e.g., **OBJ** programmers) are usually interested in computing *normal forms*. Thus, regarding such normal forms semantics (nf in [Luc01c]), we say that a different rewriting semantics (e.g., the one described by $eval_\varphi$) is:

correct if it only returns normal forms (e.g., $eval_\varphi(t) \subseteq nf(t)$ for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$), and **complete** if, whenever a term has a normal form, the semantics returns such a normal form (e.g., $nf(t) \subseteq eval_\varphi(t)$ for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$).

Computations with **OBJ** programs produce (by means of $eval_\varphi$) expressions called *E-normal forms* (*ENFs*). Such terms are not generally normal forms (i.e., terms without redexes). In fact, we have the following:

Theorem 3. [Luc01a] *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. If $s \in eval_\varphi(t)$, then s is a μ -normal form of t .*

² For instance, in **Maude**, the default local strategy associated to a k -ary symbol f , is $(1\ 2\ \dots\ k\ 0)$, see [Eke98].

³ Available at <http://www.kindsoftware.com/products/opensource/obj3/OBJ3/>.

Computing values (i.e., terms built from constructor symbols) is the essential task in functional programming. Regarding normalization of terms, if we restrict ourselves to values (i.e., we consider the semantics `eval` in [Luc01c]), *CSR* is powerful enough to compute them. Given TRS $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and $\mathcal{B} \subseteq \mathcal{C}$, we let $\mu_{\mathcal{R}}^{\mathcal{B}}$ to be $\mu_{\mathcal{R}}^{\mathcal{B}}(c) = \{1, \dots, ar(c)\}$ for all $c \in \mathcal{B}$ and $\mu_{\mathcal{R}}^{\mathcal{B}}(f) = \mu_{\mathcal{R}}^{can}(f)$ if $f \in \mathcal{F} - \mathcal{B}$. Note that $\mu_{\mathcal{R}}^{\mathcal{B}} \in CM_{\mathcal{R}}$.

Theorem 4. [Luc98a] *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear TRS, $\mathcal{B} \subseteq \mathcal{C}$ and $\mu \in M_{\mathcal{F}}$ be such that $\mu_{\mathcal{R}}^{\mathcal{B}} \sqsubseteq \mu$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{B}, \mathcal{X})$. Then, $t \rightarrow^* \delta$ iff $t \hookrightarrow_{\mu}^* \delta$.*

Theorem 4 is very easy to use in sorted signatures (as in **OBJ** programs), since, given a term t (of sort τ), we are able to establish the set of constructors $\mathcal{B} \subseteq \mathcal{C}$ which should be considered (namely, the constructor symbols of sort τ).

Unfortunately, Theorem 4 does not directly apply to **OBJ** computations, as they must obey the order of evaluation expressed by the strategy annotations. However, we have the following.

Theorem 5. *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS and $\mathcal{B} \subseteq \mathcal{C}$. Let φ be a positive *E*-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0 and \mathcal{R} is φ -terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\delta \in \mathcal{T}(\mathcal{B}, \mathcal{X})$. If $\mu_{\mathcal{R}}^{\mathcal{B}} \sqsubseteq \mu^{\varphi}$, then $t \rightarrow^! \delta$ iff $\delta \in eval_{\varphi}(t)$.*

For instance, φ can be used to compute the *value* of every expression of the sort **Nat** in the **OBJ** program of Example 1 (since $\mu_{\mathcal{R}}^{\{0, s\}} \sqsubseteq \mu^{\varphi}$). This is not true for values of the sort **LNat**.

Example 4. The evaluation of `first(s(0), from(0))` using the program of Example 1 yields:

```
=====
obj EXAMPLE
=====
reduce in EXAMPLE : first(s(0), from(0))
rewrites: 2
result LNat: cons(0, first(0, from(s(0))))
```

Note that `cons(0, first(0, from(s(0))))` is not a normal form.

Nevertheless, the strategy annotations can be fixed in such a way that *E*-normal forms are ensured to be normal forms, which guarantees correctness of **OBJ** computations:

1. Nagaya shows that if $\varphi(f)$ contains all indices $0, 1, \dots, ar(f)$ for each symbol $f \in \mathcal{F}$, and $\varphi(f)$ ends in 0 for defined symbols $f \in \mathcal{D}$, then $eval_{\varphi}(t) \subseteq NF_{\mathcal{R}}$ for every term t (Theorem 6.1.12 in [Nag99]).
2. Nakamura and Ogata show that normal forms can be obtained in **OBJ**'s computations as follows: given a strategy map φ ensuring that terms in $eval_{\varphi}(t)$ are head-normal forms (for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$), any φ' given by $\varphi'(f) = \varphi(f) ++ (i_1 \dots i_n)$ for all symbol $f \in \mathcal{F}$ (where $++$ appends two lists, and $\{i_1, \dots, i_n\} = \{1, \dots, ar(f)\} - \mu_{\varphi}(f)$) ensures that terms in $eval_{\varphi'}(t)$ are normal forms (Theorem 3.2 in [NO01]).

For instance, φ as given in Example 1 is head-normalizing (this follows from Theorem 3 and the fact that μ -normal forms of a left-linear TRS \mathcal{R} are head-normal forms whenever $\mu \in CM_{\mathcal{R}}$, see Theorem 8 in [Luc98a]). Moreover, since the **OBJ** program of Example 1 is φ -terminating, $eval_{\varphi}(t)$ is not empty for any term t . Thus, the evaluation of every term t yields a head-normal form of t (i.e., φ is correct regarding the head-normalization semantics). Unfortunately when using φ' instead of φ , we are *not* able to compute (head-)normal forms anymore: the head-normalizing behavior of φ gets lost.

Example 5. Consider the program of Example 1 with $\varphi'(\cdot) = (1\ 0\ 2)$ and $\varphi'(f) = \varphi(f)$ for every other symbol f . Consider again the evaluation of $t = \text{first}(s(0), \text{from}(0))$:

```
=====
obj EXAMPLE-INF
=====
```

```

reduce in EXAMPLE-INF : first(s(0),from(0))
Error: Value stack overflow.
Fast links are on: do (si::use-fast-links nil) for debugging
Error signalled by CATCH.
Broken at OBJ3. Type :H for Help.

```

The problem is that the evaluation of t , i.e., the evaluation of

$$\varphi(t) = \text{first}_{(1\ 2\ 0)}(\text{s}_{(1)}(0_{\text{nil}}), \text{from}_{(1\ 0)}(0_{\text{nil}}))$$

using \rightarrow_{φ} does not terminate (we underline the redex contracted at each step):

```

(first (1 2 0) (s (1) (0 nil)), from (1 0) (0 nil)), A
→φ (first (2 0) (s (1) (0 nil)), from (1 0) (0 nil)), 1
→φ (first (2 0) (s nil (0 nil)), from (1 0) (0 nil)), 1.1
→φ (first (2 0) (s nil (0 nil)), from (1 0) (0 nil)), 1
→φ (first (2 0) (s nil (0 nil)), from (1 0) (0 nil)), A
→φ (first (0) (s nil (0 nil)), from (1 0) (0 nil)), 2
→φ (first (0) (s nil (0 nil)), from (0) (0 nil)), 2.1
→φ (first (0) (s nil (0 nil)), from (0) (0 nil)), 2
→φ (first (0) (s nil (0 nil)), 0 nil: (1 0 2) from (1 0) (s (1) (0 nil))), 2
→φ+ (first (0) (s nil (0 nil)), 0 nil: (2) from (1 0) (s (1) (0 nil))), 2
→φ (first (0) (s nil (0 nil)), 0 nil: nil from (1 0) (s (1) (0 nil))), 2.2
→φ+ (first (0) (s nil (0 nil)), 0 nil: nil from (0) (s nil (0 nil))), 2.2
→φ (first (0) (s nil (0 nil)), 0 nil: nil s nil (0 nil): (1 0 2) from (1 0) (s (1) (s nil (0 nil)))), 2.2
→φ ...

```

Using the OBJ3 interpreter (as above), this nonterminating sequence shows as a ‘stack overflow’.

Thus, the φ -termination of \mathcal{R} (see Example 3) does *not* ensure φ' -normalization as the previous results by Nagaya, and Nakamura-Ogata may suggest. Moreover, $eval_{\varphi}$ was able to obtain head-normal forms that $eval_{\varphi'}$ does *not* obtain (compare the evaluation of t in Examples 4 and 5). Example 5 also shows that requiring φ -termination in Theorem 5 is essential for ensuring correct and complete evaluations (note that \mathcal{R} and φ' in Example 5 fulfill all requirements in Theorem 5, except for φ' -termination).

In the following section, we propose a solution to (partially) overcome this problem which is based on program transformation.

