Electronic Notes in Theoretical Computer Science 71 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume71.html 20 pages

Correct and Complete (Positive) Strategy Annotations for OBJ⋆

María Alpuente [1], Santiago Escobar [2], and Salvador Lucas [3]

DSIC, Universidad Politécnica de Valencia, Spain.

Abstract

Strategy annotations are used in several rewriting-based programming languages to introduce replacement restrictions aimed at improving efficiency and/or reducing the risk of nontermination. Unfortunately, rewriting restrictions can have a negative impact on 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.

Key words: Declarative programming, OBJ, strategy annotations.

1 Introduction

Strategy annotations are used in the OBJ family of languages [4] (OBJ2 [6], OBJ3 [8], CafeOBJ [7], and Maude [3]) to *avoid nontermination* ([8], Section 2.4.4).

Example 1.1 The following OBJ program:

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

⋆ 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] Email: alpuente@dsic.upv.es
[2] Email: sescobar@dsic.upv.es
[3] Email: slucas@dsic.upv.es

[4] As in [8], by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

```
  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 4.4 below for a formal justification of this claim).

Termination of rewriting under strategy annotations has been studied in a number of papers [5,13,14]. Unfortunately, using rewriting restrictions may cause *incompleteness*, i.e., normal forms of input expressions could be unreachable by restricted computation. For instance, using the program in Example 1.1 we are *not* able to compute the list of integers that corresponds to the evaluation of `first(s(0),from(0))`. As we show below, the evaluation of this expression stops yielding the term `cons(0,first(0,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, 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 $a\ R\ b$; NF_R is the set of R-normal forms [1]. 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 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 context $C[\,]$ is a term from $\mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{X})$, where $\Box$ 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 $p, q, \ldots$ are represented by

chains of positive natural numbers used to address subterms of t. By Λ, we denote the empty chain (referring to the root of the term). 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 $\mathcal{P}os(t)$, we denote the set of positions of a term t. Positions of non-variable symbols in t are denoted as $\mathcal{P}os_{\mathcal{F}}(t)$ and $\mathcal{P}os_{\mathcal{X}}(t)$ 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 $root(t)$ and $root(t, p)$ is $root(t|_p)$. A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ which homomorphically extends to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$.

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 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 $\mathcal{D} = \{f \mid f(l_1, \ldots, l_k) \to r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. Then, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ is the set of constructor terms. Let $\mathcal{P}os_{\mathcal{D}}(t)$ (resp. $\mathcal{P}os_{\mathcal{C}}(t)$) be the set of positions of defined (resp. constructor) symbols of term t. An instance $\sigma(l)$ of a *lhs* $l \in L(\mathcal{R})$ is a redex. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to s (at position p), written $t \xrightarrow{p}_{\mathcal{R}} s$ (or just $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some $l \to r \in R$, $p \in \mathcal{P}os(t)$ and substitution σ. A term is a normal form if it is a $\to$-normal form. Let $\mathsf{NF}_{\mathcal{R}}$ 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. Let $\mathsf{HNF}_{\mathcal{R}}$ be the set of head-normal forms of $\mathcal{R}$. A TRS is terminating if $\to$ is terminating.

3 Rewriting with syntactic replacement restrictions

A mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a *replacement map* (or $\mathcal{F}$-map) if $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ for all $f \in \mathcal{F}$ [9]. 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* $\mathcal{P}os^{\mu}(t)$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is: $\mathcal{P}os^{\mu}(t) = \{\Lambda\}$ if $t \in \mathcal{X}$, and $\mathcal{P}os^{\mu}(t) = \{\Lambda\} \cup \bigcup_{i \in \mu(root(t))} i.\mathcal{P}os^{\mu}(t|_i)$ if $t \notin \mathcal{X}$. In *context-sensitive rewriting* (*CSR* [9]), we (only) rewrite *replacing* redexes: t μ-rewrites to s (written $t \hookrightarrow_{\mu} s$) if $t \xrightarrow{p}_{\mathcal{R}} s$ and $p \in \mathcal{P}os^{\mu}(t)$. The $\hookrightarrow_{\mu}$-normal forms are called μ-normal forms. $\mathsf{NF}^{\mu}_{\mathcal{R}}$ is the set of μ-normal forms of $\mathcal{R}$. The μ-normal forms include all normal forms of $\mathcal{R}$ (but not viceversa). A TRS $\mathcal{R}$ is μ-terminating if $\hookrightarrow_{\mu}$ is terminating. The canonical replacement map $\mu^{can}_{\mathcal{R}}$ is *the most restrictive replacement map which ensures that the non-variable subterms of the left-hand sides of the rules of $\mathcal{R}$ are replacing.* Note that $\mu^{can}_{\mathcal{R}}$

is easily obtained from $\mathcal{R}$: for all $f \in \mathcal{F}$ and $i \in \{1, \ldots, ar(f)\}$,

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

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

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 $\{0, 1, \ldots, k\}$ which are given in parentheses (see Example 1.1). A mapping φ that associates a local strategy $\varphi(f)$ to every $f \in \mathcal{F}$ is called an E-strategy map [19]. Algebraic languages such as OBJ2, OBJ3, CafeOBJ and Maude admit the specification of E-strategies. Symbols without an explicit local strategy are given a *default* strategy whose concrete shape depends on the language considered [5]. Given an OBJ program P, we (separately) consider the corresponding TRS $\mathcal{R}$ which consists of the set of rewriting rules in P and the E-strategy map φ that corresponds to its strategy annotations. Semantics of OBJ programs under a given E-strategy map φ is given by means of a mapping $eval_\varphi : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{P}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ (from terms to their sets of 'computed values'). Following [17,19] we describe $eval_\varphi$ by using a reduction relation $\to_\varphi$ on pairs of labelled terms and positions.

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 exceding $n \in \mathbb{N}$. 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 (labelling) mapping from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$ as follows:

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

The mapping $erase : \mathcal{T}(\mathcal{F}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ removes labellings from symbols in the obvious way. Given a TRS $\mathcal{R} = (\mathcal{F}, R)$ and an E-strategy map φ for $\mathcal{F}$, the binary relation $\to_\varphi$ 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 [17,19]: $\langle t, p\rangle \to_\varphi \langle s, q\rangle$ if and only if $p \in \mathcal{P}os(t)$ and either

(i) $root(t,p) = f_{nil}$, $s = t$ and $p = q.i$ for some i; or

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

(iii) $t|_p = f_{0:L}(t_1, \ldots, t_k)$, $erase(t|_p)$ is not a redex, $s = t[f_L(t_1, \ldots, t_k)]_p$, $q = p$; or

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

We let $eval_\varphi(t) = \{erase(s) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \langle \varphi(t), \Lambda\rangle \to_\varphi^! \langle s, \Lambda\rangle\}$ [17,19]. A TRS

[5] For instance, in Maude, the default local strategy associated to a k-ary symbol f, is $(1\ 2\ \cdots k\ 0)$, see [4].

$\mathcal{R}$ is φ-terminating if, for all $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, there is no infinite $\rightarrow_\varphi$-rewrite sequence starting from $\langle \varphi(t), \Lambda \rangle$. An OBJ program P is **terminating** if the corresponding TRS $\mathcal{R}$ is φ-terminating [13].

4.1 E-strategies and context-sensitive rewriting

Rewriting with strategy annotations is closely related to *CSR*. Given an *E*-strategy map φ for $\mathcal{F}$, we define $\mu^\varphi \in M_\mathcal{F}$ as follows: $\mu^\varphi(f) = \{i \in \varphi(f) \mid i \neq 0\}$ for all $f \in \mathcal{F}$, where $e \in L$ means that item e appears somewhere within the list L. We will drop superscript φ from μ^φ if no confusion arises. Moreover, we also write $\varphi \in CM_\mathcal{R}$ meaning that $\mu^\varphi \in CM_\mathcal{R}$.

Example 4.1 The TRS $\mathcal{R}$:

```
sel(0,cons(x,z))       → x
sel(s(x),cons(y,z))    → sel(x,z)
first(0,z)             → nil
first(s(x),cons(y,z))  → cons(y,first(x,z))
from(x)                → cons(x,from(s(x)))
```

together with the replacement map

$$\mu(\mathtt{s}) = \mu(\mathtt{cons}) = \mu(\mathtt{from}) = \{1\} \quad \text{and} \quad \mu(\mathtt{sel}) = \mu(\mathtt{first}) = \{1,2\}$$

correspond to the OBJ program in Example 1.1.

Every $\rightarrow_\varphi$-reduction step issued on $\langle t, p\rangle$ correspond to a μ^φ-rewriting step on the unlabelled version *erase*(t) of t (or *erase*(t) just remains unchanged).

Theorem 4.2 [13] *Let $\mathcal{R}$ be a TRS and φ be an E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}_\mathcal{L}, \mathcal{X}_\mathcal{L})$, and $p \in \mathcal{P}os^\mu(erase(t))$ be s.t. $root(t,p) = f_L$ for some suffix L of $\varphi(f)$. If $\langle t,p\rangle \rightarrow_\varphi \langle s,q\rangle$, then $q \in \mathcal{P}os^\mu(erase(s))$ and $erase(t) \hookrightarrow_\mu^= erase(s)$.*

Termination of OBJ programs and termination of *CSR* are also related.

Theorem 4.3 [13] *An* OBJ *program* P *with 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 [12] for an overview of the different methods for proving termination of *CSR*.

Example 4.4 Consider $\mathcal{R}$ and μ as in Example 4.1. The μ-termination of (a superset of) $\mathcal{R}$ is demonstrated in Example 7 of [2]. Hence, by Theorem 4.3, the OBJ program in Example 1.1 is terminating.

5 Correctness and completeness

A *rewriting semantics* for a TRS $\mathcal{R} = (\mathcal{F}, R)$ is a mapping $\mathsf{S} : \mathcal{T}(\mathcal{F},\mathcal{X}) \rightarrow \mathcal{P}(\mathcal{T}(\mathcal{F},\mathcal{X}))$ such that, for all $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and $s \in \mathsf{S}(t)$, $t \rightarrow_\mathcal{R}^* s$ [15]. Note that, given a TRS $\mathcal{R}$ and an *E*-strategy map φ, $eval_\varphi$ is a rewriting semantics for $\mathcal{R}$. A semantics S is *deterministic* (resp. *defined*) if $\forall t \in \mathcal{T}(\mathcal{F},\mathcal{X}), |\mathsf{S}(t)| \leq 1$

(resp. $|\mathsf{S}(t)| \geq 1$). In general, $eval_\varphi$ is not deterministic or defined. Note that φ-termination of $\mathcal{R}$ implies definedness of $eval_\varphi$.

The semantics which is most commonly considered in functional programming is the set of constructor terms that $\mathcal{R}$ is able to produce in a finite number of rewriting steps ($\mathsf{eval}(t) = \{s \in \mathcal{T}(\mathcal{C},\mathcal{X}) \mid t \to^*_\mathcal{R} s\}$). Other kinds of semantics often considered for $\mathcal{R}$ are, e.g., the set of all possible reducts of a term which are head-normal forms ($\mathsf{hnf}(t) = \{s \in \mathsf{HNF}_\mathcal{R} \mid t \to^*_\mathcal{R} s\}$), or normal forms ($\mathsf{nf}(t) = \mathsf{hnf}(t) \cap \mathsf{NF}_\mathcal{R}$). Thus, given a semantics S for $\mathcal{R}$ (e.g., $\mathsf{S} \in \{\mathsf{eval}, \mathsf{hnf}, \mathsf{nf}\}$), a different rewriting semantics for $\mathcal{R}$ (e.g., $eval_\varphi$) is:

correct (w.r.t. S) if $eval_\varphi(t) \subseteq \mathsf{S}(t)$ for all $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, and

complete (w.r.t. S) if, $\mathsf{S}(t) \subseteq eval_\varphi(t)$ for all $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$.

Computations with OBJ programs produce expresions (by means of $eval_\varphi$) called *E-normal forms* (*ENF*s). Such terms are not generally normal forms (i.e., terms without redexes). Therefore, $eval_\varphi$ is not guaranteed to be either correct or complete w.r.t. nf. In fact, we have the following:

Theorem 5.1 [13] *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and φ be a 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.*

Requiring that $\varphi(f)$ ends in 0 for all $f \in \mathcal{D}$ is essential in our development (see also [4] for a thorough analysis of the relevance of this requirement). Thus, we say that a E-strategy map φ is regular [6] if this condition holds.

If the strategy annotations are 'compatible' with the canonical replacement map, we can ensure that the E-strategy is correct w.r.t. hnf.

Theorem 5.2 [13] *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear TRS and φ be a regular E-strategy map such that $\varphi \in CM_\mathcal{R}$. If $s \in eval_\varphi(t)$, then s is a head-normal form.*

If we restrict the attention to the computation of values (i.e., constructor terms), then *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 $\mu^\mathcal{B}_\mathcal{R}$ to be $\mu^\mathcal{B}_\mathcal{R}(c) = \{1, \ldots, ar(c)\}$ for all $c \in \mathcal{B}$ and $\mu^\mathcal{B}_\mathcal{R}(f) = \mu^{can}_\mathcal{R}(f)$ if $f \in \mathcal{F} - \mathcal{B}$. Note that $\mu^\mathcal{B}_\mathcal{R} \in CM_\mathcal{R}$.

Theorem 5.3 [9] *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 $\mu^\mathcal{B}_\mathcal{R} \sqsubseteq \mu$. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{B},\mathcal{X})$. Then, $t \to^* \delta$ iff $t \hookrightarrow^*_\mu \delta$.*

Theorem 5.3 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 5.3 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.

[6] This terminology is used in [20], with a slightly different meaning.

Theorem 5.4 *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 regular E-strategy map such that $\mathcal{R}$ is φ-terminating. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\delta \in \mathcal{T}(\mathcal{B}, \mathcal{X})$. If $\mu^{\mathcal{B}}_{\mathcal{R}} \sqsubseteq \mu^{\varphi}$, then $t \to^{!} \delta$ 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 in Example 1.1 (since $\mu^{\{0,s\}}_{\mathcal{R}} \sqsubseteq \mu^{\varphi}$). This is not true for expressions of the sort `LNat` as the following example shows.

Example 5.5 The evaluation of expression t = `first(s(0),from(0))` of sort `LNat` using the program in Example 1.1 yields (we use the version 1.0.5 of the Maude interpreter [7] but other interpreters behave likewise [8]):