6 Program transformations for complete evaluations

The discussion and examples in the previous section show that we need to isolate the replacement restrictions which are needed to achieve the head-evaluation of a term t from the restrictions which are needed to get them within a constructor context $C[\] \in \mathcal{T}(\mathcal{B} \cup \{\square\}, \mathcal{X})$ for some $\mathcal{B} \subseteq \mathcal{C}$. Below the outermost defined symbols, reductions should be performed only up to head-evaluation. The key idea for the transformation is to introduce a set \mathcal{B}' of fresh constructor symbols having no replacement restrictions: they are renamed versions c' of the original constructors $c \in \mathcal{B}$ such that $\mu(c') = \{1, \dots, ar(c')\}$ (disregarding $\mu(c)$). In practice, we only need (and want) to fix the sort of input expressions we want to evaluate for obtaining the ‘interesting’ constructor terms in $\mathcal{T}(\mathcal{B}, \mathcal{X})$. Assume that (as usual in OBJ programs) symbols $f \in \mathcal{F}$ are sorted by: $f : \tau_1 \times \dots \times \tau_k \rightarrow \tau$. The *sort* of f is $sort(f) = \tau$. Sorts of arguments of f are gathered in $sortarg(f) = \{\tau_1, \dots, \tau_k\}$. Variables $x \in \mathcal{X}$ also have a sort, $sort(x)$. We also assume that all terms are well sorted everywhere. The sort of a term t is the sort of its outermost symbol. Given a sort τ , let $\mathcal{C}_{\tau}^* \subseteq \mathcal{C}$ be the set of constructor symbols that can be found in constructor terms of sort τ . For instance, $\mathcal{C}_{\text{Nat}}^* = \{0, \text{s}\}$ and $\mathcal{C}_{\text{LNat}}^* = \{0, \text{s}, \square, :\}$.

The renaming of constructor symbols $c \in \mathcal{C}_{\tau}^*$ into new constructor symbols $c' \in \mathcal{C}'$ is performed by the rules

$$\text{quote}_{sort(c)}(c(x_1, \dots, x_k)) \rightarrow c'(\text{quote}_{sort(x_1)}(x_1), \dots, \text{quote}_{sort(x_k)}(x_k))$$

Let $Quote$ be the set containing all these symbols: $Quote = \{\mathbf{quote}_{\tau'} \mid \exists c \in \mathcal{C}_{\tau}^*, \text{sort}(c) = \tau'\}$. The evaluation of a term t would proceed by reducing $\mathbf{quote}_{\text{sort}(t)}(t)$. The obtained value is built by using symbols in \mathcal{C}' only. The introduction of such new symbols could be avoided by appropriately distinguishing the two different kinds of constructor symbols (with and without replacement restrictions). However, this is not available in current implementations of **OBJ** interpreters. Nevertheless, we can solve the problem by introducing a postprocessing that recovers the old names of constructor symbols after the evaluation of the initial expression: New symbols $\mathbf{unquote}_{\tau} : \tau \rightarrow \tau$ are used to reverse the renamings. Then, for each constant $b \in \mathcal{C}_{\tau}^*$, we add a rule

$$\mathbf{unquote}_{\text{sort}(b)}(b') \rightarrow b$$

For each $c \in \mathcal{C}_{\tau}^*$ such that $c : \tau_1 \times \dots \times \tau_k \rightarrow \tau'$, $k > 0$, and $\mu^{\varphi}(c) = \{1, \dots, k\}$, we add a rule

$$\mathbf{unquote}_{\tau'}(c'(x_1, \dots, x_k)) \rightarrow c(\mathbf{unquote}_{\tau_1}(x_1), \dots, \mathbf{unquote}_{\tau_k}(x_k))$$

Finally, for each $c \in \mathcal{C}_{\tau}^*$ such that $c : \tau_1 \times \dots \times \tau_k \rightarrow \tau'$, $k > 0$, and $\mu^{\varphi}(c) \neq \{1, \dots, k\}$, we consider a new symbol $f_c : \tau_1 \times \dots \times \tau_k \rightarrow \tau'$; we add two rules

$$\begin{aligned} \mathbf{unquote}_{\tau'}(c'(x_1, \dots, x_k)) &\rightarrow f_c(\mathbf{unquote}_{\tau_1}(x_1), \dots, \mathbf{unquote}_{\tau_k}(x_k)) \\ f_c(x_1, \dots, x_k) &\rightarrow c(x_1, \dots, x_k) \end{aligned}$$

We collect these new symbols together in a new set $Unquote$. Denote the TRS obtained from joining these rules together with those of \mathcal{R} as $E_{\tau}(\mathcal{R})$. We also extend the (original) E -strategy φ : let $\varphi' = Emap_{\tau}(\varphi)$ as follows: $\varphi'(f) = \varphi(f)$ if $f \in \mathcal{F}$, $\varphi'(c') = (1 \dots ar(c'))$ if $c' \in \mathcal{C}'$, $\varphi'(\mathbf{quote}_{\tau'}) = \varphi'(\mathbf{unquote}_{\tau'}) = (1 \ 0)$ for all sort τ' , and $\varphi'(f_c) = (1 \dots ar(c) \ 0)$ for each $c \in \mathcal{C}_{\tau}^*$ such that $\mu^{\varphi}(c) \neq \{1, \dots, ar(c)\}$. In the following results, $eval_{\varphi'}$ uses φ' and $E_{\tau}(\mathcal{R})$ for evaluating terms ($eval_{\varphi'}$ uses φ and \mathcal{R} , as above). Our transformation is correct in a very general setting.

Theorem 6 (Correctness). *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $\text{sort}(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_{\tau}(\mathcal{R})$ and $\varphi' = Emap_{\tau}(\varphi)$. If $\delta \in eval_{\varphi'}(\mathbf{unquote}_{\tau}(\mathbf{quote}_{\tau}(t)))$, then $t \rightarrow_{\mathcal{R}}^* \delta$.*

Thus, no ‘unexpected’ value can be obtained when evaluating $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ of sort τ as $\mathbf{unquote}_{\tau}(\mathbf{quote}_{\tau}(t))$. Moreover, no computed value obtained by using φ and \mathcal{R} gets lost with $Emap_{\tau}(\varphi)$ and $E_{\tau}(\mathcal{R})$.

Theorem 7. *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $\text{sort}(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_{\tau}(\mathcal{R})$ and $\varphi' = Emap_{\tau}(\varphi)$. If $\delta \in eval_{\varphi}(t)$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_{\tau}(\mathbf{quote}_{\tau}(t)))$.*

Completeness of the transformation (regarding the computation of constructor terms) requires some additional conditions.

Theorem 8 (Completeness). *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0, $\mu^{\varphi} \in CM_{\mathcal{R}}$, and \mathcal{R} is φ -terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $\text{sort}(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_{\tau}(\mathcal{R})$ and $\varphi' = Emap_{\tau}(\varphi)$. If $t \rightarrow_{\mathcal{R}}^* \delta$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_{\tau}(\mathbf{quote}_{\tau}(t)))$.*

Note that, in contrast to Theorem 5, now we can start with any E -strategy map φ such that $\mu^{\varphi} \in CM_{\mathcal{R}}$: The transformation introduces the necessary ‘relax’ of the replacement restrictions (via $Emap_{\tau}(\varphi)$).

Example 6. The following **OBJ3** program:

```
obj EXAMPLE-STR is
  sorts Nat LNat .
```

```

ops 0 0' : -> Nat .
ops s s' : Nat -> Nat [strat (1 0)] .
ops nil nil' : -> LNat .
op cons : Nat LNat -> LNat [strat (1 0)] .
ops cons' fcons : Nat LNat -> LNat [strat (1 2 0)] .
op from : Nat -> LNat [strat (1 0)] .
op sel : Nat LNat -> Nat [strat (1 2 0)] .
op first : Nat LNat -> LNat [strat (1 2 0)] .
ops quote unquote : Nat -> Nat [strat (1 0)] .
ops quote' unquote' : LNat -> LNat [strat (1 0)] .
vars X Y : Nat .
var Z : LNat .
eq sel(s(X),cons(Y,Z)) = sel(X,Z) .
eq sel(0,cons(X,Z)) = X .
eq first(0,Z) = nil .
eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
eq from(X) = cons(X,from(s(X))) .
eq quote(0) = 0' .
eq quote'(cons(X,Z)) = cons'(quote(X),quote'(Z)) .
eq quote'(nil) = nil' .
eq quote(s(X)) = s'(quote(X)) .
eq unquote(0') = 0 .
eq unquote(s'(X)) = s(unquote(X)) .
eq unquote'(nil') = nil .
eq unquote'(cons'(X,Z)) = fcons(unquote(X),unquote'(Z)) .
eq fcons(X,Z) = cons(X,Z) .

```

endo

is the transformed version of the OBJ program of Example 1. Now, the evaluation of term `unquote'(quote'(first(s(0),from(0))))` yields:

```

=====
obj EXAMPLE-STR
=====
reduce in EXAMPLE-STR : unquote'(quote'(first(s(0),from(0))))
rewrites: 11
result LNat: cons(0,nil)

```

Note the difference between ‘unquoting’ rules for symbols `s'` and `cons'`. The unquoting of `cons'` is indirect; the obvious short-cut:

$$\text{unquote}(\text{cons}'(X,Z)) = \text{cons}(\text{unquote}(X),\text{unquote}'(Z))$$

in the program of Example 6 does not work: the reason is that after applying this rule, *the second argument of cons remains non-replacing*. For instance, in contrast to Example 6, the evaluation of `unquote'(quote'(first(s(0),from(0))))` would yield

$$\text{cons}(0,\text{unquote}'(\text{nil}'))$$

This is solved by introducing the intermediate defined symbol `fcons` which first evaluate its arguments (thus performing the renaming) and then reduces to `cons`. In this sense, it is also crucial the *explicit* annotation (1 2 0) for symbol `fcons`; otherwise, the interpreter could associate a default strategy which does not permit the renamings (for instance, OBJ3 associates the strategy (0 1 2 0) to `fcons`; with this default annotation, we would also obtain `cons(0,unquote'(nil'))` instead of the desired value).

Unfortunately, the previous transformation does not preserve termination of the original program.

Example 7. (Continuing Example 3) Consider now the evaluation of `from(0)` using the program of Example 6. For input call `quote'(from(0))`, we obtain the following:

```

=====

```

```

obj EXAMPLE-STR
=====
reduce in EXAMPLE-STR : quote'(from(0))
Error: Value stack overflow.
Fast links are on: do (si::use-fast-links nil) for debugging
Error signalled by CATCH.
Broken at OBJ3. Type :H for Help.

```

This is in contrast to the terminating evaluation (w.r.t. the program of Example 1) showed in Example 3.

6.1 Preserving completeness and termination

Example 7 shows that the annotation $\varphi'(\mathbf{quote}_\tau) = (1 \ 0)$ may cause non-termination. We can try to avoid this problem by restricting the E -strategy for \mathbf{quote}_τ to (0) . In this case, however, we need to add new rules for enabling the evaluation in some alternative way. In [Luc97] we have introduced a program transformation which is able to achieve a similar effect. In the following, by an outermost (occurrence of a) defined symbol in a term t , we mean a defined symbol which only has constructor symbols above it in t . Now, the new constructors are introduced in computations by the contraction of redexes of outermost defined symbols f . Thus, we add new defined symbols f' which will show up when these outermost defined f symbols emerge, and new rules for defining these symbols. The new rules $f'(\bar{l}) \rightarrow r'$ come from the original ones $f(\bar{l}) \rightarrow r$ as follows: occurrences of outermost defined symbols g in r are renamed in r' as g' ; occurrences of constructor symbols c above those g in r are renamed in r' as c' ; occurrences of variables x in r which only have constructor symbols above them are marked as $\mathbf{quote}_{\mathit{sort}(x)}(x)$ in r' . Symbols \mathbf{quote}_τ are also intended to rename outermost defined symbols f (of sort τ) as their alias f' (of the same sort). In order to simplify the transformation, it is tempting not to take care of the number of extra rules which are added to the transformed TRS. Unfortunately, this may unnecessarily cause non-termination.

Example 8. Consider the rule

$$\mathbf{from}(\mathbf{x}) \rightarrow \mathbf{x}:\mathbf{from}(\mathbf{s}(\mathbf{x}))$$

of our running example. Then, we would introduce the rule:

$$\mathbf{from}'(\mathbf{x}) \rightarrow \mathbf{quote}(\mathbf{x}):'\mathbf{from}'(\mathbf{s}(\mathbf{x}))$$

For example, in the evaluation of $t = \mathbf{first}(\mathbf{s}(0), \mathbf{from}(0))$ in Example 4, the symbol \mathbf{from} does *not* emerge as outermost: roughly speaking, the only possibility is that either the right-hand side of a rule defining \mathbf{first} contains a variable of sort \mathbf{LNat} having only constructor symbols above, or that \mathbf{from} is outermost in some right-hand side. This does not happen in our example. Thus, we do not need the rule which would introduce *non-termination* since, now, $\mu(:') = \{1, 2\}$. Indeed, this rule does *not* add computational power to the transformation. For this reason, we perform a more accurate analysis of the required additional rules.

The following notations are auxiliary [Luc97]: Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,

- $CVar(t) = \{x \in Var(t) \mid \exists p \in Pos(t), t|_p = x \wedge \forall q < p, root(t|_q) \in \mathcal{C}\}$ is the set of *constructor variables* of t , i.e., variables of t having a maximal proper prefix which only points to constructor symbols. We also use $\mathcal{C}_\tau = \{c \in \mathcal{C} \mid sort(c) = \tau\}$.
- The set of possible sorts for symbols arising by instantiation of a constructor variable x is $CVSort(sort(x))$ where, given a sort τ ,
$$CVSort(\tau) = \{\tau\} \cup \bigcup_{\substack{c \in \mathcal{C}_\tau \\ \tau' \in sortarg(c)}} CVSort(\tau')$$
- $Vouter(t) = \bigcup_{x \in CVar(t)} \{f \in \mathcal{D} \mid sort(f) \in CVSort(sort(x))\}$ are the defined symbols which can root the subterms introduced in t by instantiation of constructor variables of t (that is, which emerge as outermost in t after instantiation).

Example 9. Consider the term $t = \mathbf{y}:\mathbf{first}(\mathbf{x}, \mathbf{z})$, where $\mathit{sort}(\mathbf{y}) = \mathbf{Nat}$ and $\mathbf{first} : \mathbf{Nat} \times \mathbf{LNat} \rightarrow \mathbf{LNat}$. Then,

- $CVar(t) = \{\mathbf{y}\}$; note that $\mathit{sort}(\mathbf{y}) = \mathbf{Nat}$.
- $CVSort(\mathbf{Nat}) = \{\mathbf{Nat}\}$ and

$$\begin{aligned} CVSort(\mathbf{LNat}) &= \{\mathbf{LNat}\} \cup \bigcup_{\substack{c \in \{\square, :\} \\ \tau' \in \mathit{sortarg}(c)}} CVSort(\tau') \\ &= \{\mathbf{LNat}\} \cup CVSort(\mathbf{Nat}) \cup CVSort(\mathbf{LNat}) \\ &= \{\mathbf{LNat}, \mathbf{Nat}\} \end{aligned}$$

- $Vouter(t) = \{f \in \mathcal{D} \mid \mathit{sort}(f) \in CVSort(\mathit{sort}(\mathbf{y}))\} = \{f \in \mathcal{D} \mid \mathit{sort}(f) \in \{\mathbf{Nat}\}\} = \{\mathbf{sel}\}$.

Given a TRS $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and $f \in \mathcal{D}$,

- $outrhs_{\mathcal{R}}(f) \subseteq \mathcal{D}$ contains the outermost defined symbols in *rhs*'s of the f -rules: $outrhs_{\mathcal{R}}(f) = \bigcup_{f(\bar{t}) \rightarrow r \in R} \{root(r|_p) \in \mathcal{D} \mid \forall q < p. root(r|_q) \notin \mathcal{D}\}$.
- $Vrhs_{\mathcal{R}}(f) \subseteq \mathcal{D}$ is the set of outermost defined symbols which can appear by instantiation of constructor variables in *rhs*'s of the f -rules: $Vrhs_{\mathcal{R}}(f) = \bigcup_{f(\bar{t}) \rightarrow r \in R} Vouter(r)$.
- $newouter_{\mathcal{R}}(f) = outrhs_{\mathcal{R}}(f) \cup Vrhs_{\mathcal{R}}(f)$.

Example 10. Consider the TRS \mathcal{R} of Example 2 (assume the sorts as given in the signature of the original OBJ program of Example 1). We have:

- $outrhs_{\mathcal{R}}(\mathbf{sel}) = \{\mathbf{sel}\}$ and $outrhs_{\mathcal{R}}(\mathbf{first}) = \{\mathbf{first}\}$.
- $Vrhs_{\mathcal{R}}(\mathbf{sel}) = Vouter(\mathbf{x}) \cup Vouter(\mathbf{sel}(\mathbf{x}, \mathbf{z})) = Vouter(\mathbf{x}) = \{\mathbf{sel}\}$ (note that $\mathit{sort}(\mathbf{x}) = \mathbf{Nat}$) and $Vrhs_{\mathcal{R}}(\mathbf{first}) = Vouter(\square) \cup Vouter(\mathbf{y}:\mathbf{first}(\mathbf{x}, \mathbf{z})) = Vouter(\mathbf{y}:\mathbf{first}(\mathbf{x}, \mathbf{z})) = \{\mathbf{sel}\}$ (see Example 9)
- $newouter_{\mathcal{R}}(\mathbf{sel}) = outrhs_{\mathcal{R}}(\mathbf{sel}) \cup Vrhs_{\mathcal{R}}(\mathbf{sel}) = \{\mathbf{sel}\}$ and $newouter_{\mathcal{R}}(\mathbf{first}) = outrhs_{\mathcal{R}}(\mathbf{first}) \cup Vrhs_{\mathcal{R}}(\mathbf{first}) = \{\mathbf{first}, \mathbf{sel}\}$.