```
Maude> reduce first(s(0),from(0)) .
reduce in EXAMPLE : first(s(0), from(0)) .
rewrites: 2 in -10ms cpu (0ms real) (~ rewrites/second)
result LNat: cons(0, first(0, from(s(0))))
```

Note that `cons(0,first(0,from(s(0))))` is not a normal form. However, $t \to^{*}$ `cons(0,nil)` $\in \mathcal{T}(\mathcal{C}, \mathcal{X})$, i.e., `cons(0,nil)` is a value of t which cannot be obtained by using the Maude interpreter.

Correctness of OBJ computations w.r.t. nf can also be achieved:

(i) Nagaya shows that if $\varphi(f)$ contains all indices $0, 1, \ldots, ar(f)$ for each symbol $f \in \mathcal{F}$, and $\varphi(f)$ ends in 0 for defined symbols $f \in \mathcal{D}$, then $eval_{\varphi}$ is correct w.r.t. nf (Theorem 6.1.12 in [17]).

(ii) Nakamura and Ogata show that given a strategy map φ, if $eval_{\varphi}$ is correct w.r.t. hnf, then $eval_{\varphi'}$ is correct w.r.t. nf for any φ' given by $\varphi'(f) = \varphi(f)$++$(i_1 \ \cdots \ i_n)$ for all symbol $f \in \mathcal{F}$ (where '++' appends two lists, and $\{i_1, \ldots, i_n\} = \{1, \ldots, ar(f)\} - \mu^{\varphi}(f)$) (Theorem 3.2 in [19]).

For instance, φ as given in Example 1.1 is correct w.r.t. hnf (use Theorem 5.2). Moreover, since the OBJ program in Example 1.1 is φ-terminating, $eval_{\varphi}$ is defined. Thus, the evaluation of every term t yields a head-normal form of t (i.e., φ can be thought of as being *head-normalizing*). Unfortunately $eval_{\varphi'}$ is not defined anymore: the head-normalizing behavior of φ gets lost.

Example 5.6 Consider the program in Example 1.1 with φ'(`cons`) = (1 2) and $\varphi'(f) = \varphi(f)$ for every other symbol f (rename the program to `EXAMPLE-INF`). Consider again the evaluation of t = `first(s(0),from(0))`:

```
Maude> reduce first(s(0),from(0)) .
reduce in EXAMPLE-INF : first(s(0), from(0)) .
Segment violation
```

[7] Available at `http://maude.csl.sri.com/system/`.

[8] We have reproduced all our experiments using the OBJ3 interpreter v. 2.0 (available at `http://www.kindsoftware.com/products/opensource/obj3/OBJ3/`) and the CafeOBJ interpreter v. 1.3.1 (available at `http://www.ipa.go.jp/STC/CafeP/cafe.html`).

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

$$\varphi'(t) = \mathtt{first}_{(1\ \ 2\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil}))$$

using $\to_{\varphi'}$ does not terminate (we underline the contracted redexes):

$$\begin{aligned}
&\langle \mathtt{first}_{(1\ \ 2\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), \Lambda\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(2\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), 1\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(2\ \ 0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), 1.1\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(2\ \ 0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), 1\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(2\ \ 0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), \Lambda\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{from}_{(1\ \ 0)}(\mathtt{0}_{nil})), 2\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{from}_{(0)}(\mathtt{0}_{nil})), 2.1\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\underline{\mathtt{from}_{(0)}(\mathtt{0}_{nil})}), 2\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{cons}_{(1\ \ 2)}(\mathtt{0}_{nil},\mathtt{from}_{(1\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))), 2\rangle\\
&\quad\to^{+}_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{cons}_{(2)}(\mathtt{0}_{nil},\mathtt{from}_{(1\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))), 2\rangle\\
&\quad\to_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{cons}_{nil}(\mathtt{0}_{nil},\mathtt{from}_{(1\ \ 0)}(\mathtt{s}_{(1)}(\mathtt{0}_{nil})))), 2.2\rangle\\
&\quad\to^{+}_{\varphi'} \langle \mathtt{first}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}),\mathtt{cons}_{nil}(\mathtt{0}_{nil},\underline{\mathtt{from}_{(0)}(\mathtt{s}_{nil}(\mathtt{0}_{nil}))})), 2.2\rangle\\
&\quad\to_{\varphi'} \cdots
\end{aligned}$$

The Maude interpreter 'shows' this infinite sequence as a 'segment violation'.

Thus, the φ-termination of $\mathcal{R}$ (see Example 4.4) does *not* ensure definedness of $eval_{\varphi'}$ 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 5.5 and 5.6). Example 5.6 also shows that requiring φ-termination in Theorem 5.4 is essential for ensuring correct and complete evaluations (note that $\mathcal{R}$ and φ' in Example 5.6 fulfill all requirements in Theorem 5.4, 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 suggest to isolate the replacement restrictions which are needed to achieve the head-evaluation of a term t (which, at least, requires $\mu^{can}_{\mathcal{R}}$, see Theorem 5.2) from the restrictions which are needed to get them within a constructor context $C[\ ] \in \mathcal{T}(\mathcal{B} \cup \{\Box\}, \mathcal{X})$ for some $\mathcal{B} \subseteq \mathcal{C}$ (which, at least, requires $\mu^{\mathcal{B}}_{\mathcal{R}}$, see Theorem 5.4). In practice, we only need (and want) to fix the sort τ of input expressions we want to evaluate in order to fix the 'interesting' constructor terms. Assume that symbols $f \in \mathcal{F}$ are sorted by: $f : \tau_1 \times \cdots \times \tau_k \to \tau'$. The (output) *sort* of f is $sort(f) = \tau'$. 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, $\mathcal{C}^*_{\mathtt{Nat}} = \{\mathtt{0}, \mathtt{s}\}$ and $\mathcal{C}^*_{\mathtt{LNat}} = \{\mathtt{0}, \mathtt{s}, \mathtt{nil}, \mathtt{cons}\}$. We introduce a set

$\mathcal{C}'$ of fresh constructor symbols: they are renamed versions c' of the original constructors $c \in \mathcal{C}^*_\tau$ that evaluate all the arguments.

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

$$\mathtt{quote}_{\tau'}(c(x_1,\ldots,x_k)) \to c'(\mathtt{quote}_{\tau_1}(x_1),\ldots,\mathtt{quote}_{\tau_k}(x_k))$$

where $c, c' : \tau_1 \times \cdots \times \tau_k \to \tau'$. Let *Quote* be the set containing all these symbols: $Quote = \{\mathtt{quote}_{\tau'} \mid \exists c \in \mathcal{C}^*_\tau, sort(c) = \tau'\}$. The evaluation of a term t would proceed by reducing $\mathtt{quote}_{sort(t)}(t)$. The obtained value is built by using symbols in $\mathcal{C}'$ only. After the evaluation, new symbols $\mathtt{unquote}_{\tau'} : \tau' \to \tau'$ are used to reverse the renamings. For each constant $b \in \mathcal{C}^*_\tau$, we add a rule

$$\mathtt{unquote}_{sort(b)}(b') \to b$$

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

$$\mathtt{unquote}_{\tau'}(c'(x_1,\ldots,x_k)) \to c(\mathtt{unquote}_{\tau_1}(x_1),\ldots,\mathtt{unquote}_{\tau_k}(x_k))$$

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

$$\mathtt{unquote}_{\tau'}(c'(x_1,\ldots,x_k)) \to f_c(\mathtt{unquote}_{\tau_1}(x_1),\ldots,\mathtt{unquote}_{\tau_k}(x_k))$$

$$f_c(x_1,\ldots,x_k) \to c(x_1,\ldots,x_k)$$

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 $\mathsf{E}_\tau(\mathcal{R})$. The transformed TRS $\mathsf{E}_\tau(\mathcal{R})$ includes the rules of the original TRS $\mathcal{R}$. The added rules manage the appropriate quoting and unquoting of constructor symbols: quoted constructors enable the evaluation of all their arguments; after evaluating them, symbol `unquote` restores the original constructor c. Therefore, we also extend the (original) E-strategy φ: let $\varphi' = Emap_\tau(\varphi)$ as follows: $\varphi'(f) = \varphi(f)$ if $f \in \mathcal{F}$, $\varphi'(c') = (1\ \cdots\ ar(c'))$ if $c' \in \mathcal{C}'$, $\varphi'(\mathtt{quote}_{\tau'}) = \varphi'(\mathtt{unquote}_{\tau'}) = (1\ 0)$ for all sort τ', and $\varphi(f_c) = (1\ \cdots\ ar(c)\ 0)$ for each $c \in \mathcal{C}^*_\tau$ such that $\mu^\varphi(c) \neq \{1,\ldots,ar(c)\}$. In the following results, $eval_{\varphi'}$ uses φ' and $\mathsf{E}_\tau(\mathcal{R})$ to evaluate terms ($eval_\varphi$ uses φ and $\mathcal{R}$, as above). Our transformation is correct [9] in a very general setting.

Theorem 6.1 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a regular E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $sort(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathsf{E}_\tau(\mathcal{R})$ and $\varphi' = Emap_\tau(\varphi)$. If $\delta \in eval_{\varphi'}(\mathtt{unquote}_\tau(\mathtt{quote}_\tau(t)))$, then $t \to^*_{\mathcal{R}} \delta$.*

Thus, no 'unexpected' value can be obtained when evaluating $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ of sort τ as $\mathtt{unquote}_\tau(\mathtt{quote}_\tau(t))$. Moreover, no constructor term (of sort τ) obtained by using φ and $\mathcal{R}$ gets lost when $Emap_\tau(\varphi)$ and $\mathsf{E}_\tau(\mathcal{R})$ are used instead.

[9] In this section we do not use 'corrrect' and 'complete' in the technical sense defined in Section 5 because we need to consider two rewrite systems rather than only one.

Theorem 6.2 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a regular E-strategy map. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be such that $sort(t) = \tau$ and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathsf{E}_\tau(\mathcal{R})$ and $\varphi' = Emap_\tau(\varphi)$. If $\delta \in eval_\varphi(t)$, then $\delta \in eval_{\varphi'}(\mathtt{unquote}_\tau(\mathtt{quote}_\tau(t)))$.*

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

Theorem 6.3 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a regular E-strategy map such that $\varphi \in CM_\mathcal{R}$, 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 $\mathcal{R}' = \mathsf{E}_\tau(\mathcal{R})$ and $\varphi' = Emap_\tau(\varphi)$. If $t \to^*_\mathcal{R} \delta$, then $\delta \in eval_{\varphi'}(\mathtt{unquote}_\tau(\mathtt{quote}_\tau(t)))$.*

Note that, in contrast to Theorem 5.4, we can now start with any E-strategy map $\varphi \in CM_\mathcal{R}$:

Example 6.4 The following OBJ program:

```
obj EXAMPLE-STR is
  sorts Nat LNat .
  ops 0 0' : -> Nat .
  ops s s' : Nat -> Nat [strat (1)] .
  ops nil nil' : -> LNat .
  op cons : Nat LNat -> LNat [strat (1)] .
  op cons' : Nat LNat -> LNat [strat (1 2)] .
  op 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 in Example 1.1. Now, the evaluation of term `unquote'(quote'(first(s(0),from(0))))` yields:

```
Maude> reduce unquote'(quote'(first(s(0), from(0)))) .
reduce in EXAMPLE-STR : unquote'(quote'(first(s(0), from(0)))) .
rewrites: 11 in -10ms cpu (0ms real) (~ rewrites/second)
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:

```
unquote(cons'(X,Z)) = cons(unquote(X),unquote'(Z))
```

in the program in Example 6.4 does not work: the reason is that after applying this rule, *the second argument of* `cons` *remains non-replacing*. For instance, by using such a rule (instead of the last two rules of the program in Example 6.4) the evaluation of `unquote'(quote'(first(s(0),from(0))))` would yield

```
cons(0,unquote'(nil'))
```

This is solved by introducing the intermediate defined symbol `fcons` which first evaluates its arguments (thus performing the renaming) and then reduces to `cons`. In this sense, the *explicit* annotation `(1 2 0)` is also crucial 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 (proved in Example 4.4).

Example 6.5 The evaluation of $t =$ `quote'(from(0))` yields:

```
Maude> reduce quote'(from(0)) .
reduce in EXAMPLE-STR : quote'(from(0)) .

ADVISORY: closing open files.
Debug(1)> Bye.
```

where we were forced to abort the non-terminating execution. Again, the problem is that the evaluation of t, i.e., the evaluation of

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

does not terminate:

$$\begin{aligned}
&\langle \texttt{quote'}_{(1\ 0)}(\texttt{from}_{(1\ 0)}(\texttt{0}_{nil})), \Lambda\rangle\\
&\quad\to_\varphi \langle \texttt{quote'}_{(0)}(\texttt{from}_{(1\ 0)}(\texttt{0}_{nil})), 1\rangle\\
&\quad\to_\varphi \langle \texttt{quote'}_{(0)}(\texttt{from}_{(0)}(\texttt{0}_{nil})), 1.1\rangle\\
&\quad\to_\varphi \langle \texttt{quote'}_{(0)}(\underline{\texttt{from}_{(0)}(\texttt{0}_{nil})}), 1\rangle\\
&\quad\to_\varphi \langle \texttt{quote'}_{(0)}(\texttt{cons}_{(1)}(\texttt{0}_{nil},\texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})))), 1\rangle\\
&\quad\to_\varphi \langle \texttt{quote'}_{(0)}(\texttt{cons}_{nil}(\texttt{0}_{nil},\texttt{from}_{(1\ 0)}(\texttt{s}_{(1\ 0)}(\texttt{0}_{nil})))), 1.1\rangle
\end{aligned}$$

$\to_\varphi \langle \mathtt{quote'}_{(0)}(\mathtt{cons}_{nil}(\mathtt{0}_{nil},\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1\ 0)}(\mathtt{0}_{nil}))),1\rangle$
$\to_\varphi \langle \mathtt{quote'}_{(0)}(\mathtt{cons}_{nil}(\mathtt{0}_{nil},\underline{\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1\ 0)}(\mathtt{0}_{nil}))}),\Lambda\rangle$
$\to_\varphi \langle \mathtt{cons'}_{(1\ 2)}(\mathtt{quote}_{(1\ 0)}(\mathtt{0}_{nil}),\mathtt{quote'}_{(1\ 0)}(\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1\ 0)}(\mathtt{0}_{nil})))),\Lambda\rangle$
$\to^+_\varphi \langle \mathtt{cons'}_{(2)}(\mathtt{0'}_{nil},\mathtt{quote'}_{(1\ 0)}(\mathtt{from}_{(1\ 0)}(\mathtt{s}_{(1\ 0)}(\mathtt{0}_{nil})))),2\rangle$
$\to_\varphi \cdots$

6.1 Preserving completeness and termination

Example 6.5 shows that the annotation $\varphi'(\mathtt{quote}_\tau) = \mathtt{(1\ 0)}$ may cause non-termination. We can try to avoid this problem by restricting the E-strategy for $\mathtt{quote}_\tau$ to `(0)`. In this case, however, we need to add new rules to enable the evaluation in some alternative way. In [16], 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. The new constructors are now introduced in computations by the contraction of redexes of outermost defined symbols f. Thus, we add both 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 $f'(l_1,\ldots,l_k) \to r'$ come from the original ones $f(l_1,\ldots,l_k) \to r$ 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 $\mathtt{quote}_{sort(x)}(x)$ in r'. Now (in contrast to the previous transformation) symbols $\mathtt{quote}_\tau$ 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 into account the number of extra rules which are added to the transformed TRS and introduce new rules $f'(l_1,\ldots,l_k) \to r'$ for each defined symbol f. Unfortunately, this may unnecessarily cause non-termination.

Example 6.6 Consider the rule