In contrast to transformation E_{τ} , here we are mainly interested in evaluating term $f(\bar{t})$ for a given defined symbol $f \in \mathcal{D}$. Given $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and $f \in \mathcal{D}$, we let $\mathcal{D}_{\mathcal{R}}^f \subseteq \mathcal{D}$ be:

$$\mathcal{D}_{\mathcal{R}}^f = \{f\} \cup \bigcup_{g \in newouter_{\mathcal{R}}(f)} \mathcal{D}_{\mathcal{R}}^g$$

$\mathcal{D}_{\mathcal{R}}^f$ contains the outermost defined symbols which arise when a (well sorted) f -rooted term $f(\bar{t})$ is arbitrarily rewritten (see [Luc97]). In practice, since the definition of $\mathcal{D}_{\mathcal{R}}^f$ is mutually recursive, we must consider all possible equations

$$\begin{aligned} \mathcal{D}_{\mathcal{R}}^{f_1} &= \{f_1\} \cup \bigcup_{g \in newouter_{\mathcal{R}}(f_1)} \mathcal{D}_{\mathcal{R}}^g \\ &\vdots \\ \mathcal{D}_{\mathcal{R}}^{f_n} &= \{f_n\} \cup \bigcup_{g \in newouter_{\mathcal{R}}(f_n)} \mathcal{D}_{\mathcal{R}}^g \end{aligned}$$

(where $f_1 = f$ and f_2, \dots, f_n are all the defined symbols successively occurring in $newouter_{\mathcal{R}}(f_1) \cup \dots \cup newouter_{\mathcal{R}}(f_n)$) and compute the (least) solutions $\mathcal{D}_{\mathcal{R}}^{f_1}, \dots, \mathcal{D}_{\mathcal{R}}^{f_n}$ by using fixpoint techniques (see [Luc98b] for the technical details).

Example 11. (Continuing Example 10) Since $newouter_{\mathcal{R}}(\mathbf{first}) = \{\mathbf{first}, \mathbf{sel}\}$ and $newouter_{\mathcal{R}}(\mathbf{sel}) = \{\mathbf{sel}\}$, we have the system:

$$\begin{aligned} \mathcal{D}_{\mathcal{R}}^{\mathbf{first}} &= \{\mathbf{first}\} \cup \mathcal{D}_{\mathcal{R}}^{\mathbf{sel}} \cup \mathcal{D}_{\mathcal{R}}^{\mathbf{first}} \\ \mathcal{D}_{\mathcal{R}}^{\mathbf{sel}} &= \{\mathbf{sel}\} \cup \mathcal{D}_{\mathcal{R}}^{\mathbf{sel}} \end{aligned}$$

having a simple solution: $\mathcal{D}_{\mathcal{R}}^{\mathbf{first}} = \{\mathbf{first}, \mathbf{sel}\}$ and $\mathcal{D}_{\mathcal{R}}^{\mathbf{sel}} = \{\mathbf{sel}\}$. Note that $\mathbf{from} \notin \mathcal{D}_{\mathcal{R}}^{\mathbf{first}}$ and $\mathbf{from} \notin \mathcal{D}_{\mathcal{R}}^{\mathbf{sel}}$

The set $ev^f(\mathcal{F}, \mathcal{X})$ of terms is given as follows: (1) $\mathcal{X} \subseteq ev^f(\mathcal{F}, \mathcal{X})$, (2) $g(\bar{t}) \in ev^f(\mathcal{F}, \mathcal{X})$ if $g \in \mathcal{D}_{\mathcal{R}}^f$, and (3) $c(t_1, \dots, t_k) \in ev^f(\mathcal{F}, \mathcal{X})$ if $c \in \mathcal{C}_{sort(f)}^*$ and $t_1, \dots, t_k \in ev^f(\mathcal{F}, \mathcal{X})$. If we do not require (1) (and change the inductive case (3) accordingly), then we are defining the set $gev^f(\mathcal{F}, \mathcal{X})$. Roughly speaking, if we rewrite on a term $t = g(\bar{t})$ for some $g \in \mathcal{D}_{\mathcal{R}}^f$, then every possible reduct of t belongs to $ev^f(\mathcal{F}, \mathcal{X})$. If t is ground, then we only need to consider $gev^f(\mathcal{F}, \mathcal{X})$.

We define now the program transformation. First, we give the new signature.

Definition 1. Given a TRS $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and $f \in \mathcal{D}$, $\mathcal{F}^f = \mathcal{F} \uplus \mathcal{D}' \uplus \mathcal{C}' \uplus \text{Quote} \uplus \text{Unquote}$, where: $c' \in \mathcal{C}' \Leftrightarrow c \in \mathcal{C}_{sort(f)}^* \wedge ar(c') = ar(c)$ and $g' \in \mathcal{D}' \Leftrightarrow g \in \mathcal{D}_{\mathcal{R}}^f \wedge ar(g') = ar(g)$. *Quote* and *Unquote* are as above.

The transformation introduces rules to deal with the different symbols that we consider, accordingly to the informal description above.

Definition 2 (Transformation V). Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and $f \in \mathcal{D}$. Then $V^f(\mathcal{R}) = (\mathcal{F}^f, R \cup S \cup Q \cup U)$, where:

- $S = \{g'(\bar{t}) \rightarrow \kappa^f(r) \mid g(\bar{t}) \rightarrow r \in R \wedge g \in \mathcal{D}_{\mathcal{R}}^f\}$, where
 $\kappa^f(x) = \text{quote}_{sort(x)}(x)$, for $x \in \mathcal{X}$, $\kappa^f(g(\bar{t})) = g'(\bar{t})$ if $g \in \mathcal{D}_{\mathcal{R}}^f$, and $\kappa^f(c(\bar{t})) = c'(\kappa^f(\bar{t}))$ if $c \in \mathcal{C}$.
- Rules in Q define symbols quote_{τ} in order to rename external constructors $c \in \mathcal{C}_{sort(f)}^*$ of sort τ to constructors $c' \in \mathcal{C}'$ (of the same sort), and outermost application of $g \in \mathcal{D}_{\mathcal{R}}^f$ to outermost applications of the corresponding $g' \in \mathcal{D}'$.

$$Q = \{\text{quote}_{sort(c)}(c(x_1, \dots, x_k)) \rightarrow c'(\text{quote}_{sort(x_1)}(x_1), \dots, \text{quote}_{sort(x_k)}(x_k)) \mid c \in \mathcal{C}_{sort(f)}^*\} \\ \cup \{\text{quote}_{sort(g)}(g(x_1, \dots, x_k)) \rightarrow g'(x_1, \dots, x_k) \mid g \in \mathcal{D}_{\mathcal{R}}^f\}$$
- Rules in U define symbols in *Unquote* exactly as in the previous transformation E_{τ} .

Given an E -strategy map φ , we define the new E -strategy map φ' ; we let $\varphi' = Emap^f(\varphi)$ as follows: $\varphi'(g) = \varphi(f)$ if $g \in \mathcal{D}$, $\varphi'(g') = \varphi(g)$ if $g \in \mathcal{D}_{\mathcal{R}}^f$, $\varphi'(c) = \varphi(c)$ if $c \in \mathcal{C}$, and $\varphi'(c') = (1 \dots ar(c'))$ if $c' \in \mathcal{C}'$, $\varphi(\text{quote}_{\tau}) = (0)$ and $\varphi(\text{unquote}_{\tau}) = (1 \ 0)$ for all sort τ ; and $\varphi(f_c) = (1 \dots ar(c) \ 0)$ for each $c \in \mathcal{C}_{sort(f)}^*$ such that $\mu^{\varphi}(c) \neq \{1, \dots, ar(c)\}$.

For the new transformation, we have similar results as for the simpler one.

Theorem 9 (Correctness). Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = V^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $\delta \in eval_{\varphi'}(\text{unquote}_{sort(t)}(\text{quote}_{sort(t)}(t)))$, then $t \rightarrow_{\mathcal{R}}^* \delta$.

Theorem 10. Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = V^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $\delta \in eval_{\varphi}(t)$, then $\delta \in eval_{\varphi'}(\text{unquote}_{sort(t)}(\text{quote}_{sort(t)}(t)))$.

Theorem 11 (Completeness). Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0, $\mu^{\varphi} \in CM_{\mathcal{R}}$, and \mathcal{R} is φ -terminating. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = V^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $t \rightarrow_{\mathcal{R}}^* \delta$, then $\delta \in eval_{\varphi'}(\text{unquote}_{sort(t)}(\text{quote}_{sort(t)}(t)))$.

Example 12. The following OBJ3 program:

```
obj EXAMPLE-TR is
  sorts Nat LNat .
  ops 0 0' : -> Nat .
  ops s s' : Nat -> Nat [strat (1 0)] .
  ops nil nil' : -> LNat .
```

```

op cons : Nat LNat -> LNat [strat (1 0)] .
ops cons' fcons : Nat LNat -> LNat [strat (1 2 0)] .
op from : Nat -> LNat [strat (1 0)] .
ops sel sel' : Nat LNat -> Nat [strat (1 2 0)] .
ops first first' : Nat LNat -> LNat [strat (1 2 0)] .
op quote : Nat -> Nat [strat (0)] .
op unquote : Nat -> Nat [strat (1 0)] .
op quote' : LNat -> LNat [strat (0)] .
op unquote' : LNat -> LNat [strat (1 0)] .
vars X Y : Nat .
var Z : LNat .
eq sel(s(X),cons(Y,Z)) = sel(X,Z) .
eq sel(0,cons(X,Z)) = X .
eq first(0,Z) = nil .
eq first(s(X),cons(Y,Z)) = cons(Y,first(X,Z)) .
eq from(X) = cons(X,from(s(X))) .
eq sel'(s(X),cons(Y,Z)) = sel'(X,Z) .
eq sel'(0,cons(X,Z)) = quote(X) .
eq first'(0,Z) = nil' .
eq first'(s(X),cons(Y,Z)) = cons'(quote(Y),first'(X,Z)) .
eq quote(0) = 0' .
eq quote'(cons(X,Z)) = cons'(quote(X),quote'(Z)) .
eq quote'(nil) = nil' .
eq quote(s(X)) = s'(quote(X)) .
eq quote(sel(X,Z)) = sel'(X,Z) .
eq quote'(first(X,Z)) = first'(X,Z) .
eq unquote(0') = 0 .
eq unquote(s'(X)) = s(unquote(X)) .
eq unquote'(nil') = nil .
eq unquote'(cons'(X,Z)) = fcons(unquote(X),unquote'(Z)) .
eq fcons(X,Z) = cons(X,Z) .
endo

```

is the (definitive) transformed version of the OBJ program of Example 1. Now, the evaluation of `unquote'(quote'(first(s(0),from(0))))` yields:

```

=====
obj EXAMPLE-TR
=====
reduce in EXAMPLE-TR : unquote'(quote'(first(s(0),from(0))))
rewrites: 10
result LNat: cons(0,nil)

```

For our running example, we can even guarantee that no infinite evaluation sequence is possible, i.e., the OBJ program of Example 12 is terminating. This can be done by using the context-sensitive recursive path ordering of [BLR02].

Example 13. Consider again the evaluation of the non-terminating expression `from(0)` using the program of Example 12. Now, we obtain:

```

=====
obj EXAMPLE-TR
=====
reduce in EXAMPLE-TR : unquote'(quote'(from(0)))
rewrites: 0
result LNat: unquote'(quote'(from(0)))

```

7 Conclusions and Related work

We summarize the contributions of this paper as follows:

- We first clarify our notion of correct and complete computations with (positive) strategy annotations. Not existing a standard, commonly accepted terminology, current definitions are rather misleading and we think this may cause an erroneous understanding of known results in the field (e.g., compare the mix of different conceits for the notion of correctness/completeness in [NF01,NO01,Pol01]).
- We demonstrate that previously known approaches for computing normal forms with (non-terminating) OBJ programs using positive strategy annotations (e.g., Nakamura and Ogata’s technique of ‘completing’ head-normalizing E -strategy maps φ for obtaining a normalizing one φ') are not completely satisfactory in practice: they do ensure correctness (that is, that computed E -normal forms are normal forms) but the desired termination do not. Hence, the proposed methodology fails to achieve the pursued semantics of normal forms (thus giving no actual opportunity for correctness).
- We ascertain the conditions (on φ) ensuring that OBJ programs using (positive) strategy annotations do compute the value of any given expression (Theorem 5). As shown in Example 5, termination of the program (under φ) is essential for achieving correct (and complete) computations.
- Theorem 5 requires that all arguments of constructor symbols are replacing. This may incur in unnecessary nontermination. Thus, we have formalized a transformation which is able to achieve (correct and) complete computations without worsening the termination behavior. Our technique differs from Nakamura and Ogata’s (or Nagaya’s) approach: we only relax the replacement restrictions associated to the (constructor) symbols after a thorough analysis of their role in the computation.

The only work addressing completeness of the E -strategy (w.r.t. normalization) is Nagaya’s thesis (although completeness is called ‘normalizability’ in Nagaya’s terminology). Nagaya establishes conditions (on the TRS and the E -strategy φ) ensuring that φ is normalizing, i.e., it is able to compute a normal form of a term whenever it exists [Nag99]. However, these results only apply to a rather restricted subclass of orthogonal TRSs. In this paper, we have focused on the functional evaluation semantics, i.e., computations leading to constructor terms or values. We are able to deal with more general programs (represented by left-linear and confluent TRSs); as a counterpart, the termination of the program must be proved either before or after transforming it for ensuring correctness and completeness (regarding functional evaluation). In *CSR*, normal forms of a term t can be obtained by successively computing its μ -normal forms s , and continuing the evaluation of t by (recursively) normalizing the maximal non-replacing subterms of s (normalization via μ -normalization [Luc02a,Luc02b]). This works for replacement maps μ which are less restrictive than the canonical replacement map $\mu_{\mathcal{R}}^{can}$. In OBJ programs, we could proceed in a similar way provided that E -normal forms are μ -normal forms. Unfortunately, we would need a ‘meta-operation’ that uses $eval_{\varphi}$ to obtain partially evaluated results (i.e., E -normal forms) and then ‘jumps’ into the non-replacing parts of them in order to obtain normal forms. Of course, this procedure is not be directly available in current OBJ implementations. The possibility of achieving a similar effect by using program transformation is a subject of future work.

References

- [BLR02] C. Borralleras, S. Lucas, and A. Rubio. Recursive Path Orderings can be Context-Sensitive. In A. Voronkov, editor *Proc. of 18th International Conference on Automated Deduction, CADE’02*, Springer LNAI to appear, 2002.
- [CELM96] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. *Electronic Notes in Theoretical Computer Science*, volume 4, 25 pages, 1996.

- [Eke98] S. Eker. Term Rewriting with Operator Evaluation Strategies. *Electronic Notes in Theoretical Computer Science*, volume 15, 20 pages, 1998.
- [FGJM85] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proc. of POPL'85*, pages 52-66, ACM Press, 1985.
- [FGK01] O. Fissore, I. Gnaedig, and H. Kirchner. Induction for termination with local strategies. *Electronic Notes in Theoretical Computer Science*, volume 58(2), 2001.
- [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 *Proc. of 1st International Conference on Formal Engineering Methods*, 1997.
- [GWMFJ00] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*, Kluwer, 2000.
- [Luc97] S. Lucas. Transformations for Efficient Evaluations in Functional Programming. In H. Glaser and P. Hartel, editors, *Proc of 9th International Symposium on Programming Languages, Implementations, Logics and Programs, PLILP'97*, LNCS 1292:127-141, Springer-Verlag, Berlin, 1997.
- [Luc98a] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1-61, January 1998.
- [Luc98b] S. Lucas. Rewriting with replacement restrictions. PhD Thesis, DSIC, Universidad Politécnic de Valencia, in spanish, October 1998.
- [Luc01a] S. Lucas. Termination of on-demand rewriting and termination of OBJ programs. In *Proc. of 3rd International Conference on Principles and Practice of Declarative Programming, PPDP'01*, pages 82-93, ACM Press, 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*, LNAI 2250:669-684, Springer-Verlag, Berlin, 2001.
- [Luc01c] S. Lucas. Transfinite Rewriting Semantics for Term Rewriting Systems. In A. Middeldorp, editor, *Proc. of 12th International Conference on Rewriting Techniques and Applications, RTA '01*, LNCS 2051:216-230. Springer-Verlag, Berlin, 2001.
- [Luc02a] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, to appear.
- [Luc02b] S. Lucas. Termination of (Canonical) Context-Sensitive Rewriting. In S. Tison, editor, *Proc. of 13th International Conference on Rewriting Techniques and Applications, RTA '02*, LNCS to appear, 2002.
- [Nag99] T. Nagaya. Reduction Strategies for Term Rewriting Systems. PhD Thesis, School of Information Science, Japan Advanced Institute of Science and Technology, March 1999.
- [NF01] M. Nakamura and K. Futatsugi. Completeness and strictness analysis for the evaluation strategy. In Y. Toyama, editor, *Proc. of 1th International Workshop on Rewriting y Proof and Computation, RPC'01*, pages 80-89, RIEC, Tohoku University, 2001.
- [NO01] M. Nakamura and K. Ogata. The evaluation strategy for head normal form with and without on-demand flags. *Electronic Notes in Theoretical Computer Science*, volume 36, 17 pages, 2001.
- [Pol01] J. van de Pol. Just-in-time: on Strategy Annotations. *Electronic Notes in Theoretical Computer Science*, volume 57, 2001.