```
from(x) → cons(x,from(s(x)))
```

of our running example. We then introduce the rule:

```
from'(x) → cons'(quote(x),from'(s(x)))
```

For example, in the evaluation of t = `first(s(0),from(0))` in Example 5.5, the symbol `from` does *not* emerge as outermost: roughly speaking, the only possibility is that either the right-hand side of a rule defining `first` contains a variable of sort `LNat` having only constructor symbols above, or that `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 reductions are allowed on both arguments of `cons'`. For this reason, we perform a more accurate analysis of the required additional rules by carefully identifying the outermost defined symbols that can emerge during the evaluation of a given expression.

The following notations are auxiliary [16]: Given $f : \tau_1 \times \ldots \times \tau_k \to \tau$, the sorts of arguments of f are gathered in the set [10] $sortarg(f) = \{\tau_1, \ldots, \tau_k\}$. Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,

- $CVar(t) = \{x \in \mathcal{V}ar(t) \mid \exists p \in \mathcal{P}os(t),\ t|_p = x \wedge \forall\ q < p,\ q \in \mathcal{P}os_{\mathcal{C}}(t)\}$ 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 6.7 Consider the term $t = \texttt{cons(y,first(x,z))}$, where $sort(\texttt{y}) = \texttt{Nat}$ and $\texttt{first} : \texttt{Nat} \times \texttt{LNat} \to \texttt{LNat}$. Then,

- $CVar(t) = \{\texttt{y}\}$; note that $sort(\texttt{y}) = \texttt{Nat}$.
- $CVSort(\texttt{Nat}) = \{\texttt{Nat}\}$ and
$$\begin{aligned} CVSort(\texttt{LNat}) &= \{\texttt{LNat}\} \cup \bigcup_{\substack{c \in \{\texttt{nil}, \texttt{cons}\} \\ \tau' \in sortarg(c)}} CVSort(\tau') \\ &= \{\texttt{LNat}\} \cup CVSort(\texttt{Nat}) \cup CVSort(\texttt{LNat}) \\ &= \{\texttt{LNat}, \texttt{Nat}\} \end{aligned}$$
- $Vouter(t) = \{f \in \mathcal{D} \mid sort(f) \in CVSort(sort(\texttt{y}))\} = \{f \in \mathcal{D} \mid sort(f) \in \{\texttt{Nat}\}\} = \{\texttt{sel}\}$.

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: $outrhs_{\mathcal{R}}(f) = \cup_{f(l_1,\ldots,l_k) \to r \in R} \{root(r, p) \mid p \in \mathcal{P}os_{\mathcal{D}}(r) \wedge \forall\ q < p.\ q \in \mathcal{P}os_{\mathcal{C}}(r)\}$.
- $Vrhs_{\mathcal{R}}(f) \subseteq \mathcal{D}$ is the set of outermost defined symbols which can appear by instantiation of constructor variables in rhs's of the f-rules: $Vrhs_{\mathcal{R}}(f) = \cup_{f(l_1,\ldots,l_k) \to r \in R} Vouter(r)$.
- $newouter_{\mathcal{R}}(f) = outrhs_{\mathcal{R}}(f) \cup Vrhs_{\mathcal{R}}(f)$.

[10] Here, we disregard from the ordering of the argument sorts (i.e., we do not use a list of sorts) since it is not important for our purposes.

Example 6.8 Consider the TRS $\mathcal{R}$ in Example 4.1 (assume the sorts as given in the signature of the original OBJ program in Example 1.1). We have:

- $outrhs_{\mathcal{R}}(\mathtt{sel}) = \{\mathtt{sel}\}$ and $outrhs_{\mathcal{R}}(\mathtt{first}) = \{\mathtt{first}\}$. Let us develop the first one: the rules defining `sel` are

  $$\mathtt{sel(0,cons(x,z))} \to \mathtt{x} \text{ and } \mathtt{sel(s(x),cons(y,z))} \to \mathtt{sel(x,z)}.$$

  The *rhs* of the first rule is a variable; hence it does not contribute to $outrhs_{\mathcal{R}}(\mathtt{sel})$. On the other hand, the only outermost defined symbol of the second *rhs* is `sel`; hence, $outrhs_{\mathcal{R}}(\mathtt{sel}) = \{\mathtt{sel}\}$.
- $Vrhs_{\mathcal{R}}(\mathtt{sel}) = Vouter(\mathtt{x}) \cup Vouter(\mathtt{sel(x,z)}) = Vouter(\mathtt{x}) = \{\mathtt{sel}\}$ (note that $sort(\mathtt{x}) = \mathtt{Nat}$) and, according to Example 6.7:

  $$\begin{aligned} Vrhs_{\mathcal{R}}(\mathtt{first}) &= Vouter(\mathtt{nil}) \cup Vouter(\mathtt{cons(y,first(x,z))}) \\ &= Vouter(\mathtt{cons(y,first(x,z))}) \\ &= \{\mathtt{sel}\} \end{aligned}$$

- Finally, $newouter_{\mathcal{R}}(\mathtt{sel}) = outrhs_{\mathcal{R}}(\mathtt{sel}) \cup Vrhs_{\mathcal{R}}(\mathtt{sel}) = \{\mathtt{sel}\}$ and $newouter_{\mathcal{R}}(\mathtt{first}) = outrhs_{\mathcal{R}}(\mathtt{first}) \cup Vrhs_{\mathcal{R}}(\mathtt{first}) = \{\mathtt{first}, \mathtt{sel}\}$.

In contrast to transformation E_τ, here we are mainly interested in evaluating term $f(t_1, \ldots, t_k)$ 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 $\mathcal{D}^f_{\mathcal{R}} \subseteq \mathcal{D}$ be:

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

$\mathcal{D}^f_{\mathcal{R}}$ contains the outermost defined symbols which arise when a (well sorted) f-rooted term $f(t_1, \ldots, t_k)$ is arbitrarily rewritten. In practice, since the definition of $\mathcal{D}^f_{\mathcal{R}}$ is mutually recursive, we must consider all possible equations

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

(where $f_1 = f$ and $f_2, \ldots, f_n$ are all the defined symbols successively occurring in $newouter_{\mathcal{R}}(f_1) \cup \cdots \cup newouter_{\mathcal{R}}(f_n)$) and compute the (least) solutions $\mathcal{D}^{f_1}_{\mathcal{R}}, \ldots, \mathcal{D}^{f_n}_{\mathcal{R}}$ by using fixpoint techniques (see [11,16]).

Example 6.9 (Continuing Example 6.8) Since $newouter_{\mathcal{R}}(\mathtt{sel}) = \{\mathtt{sel}\}$ and $newouter_{\mathcal{R}}(\mathtt{first}) = \{\mathtt{first}, \mathtt{sel}\}$, we have the system:

$$\mathcal{D}^{\mathtt{first}}_{\mathcal{R}} = \{\mathtt{first}\} \cup \mathcal{D}^{\mathtt{sel}}_{\mathcal{R}} \cup \mathcal{D}^{\mathtt{first}}_{\mathcal{R}}$$

$$\mathcal{D}^{\mathtt{sel}}_{\mathcal{R}} = \{\mathtt{sel}\} \cup \mathcal{D}^{\mathtt{sel}}_{\mathcal{R}}$$

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

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) $g(\overline{t}) \in ev^f(\mathcal{F},\mathcal{X})$ if $g \in \mathcal{D}^f_{\mathcal{R}}$, and (3) $c(t_1,\ldots,t_k) \in ev^f(\mathcal{F},\mathcal{X})$ if $c \in \mathcal{C}^*_{sort(f)}$ and $t_1,\ldots,t_k \in ev^f(\mathcal{F},\mathcal{X})$. If we do not require (1) (and change the inductive case (3) to be $c(t_1,\ldots,t_k) \in gev^f(\mathcal{F},\mathcal{X})$ if $c \in \mathcal{C}^*_{sort(f)}$ and $t_1,\ldots,t_k \in gev^f(\mathcal{F},\mathcal{X})$), then we are defining the set $gev^f(\mathcal{F},\mathcal{X})$. Roughly speaking, if we rewrite on a term $t = g(\overline{t})$ for some $g \in \mathcal{D}^f_{\mathcal{R}}$, 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 now define the program transformation. First, we give the new signature. Note that the transformation is parametric w.r.t. a TRS $\mathcal{R} = (\mathcal{F},R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and a defined symbol $f \in \mathcal{D}$.

Definition 6.10 Given a TRS $\mathcal{R} = (\mathcal{F},R) = (\mathcal{C} \uplus \mathcal{D}, R)$ and $f \in \mathcal{D}$, we let $\mathcal{F}^f = \mathcal{F} \uplus \mathcal{C}' \uplus \mathcal{D}' \uplus \mathit{Quote} \uplus \mathit{Unquote}$, where: $c' \in \mathcal{C}' \Leftrightarrow c \in \mathcal{C}^*_{sort(f)} \wedge ar(c') = ar(c)$ and $g' \in \mathcal{D}' \Leftrightarrow g \in \mathcal{D}^f_{\mathcal{R}} \wedge ar(g') = ar(g)$. *Quote* and *Unquote* are as above.

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

Definition 6.11 [Transformation V] Let $\mathcal{R} = (\mathcal{F},R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and $f \in \mathcal{D}$. We let $\mathsf{V}^f(\mathcal{R}) = (\mathcal{F}^f, R \cup S \cup Q \cup U)$, where:

- $S = \{g'(\overline{l}) \to \kappa^f(r) \mid g(\overline{l}) \to r \in R \wedge g \in \mathcal{D}^f_{\mathcal{R}}\}$, where
  $\kappa^f(x) = \mathtt{quote}_{sort(x)}(x)$, for $x \in \mathcal{X}$, $\kappa^f(g(\overline{t})) = g'(\overline{t})$ if $g \in \mathcal{D}^f_{\mathcal{R}}$, and $\kappa^f(c(\overline{t})) = c'(\kappa^f(\overline{t}))$ if $c \in \mathcal{C}$.
- Rules in Q define symbols $\mathtt{quote}_\tau$ in order to rename external constructors $c \in \mathcal{C}^*_\tau$ (where $\tau = sort(f)$) to constructors $c' \in \mathcal{C}'$ where $c, c' : \tau_1 \times \cdots \times \tau_k \to \tau'$, and outermost application of $g \in \mathcal{D}^f_{\mathcal{R}}$ to outermost applications of the corresponding $g' \in \mathcal{D}'$.

$$Q = \{\mathtt{quote}_{\tau'}(c(x_1,\ldots,x_k)) \to c'(\mathtt{quote}_{\tau_1}(x_1),\ldots,\mathtt{quote}_{\tau_k}(x_k)) \mid c \in \mathcal{C}^*_\tau\}$$
$$\cup \{\mathtt{quote}_{sort(g)}(g(x_1,\ldots,x_k)) \to g'(x_1,\ldots,x_k) \mid g \in \mathcal{D}^f_{\mathcal{R}}\}$$

- 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 $\varphi' = \mathit{Emap}^f(\varphi)$ as follows: $\varphi'(g) = \varphi(f)$ if $g \in \mathcal{D}$, $\varphi'(g') = \varphi(g)$ if $g \in \mathcal{D}^f_{\mathcal{R}}$, $\varphi'(c) = \varphi(c)$ if $c \in \mathcal{C}$, and $\varphi(c') = (1\ \cdots\ ar(c'))$ if $c' \in \mathcal{C}'$, $\varphi(\mathtt{quote}_\tau) = (0)$ and $\varphi(\mathtt{unquote}_\tau) = (1\ 0)$ for all sort τ; and $\varphi(f_c) = (1\ \cdots\ ar(c)\ 0)$ for each $c \in \mathcal{C}^*_{sort(f)}$ such that $\mu^\varphi(c) \neq \{1,\ldots,ar(c)\}$.

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

Theorem 6.12 *Let $\mathcal{R} = (\mathcal{F},R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a regular E-strategy map. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F},\mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathsf{V}^f(\mathcal{R})$ and $\varphi' = \mathit{Emap}^f(\varphi)$. If $\delta \in \mathit{eval}_{\varphi'}(\mathtt{unquote}_{sort(t)}(\mathtt{quote}_{sort(t)}(t)))$, then $t \to^*_{\mathcal{R}} \delta$.*

Theorem 6.13 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS. Let φ be a regular E-strategy map. Let $f \in \mathcal{D}$, $t \in ev^f(\mathcal{F}, \mathcal{X})$, and $\delta \in \mathcal{T}(\mathcal{C})$. Let $\mathcal{R}' = \mathsf{V}^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $\delta \in eval_\varphi(t)$, then $\delta \in eval_{\varphi'}(\mathtt{unquote}_{sort(t)}(\mathtt{quote}_{sort(t)}(t)))$.*

Theorem 6.14 *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear, confluent TRS. Let φ be a regular E-strategy map such that $\varphi \in CM_\mathcal{R}$ 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 $\mathcal{R}' = \mathsf{V}^f(\mathcal{R})$ and $\varphi' = Emap^f(\varphi)$. If $t \to^*_\mathcal{R} \delta$, then $\delta \in eval_{\varphi'}(\mathtt{unquote}_{sort(t)}(\mathtt{quote}_{sort(t)}(t)))$.*

Example 6.15 The following OBJ3 program:

```
obj EXAMPLE-TR is
  sorts Nat LNat .
  ops 0 0' : -> Nat .
  ops s s' : Nat -> Nat [strat (1)] .
  ops nil nil' : -> LNat .
  op cons : Nat LNat -> LNat [strat (1)] .
  op cons' : Nat LNat -> LNat [strat (1 2)] .
  op 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 new transformed version of the OBJ program in Example 1.1. Now, the evaluation of `unquote'(quote'(first(s(0),from(0))))` yields:

```
Maude> reduce unquote'(quote'(first(s(0), from(0)))) .
reduce in EXAMPLE-TR : unquote'(quote'(first(s(0), from(0)))) .
rewrites: 10 in -10ms cpu (0ms real) (~ rewrites/second)
result LNat: cons(0, nil)
```

By using the context-sensitive recursive path ordering (CSRPO) of [2] we can even prove termination of the program in Example 6.15.

Example 6.16 Consider again the evaluation of the non-terminating expression `from(0)` using the program in Example 6.15. Now, we obtain:

```
Maude> reduce unquote'(quote'(from(0))) .
reduce in EXAMPLE-TR : unquote'(quote'(from(0))) .
rewrites: 0 in -10ms cpu (0ms real) (~ rewrites/second)
result LNat: unquote'(quote'(from(0)))
```

General conditions under which this second transformation preserves termination of the original program should be further investigated.

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. As there is no standard, commonly accepted terminology, current definitions are rather misleading and we think this may cause an erroneous understanding (e.g., compare the mix of different concepts for the notion of correctness/completeness in [18,19,21]).
- 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 definedness do not.
- We ascertain the conditions (on φ) ensuring that OBJ programs using (positive) strategy annotations do compute the value of any given expression (Theorem 5.4). As shown in Example 5.6, termination of the program (under φ) is essential for achieving correct (and complete) computations.
- Theorem 5.4 requires that all arguments of constructor symbols be replac-

ing. This may incur in unnecessary nontermination. Thus, we have formalized a transformation which can 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 [17]. 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 to ensure 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 [10,12]). 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 directly available in current OBJ implementations. The possibility of achieving a similar effect by using program transformation is a subject of future work.

Acknowledgements.

We would like to thank Cristina Borralleras for providing a proof of termination of the TRS that correspond to the program of Example 6.15 using CSRPO. We also thank the anonymous referees for their helpful remarks.

References

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

[2] 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*, LNAI 2392:314-331, Springer-Verlag, Berlin, 2002.

[3] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. *Electronic Notes in Theoretical Computer Science*, volume 4, Elsevier Sciences, 1996.

[4] S. Eker. Term Rewriting with Operator Evaluation Strategies. *Electronic Notes in Theoretical Computer Science*, volume 15, Elsevier Sciences, 1998.

[5] O. Fissore, I. Gnaedig, and H. Kirchner. Induction for termination with local strategies. *Electronic Notes in Theoretical Computer Science*, volume 58(2), Elsevier Sciences, 2001.

[6] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages, POPL'85*, pages 52-66, ACM Press, 1985.

[7] 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.

[8] 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.

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

[10] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, Academic Press, to appear.

[11] S. Lucas. Rewriting with replacement restrictions. PhD Thesis, DSIC, Universidad Politécnica de Valencia, in spanish, October 1998.

[12] S. Lucas. Termination of (Canonical) Context-Sensitive Rewriting. In Sophie Tison, editor, *Proc. 13th International Conference on Rewriting Techniques and Applications (RTA'02)*, LNCS 2378:296-310, Springer-Verlag, Berlin, 2002.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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, Elsevier Sciences, 2001.

[20] K. Ogata and K. Futatsugi. Implementation of Term Rewritings with the Evaluation Strategy. In H. Glaser and P. Hartel, editors, *Proc of 9th International Symposium on Programming Languages, Implementations, Logics and Programs, PLILP'97*, LNCS 1292:225-239, Springer-Verlag, Berlin, 1997.

[21] J. van de Pol. Just-in-time: on Strategy Annotations. *Electronic Notes in Theoretical Computer Science*, volume 57, Elsevier Sciences, 2001.