A Appendix

Proofs of Section 5

Theorem 5 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS and $\mathcal{B} \subseteq \mathcal{C}$. Let φ be a positive E-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0 and \mathcal{R} is φ -terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\delta \in \mathcal{T}(\mathcal{B}, \mathcal{X})$. If $\mu_{\mathcal{R}}^{\mathcal{B}} \sqsubseteq \mu^{\varphi}$, then $t \rightarrow^! \delta$ iff $\delta \in \text{eval}_{\varphi}(t)$.

Proof. If $\delta \in \text{eval}_{\varphi}(t)$, then, by Theorem 3, $t \hookrightarrow_{\mu}^! \delta$. Since δ is a normal form, we have $t \rightarrow^! \delta$.

Assume that $t \rightarrow^! \delta$. By φ -termination, $\text{eval}_{\varphi}(t) \neq \emptyset$. Let $s \in \text{eval}_{\varphi}(t)$. By Theorem 3, $t \hookrightarrow_{\mu}^* s$ and s is a μ -normal form. Assume that $s \neq \delta$. By confluence $s \rightarrow^* \delta$ and by Theorem 4, $s \hookrightarrow_{\mu}^* \delta$. Hence, s is not a μ -normal form, a contradiction. Therefore, $s = \delta \in \text{eval}_{\varphi}(t)$.

Proofs of Section 6

Lemma 1. Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E-strategy map. Let $\mathcal{R}' = E_{\tau}(\mathcal{R})$ for a given sort τ and $\varphi' = E\text{map}_{\tau}(\varphi)$. Let $\delta \in \mathcal{T}(\mathcal{C})$ and δ' be δ with all constructor symbols c renamed to their alias c' . For all term s of sort τ , $\delta \in \text{eval}_{\varphi'}(\text{unquote}_{\tau}(s))$ if and only if $\delta' \in \text{eval}_{\varphi}(s)$.

Proof. We proceed by induction on δ . If δ is a constant b (which necessarily belongs to $\mathcal{T}(\mathcal{C}_{\tau}^*)$), then it is immediate. If $\delta = c(\delta_1, \dots, \delta_k)$ (and $\delta' = c'(\delta'_1, \dots, \delta'_k)$), then whenever $\delta \in \text{eval}_{\varphi'}(\text{unquote}_{\tau}(s))$ (i.e., we first consider the ‘only if’ part), we have:

$$\begin{aligned} \langle \text{unquote}_{(1 \ 0)}(\varphi'(s), A) \rangle &\rightarrow_{\varphi'} \text{unquote}_{(0)}(\varphi'(s), A, 1) \\ &\rightarrow_{\varphi'}^* \text{unquote}_{(0)}(c'_{\text{nil}}(\delta''_1, \dots, \delta''_k), 1) \\ &\rightarrow_{\varphi'}^* \text{unquote}_{(0)}(c'_{\text{nil}}(\delta''_1, \dots, \delta''_k), A) \\ &\rightarrow_{\varphi'} h^c_{(1 \ 2 \dots k)++L}(\text{unquote}_{(1 \ 0)}(\delta''_1), \dots, \text{unquote}_{(1 \ 0)}(\delta''_k)), 1) \end{aligned}$$

where h^c is c' if $\mu^{\varphi}(c) = \{1, \dots, k\}$ and h^c is f_c otherwise. Note that, in any case, we can write $\varphi'(h^c) = (1 \ 2 \dots k) ++ L$ where $L = (0)$ if $h^c = f_c$ and L is the empty list if $h^c = c$. We also have that $\text{erase}(\delta''_i) = \delta'_i$ for $\delta''_i = c'_{\text{nil}}(\delta''_1, \dots, \delta''_k)$. Even though $\varphi'(\delta')$ can be different from δ'' (the strategy labels of constructor symbols within δ'' are empty), since $\text{erase}(\varphi'(\delta')) = \delta' = \text{erase}(\delta''_i)$ is a constructor term, this is not a problem for applying the induction hypothesis to conclude:

$$\langle h^c_{(1 \ 2 \dots k)++L}(\text{unquote}_{(1 \ 0)}(\delta''_1), \dots, \text{unquote}_{(1 \ 0)}(\delta''_k)), 1) \rangle \rightarrow_{\varphi'}^* \langle h^c_L(\delta'''_1, \dots, \delta'''_k), A \rangle$$

where $\text{erase}(\delta'''_i) = \delta_i$. Now, we consider two cases depending on h^c .

1. If $h^c = c$, then L is the empty list, $h^c_L(\delta'''_1, \dots, \delta'''_k) = c_{\text{nil}}(\delta'''_1, \dots, \delta'''_k)$ and $\text{erase}(c_{\text{nil}}(\delta'''_1, \dots, \delta'''_k)) = \delta$.
2. If $h^c = f_c$, then $L = (0)$ and

$$\begin{aligned} \langle h^c_L(\delta'''_1, \dots, \delta'''_k), A \rangle &\rightarrow_{\varphi'} \langle c_{\varphi(c)}(\delta'''_1, \dots, \delta'''_k), A \rangle \\ &\rightarrow_{\varphi'}^* \langle c_{\text{nil}}(\delta'''_1, \dots, \delta'''_k), A \rangle \end{aligned}$$

and, again, $\text{erase}(c_{\text{nil}}(\delta'''_1, \dots, \delta'''_k)) = \delta$.

The ‘if’ easily follows from the previous considerations.

Theorem 6 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $\text{sort}(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_{\tau}(\mathcal{R})$ and $\varphi' = E\text{map}_{\tau}(\varphi)$. If $\delta \in \text{eval}_{\varphi'}(\text{unquote}_{\tau}(\text{quote}_{\tau}(t)))$, then $t \rightarrow_{\mathcal{R}}^* \delta$.

Proof. By Lemma 1, $\delta \in eval_{\varphi'}(\mathbf{unquote}_\tau(\mathbf{quote}_\tau(t)))$ if and only if $\delta' \in eval_{\varphi'}(\mathbf{quote}_\tau(t))$, where δ' is δ with all constructor symbols c renamed to their alias c' . Thus, we prove that $t \rightarrow_{\mathcal{R}}^* \delta$ if $\delta' \in eval_{\varphi'}(\mathbf{quote}_\tau(t))$. We proceed by induction on δ . If δ is a constant b , then the evaluation of $\mathbf{quote}_\tau(t)$ according to φ' is (note that, since $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\varphi(t) = \varphi'(t)$; we left the indexing sort of symbols \mathbf{quote} unspecified):

$$\begin{aligned} \langle \mathbf{quote}_{(1 \ 0)}(\varphi(t)), \Lambda \rangle &\rightarrow_{\varphi'} \mathbf{quote}_{(0)}(\varphi(t), 1) & (*) \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} \langle \mathbf{quote}_{(0)}(b_{nil}), \Lambda \rangle & (**) \\ &\rightarrow_{\varphi'} \langle b'_{nil}, \Lambda \rangle \end{aligned}$$

Thus, from the evaluation steps between (*) and (**) (which only involve rules in \mathcal{R} and annotations issued from φ on symbols in \mathcal{F}) we conclude that $b \in eval_\varphi(t)$. Thus, by Theorem 3, $t \hookrightarrow_{\mathcal{R}}^* b$, i.e., $t \rightarrow_{\mathcal{R}}^* b$.

If $\delta = c(\delta_1, \dots, \delta_k)$, then similarly:

$$\begin{aligned} \langle \mathbf{quote}_{(1 \ 0)}(\varphi(t)), \Lambda \rangle &\rightarrow_{\varphi'} \mathbf{quote}_{(0)}(\varphi(t), 1) & (*) \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} \langle \mathbf{quote}_{(0)}(c_{nil}(s'_1, \dots, s'_k)), \Lambda \rangle & (**) \\ &\rightarrow_{\varphi'} \langle c'_{(1 \ 2 \dots k)}(\mathbf{quote}_{(1 \ 0)}(s'_1), \dots, \mathbf{quote}_{(1 \ 0)}(s'_k)), \Lambda \rangle \end{aligned}$$

where, again, the evaluation steps between (*) and (**) only involve rules in \mathcal{R} , positions $p \geq 1$ and annotations issued from φ on symbols in \mathcal{F} (in particular, $s = c(erase(s'_1), \dots, erase(s'_k)) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$), and the presence of c is needed if the evaluation yields δ because the rule

$$\mathbf{quote}(c(x_1, \dots, x_k)) \rightarrow c'(\mathbf{quote}(x_1), \dots, \mathbf{quote}(x_k))$$

must be applied in order to progress the evaluation until δ' . Thus, we can say that $s \in eval_\varphi(t)$ and, by Theorem 3, $t \hookrightarrow_{\mathcal{R}}^* s$, i.e., $t \rightarrow_{\mathcal{R}}^* s$. The previous evaluation sequence continues as follows:

$$\begin{aligned} &\langle c'_{(1 \ 2 \dots k)}(\mathbf{quote}_{(1 \ 0)}(s'_1), \dots, \mathbf{quote}_{(1 \ 0)}(s'_k)), \Lambda \rangle \\ &\rightarrow_{\varphi'} \langle c'_{(2 \dots k)}(\mathbf{quote}_{(1 \ 0)}(s'_1), \dots, \mathbf{quote}_{(1 \ 0)}(s'_k)), 1 \rangle \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} \langle c'_{nil}(\delta'_1, \dots, \delta'_k), \Lambda \rangle \end{aligned}$$

where $erase(\delta'_i) = \delta_i$ for $1 \leq i \leq k$. Obviously, $\delta'_i \in eval_\varphi(\mathbf{quote}(erase(s'_i)))$ for $1 \leq i \leq k$. By the I.H., $erase(s'_i) \rightarrow_{\mathcal{R}}^* \delta_i$ for $1 \leq i \leq k$, hence $t \rightarrow_{\mathcal{R}}^* c(erase(s'_1), \dots, erase(s'_k)) \rightarrow_{\mathcal{R}}^* \delta$.

Theorem 7 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $sort(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_\tau(\mathcal{R})$ and $\varphi' = Emap_\tau(\varphi)$. If $\delta \in eval_\varphi(t)$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_\tau(\mathbf{quote}_\tau(t)))$.

Proof. Implicit in the proof of Theorem 6.

Theorem 8 Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a positive E -strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0, $\mu^\varphi \in CM_{\mathcal{R}}$, and \mathcal{R} is φ -terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $sort(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = E_\tau(\mathcal{R})$ and $\varphi' = Emap_\tau(\varphi)$. If $t \rightarrow_{\mathcal{R}}^* \delta$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_\tau(\mathbf{quote}_\tau(t)))$.

Proof. As in the proof of Theorem 6, we only need to prove that whenever $t \rightarrow_{\mathcal{R}}^* \delta$, then $\delta' \in eval_{\varphi'}(\mathbf{quote}_{sort(t)}(t))$, where δ' is δ with all constructor symbols c renamed to their alias c' . We proceed by induction on δ .

- If δ is a constant b , then, by taking $\mathcal{B} = \{b\}$, we have that $\mu_{\mathcal{R}}^{\mathcal{B}} = \mu_{\mathcal{R}}^{can}$ and by Theorem 5, $b \in eval_{\varphi}(t)$, i.e.,

$$\langle \varphi(t), A \rangle \rightarrow_{\varphi}^* \langle b_{nil}, A \rangle$$

In fact, we can use such \rightarrow_{φ} -sequence for building a corresponding $\rightarrow_{\varphi'}$ sequence issued from $\varphi'(\mathbf{unquote}_{\tau}(\mathbf{quote}_{\tau}(t)))$, where $\tau = sort(t)$ (note that, since $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\varphi(t) = \varphi'(t)$):

$$\begin{aligned} \langle \mathbf{quote}_{(1 \ 0)}(\varphi(t)), A \rangle &\rightarrow_{\varphi'} \mathbf{quote}_{(0)}(\varphi(t)), 1 \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} \langle \mathbf{quote}_{(0)}(b_{nil}), 1 \rangle \\ &\rightarrow_{\varphi'} \langle b'_{nil}, A \rangle \end{aligned}$$

- If δ is not a constant, then let $\delta = c(\delta_1, \dots, \delta_k)$. Again, by φ -termination, $eval_{\varphi}(t)$ is not empty and, by Theorem 3 every $s \in eval_{\varphi}(t)$ is a μ^{φ} -normal form which, by Theorem 8 in [Luc98a] it is a head-normal form (which, by confluence, is rooted by a symbol c). Then, let $s = c(s_1, \dots, s_k)$. We have:

$$\langle t, A \rangle \rightarrow_{\varphi}^! \langle s', A \rangle$$

where $erase(s') = s$. By confluence, $s_i \rightarrow^* \delta_i$ for $1 \leq i \leq k$ and by the induction hypothesis, $\delta'_i \in eval_{\varphi'}(\mathbf{quote}_{sort(s_i)}(s_i))$ for $1 \leq i \leq k$. By reasoning as in the base case, this derivation induces a corresponding $\rightarrow_{\varphi'}$ -derivation:

$$\begin{aligned} \langle \mathbf{quote}_{(1 \ 0)}(\varphi(t)), A \rangle &\rightarrow_{\varphi'} \mathbf{quote}_{(0)}(\varphi(t)), 1 \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} \langle \mathbf{quote}_{(0)}(c_{nil}(s'_1, \dots, s'_k)), 1 \rangle \\ &\rightarrow_{\varphi'} \langle \mathbf{quote}_{(0)}(c_{nil}(s'_1, \dots, s'_k)), A \rangle \\ &\rightarrow_{\varphi'} \langle c'_{(1 \ 2 \dots k)}(\mathbf{quote}_{(1 \ 0)}(s'_1), \dots, \mathbf{quote}_{(1 \ 0)}(s'_k)), 1 \rangle \\ &\rightarrow_{\varphi'}^* \langle c'_{nil}(\delta''_1, \dots, \delta''_k), A \rangle \end{aligned}$$

where $erase(c'_{nil}(\delta''_1, \dots, \delta''_k)) = c'(erase(\delta''_1), \dots, erase(\delta''_k)) = \delta'$.

Lemma 2. *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E-strategy map. Let $\mathcal{R}' = \mathbb{V}^f(\mathcal{R})$ for a given $f \in \mathcal{D}$ and $\varphi' = Emap^f(\varphi)$. Let $\delta \in \mathcal{T}(\mathcal{C})$ and δ' be δ with all constructor symbols c renamed to their alias c' . For all term s of sort τ , $\delta \in eval_{\varphi'}(\mathbf{unquote}_{\tau}(s))$ if and only if $\delta' \in eval_{\varphi'}(s)$.*

Proof. Since the strategy annotation for $\mathbf{unquote}_{\tau}$ is the same both in $Emap^f(\varphi)$ and in $\varphi' = Emap_{sort(f)}(\varphi)$, the proof is analogous to that of Lemma 1.

Theorem 9 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathbb{V}^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $\delta \in eval_{\varphi'}(\mathbf{unquote}_{sort(t)}(\mathbf{quote}_{sort(t)}(t)))$, then $t \rightarrow_{\mathcal{R}'}^* \delta$.*

Proof. (Sketch) As in the proof of Theorem 6 (now using Lemma 2), we only need to prove that $t \rightarrow_{\mathcal{R}'}^* \delta$ if $\delta' \in eval_{\varphi'}(\mathbf{quote}_{\tau}(t))$. We proceed by induction on δ . If δ is a constant b , then either $t = b$ or $root(t) = g \in \mathcal{F}_{\mathcal{R}}^f$ and the evaluation of $\mathbf{quote}_{\tau}(t)$ according to φ' is (note that, since $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\varphi(t) = \varphi'(t)$; we left the indexing sort of symbols \mathbf{quote} unspecified):

$$\begin{aligned} \langle \mathbf{quote}_{(0)}(g_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k))), A \rangle &\rightarrow_{\varphi'} \langle g'_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k)), A \rangle & (*) \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} h'_{\varphi(h)}(s_1, \dots, s_m), A \\ &\rightarrow_{\varphi'} \langle \delta''_{nil}, A \rangle & (**) \end{aligned}$$

where $h' \in \mathcal{F}'$ is the renamed version of a defined symbol $h \in \mathcal{F}$. Note that the evaluation steps between $(*)$ and $(**)$ only involve rules in \mathcal{R}' that either are in R (if the rewriting steps are performed below position 1) or in S (if the rewriting steps are performed at position 1). Thus, since the rules in S are modified versions of rules in \mathcal{R} , we can write $t \rightarrow_{\mathcal{R}}^* b$.

If $\delta = c(\delta_1, \dots, \delta_k)$, then either $t = c(t_1, \dots, t_k)$ or $root(t) = g \in \mathcal{F}_{\mathcal{R}}^f$. In the first case, we have:

$$\begin{aligned} \langle \mathbf{quote}_{(0)}(c_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k))), A \rangle &\rightarrow_{\varphi'} \langle c'_{(1 \ 2 \dots k)}(\mathbf{quote}_{(0)}(\varphi(t_1)), \dots, \mathbf{quote}_{(0)}(\varphi(t_k))), A \rangle \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} c'_{nil}(\delta''_1, \dots, \delta''_k), A \end{aligned}$$

where $erase(\delta''_i) = \delta'_i$ for $1 \leq i \leq k$, and the conclusion easily follows by the induction hypothesis. In the second case, let $t = g(t_1, \dots, t_k)$. We have,

$$\begin{aligned} \langle \mathbf{quote}_{(0)}(g_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k))), A \rangle &\rightarrow_{\varphi'} \langle g'_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k)), A \rangle & (*) \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} h'_{\varphi(h)}(s_1, \dots, s_m), A \\ &\rightarrow_{\varphi'} \langle s, A \rangle & (**) \end{aligned}$$

where, $h' \in \mathcal{F}'$ and $s = C[s_1, \dots, s_n]$ is such that $erase(C[\])$ is the maximal constructor prefix of s which is shared with δ' (i.e., there are $\delta''_1, \dots, \delta''_n$ such that $\delta' = erase(C[\delta''_1, \dots, \delta''_n])$). Thus, again, we can say that $t \rightarrow_{\mathcal{R}}^* s$. Note also that for all $1 \leq i \leq k$, either $s'_i = \mathbf{quote}_{(0)}(s''_i)$, or $erase(s_i) = h'(u_1, \dots, u_m)$ for $h' \in \mathcal{F}'$ and $u_j \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for $1 \leq j \leq m$ (i.e., $h(u_1, \dots, u_m)$ belongs to the ‘base’ of the inductive set $ev^f(\mathcal{F}, \mathcal{X})$). Moreover, every constructor symbol c' in $C[\]$ is labelled with $\varphi(c') = \{1, \dots, ar(c')\}$. Thus, by using the induction hypothesis (for \mathbf{quote} -rooted terms s_i is obvious; for the other terms, it follows from the fact that terms $h(u_1, \dots, u_m)$ in the ‘base’ of $ev^f(\mathcal{F}, \mathcal{X})$ are transformed into $h'(u_1, \dots, u_m)$ by a single rewriting step issued on $\mathbf{quote}(h(u_1, \dots, u_m))$), we have that $erase(s_i) \rightarrow_{\mathcal{R}}^* \delta_i$ for $1 \leq i \leq n$ and the conclusion follows.

Theorem 10 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a positive E-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathbf{V}^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $\delta \in eval_{\varphi}(t)$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_{s_{ort(t)}}(\mathbf{quote}_{s_{ort(t)}}(t)))$.*

Proof. Similar to the proof of Theorem 9.

Theorem 11 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a positive E-strategy map such that for all $f \in \mathcal{D}$, $\varphi(f)$ ends in 0, $\mu^{\varphi} \in CM_{\mathcal{R}}$, and \mathcal{R} is φ -terminating. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathbf{V}^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $t \rightarrow_{\mathcal{R}}^* \delta$, then $\delta \in eval_{\varphi'}(\mathbf{unquote}_{s_{ort(t)}}(\mathbf{quote}_{s_{ort(t)}}(t)))$.*

Proof. (Sketch) We proceed by induction on the structure of terms in $ev^f(\mathcal{F}, \mathcal{X})$ (again, we use Lemma 2). For the base case, we consider terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $root(t) \in \mathcal{D}_{\mathcal{R}}^f$ (if $t \in \mathcal{X}$, it is immediate). Let $t = g(\bar{t})$ for $g \in \mathcal{F}_{\mathcal{R}}^f$. Now we proceed by induction on δ .

- If δ is a constant or a variable b , then, by taking $\mathcal{B} = \{b\}$, we have that $\mu_{\mathcal{R}}^{\mathcal{B}} = \mu_{\mathcal{R}}^{can}$ and by Theorem 5, $b \in eval_{\varphi}(t)$, i.e.,

$$\begin{aligned} \langle g_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k)), A \rangle &\rightarrow_{\varphi} \\ &\vdots \\ &\rightarrow_{\varphi} \langle h_{\varphi(h)}(s_1, \dots, s_k), A \rangle \\ &\rightarrow_{\varphi} \langle b_{nil}, A \rangle \end{aligned}$$

In fact, we can use such \rightarrow_φ -sequence for building a corresponding $\rightarrow_{\varphi'}$ sequence issued from $\varphi'(\mathbf{quote}_{\mathit{sort}(t)}(t))$:

$$\begin{aligned} \langle \mathbf{quote}_{(0)}(g_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k))), A \rangle &\rightarrow_{\varphi'} \langle g'_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k)), A \rangle \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} h'_{\varphi(h)}(s_1, \dots, s_k), A \rangle \\ &\rightarrow_{\varphi'} \langle b'_{\mathit{nil}}, A \rangle \end{aligned}$$

Thus, $b' \in \mathit{eval}_{\varphi'}(\mathbf{quote}_{\mathit{sort}(t)}(t))$.

- If δ is not a constant, then let $c = \mathit{root}(\delta)$. Again, by φ -termination, $\mathit{eval}_\varphi(t)$ is not empty and, by Theorem 3 every $s \in \mathit{eval}_\varphi(t)$ is a μ^φ -normal form which, by Theorem 8 in [Luc98a] it is a head-normal form (which, by confluence, is rooted by a symbol c). Then, we have an initial segment of the φ -evaluation of t which is as follows:

$$\langle t, A \rangle \rightarrow_\varphi^* \dots \rightarrow_\varphi \langle h(s_1, \dots, s_k), A \rangle \rightarrow_\varphi \langle C[u_1, \dots, u_n], A \rangle$$

where $h \in \mathcal{D}$, $C[\dots] \in \mathcal{T}(\mathcal{C}_{\mathit{sort}(f)}^* \cup \{\square\}, \mathcal{X})$, and $\mathit{root}(u_i) \in \mathcal{D} \cup \mathcal{X}$ for $1 \leq i \leq n$. By confluence, we can write $\delta = C[\delta_1, \dots, \delta_n]$ and $u_i \rightarrow^* \delta_i$ for $1 \leq i \leq n$. By reasoning as in the base case, this derivation induces a corresponding $\rightarrow_{\varphi'}$ -derivation:

$$\begin{aligned} \langle \mathbf{quote}_{(0)}(g_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k))), A \rangle &\rightarrow_{\varphi'} \langle g'_{\varphi(g)}(\varphi(t_1), \dots, \varphi(t_k)), A \rangle \\ &\rightarrow_{\varphi'} \\ &\vdots \\ &\rightarrow_{\varphi'} h'_{\varphi(h)}(s_1, \dots, s_k), A \rangle \\ &\rightarrow_{\varphi'} \langle C'[u'_1, \dots, u'_n], A \rangle \end{aligned}$$

where $\mathit{root}(C'[u'_1, \dots, u'_n]) = c'_{(1 \ 2 \dots \mathit{ar}(c'))}$, and, for all $1 \leq i \leq n$, either $u'_i = \mathbf{quote}_{(0)}(u_i)$ or $\mathit{root}(\mathit{erase}(u'_i)) = f' \in \mathcal{D}'$ (and $f = \mathit{root}(\mathit{erase}(u_i))$). Thus, we can apply the (second) induction hypothesis. Therefore, $\kappa^f(\delta_i) \in \mathit{eval}_{\varphi'}(\mathbf{quote}_{\mathit{sort}(u_i)}(\mathit{erase}(u_i)))$ for $1 \leq i \leq n$. Therefore, we easily conclude that $\delta \in \mathit{eval}_{\varphi'}(\mathbf{quote}_{\mathit{sort}(t)}(t))$ (note that $\langle \mathbf{quote}_{(0)}(\varphi'(u_i)), A \rangle \rightarrow_{\varphi'} \langle u'_i, A \rangle$).

If $t = c(t_1, \dots, t_k)$ for some constructor symbol c , then, by confluence, we can write $\delta = c(\delta_1, \dots, \delta_k)$ and $t_i \rightarrow^* \delta_i$ for $1 \leq i \leq k$. By induction hypothesis, $\kappa^f(\delta_i) \in \mathit{eval}_{\varphi'}(\mathbf{quote}_{\mathit{sort}(t_i)}(t_i))$. Thus, since

$$\langle \mathbf{quote}_{(0)}(c_{\varphi'(c)}(\varphi'(t_1), \dots, \varphi'(t_k))), A \rangle \rightarrow_{\varphi'} \langle c'_{(1 \ 2 \dots k)}(\mathbf{quote}_{(0)}(\varphi(t_1)), \dots, \mathbf{quote}_{(0)}(\varphi(t_k))), A \rangle$$

and $\kappa^f(c(\delta_1, \dots, \delta_k)) = c'(\kappa^f(\delta_1), \dots, \kappa^f(\delta_k))$, the conclusion follows.